Demo Guide

git-ing Started with GitHub

Hands-On Demo

Danielle Vansia
April 2022

A CLOUD GURU
A PLURALSIGHT COMPANY

Contents

Introduction

`git` and GitHub are valuable tools in the tech industry. Knowing both technologies will allow you to contribute to open-source projects. Many open-source projects lack documentation experts, so knowing how to use `git` and GitHub reduces the learning curve when contributing to these projects.

The main goal of this demo is to get you to open your first pull request on GitHub. This demo has the following learning objectives:

- Install and configure `git` on your local system
- Get stated with a GitHub account
- Understand `git`- and GitHub-related terminology
- Understand the various `git` workflows
- Set up `git` remotes (`origin` and `upstream`)
- Create and switch between branches
- Clone, edit, add, and commit files locally
- Open a pull request on GitHub
- Understand how to pull changes from `upstream`

Prerequisites

To get started with using `git` and GitHub, you will need to install a few prerequisites on your system as well as create a GitHub account.

Locally Install `git`

1. From the Privileges application, click **Request Privileges**.
2. Open a Terminal session, and check to see if you have `git` on your system:

   ```
   git --version
   ```

   If `git` is on your system, you will see a version number in the output, **and you do not need to complete the rest of the steps in this section.** If you get an error or a message asking whether you would like to install `git`, then follow the steps below to install it via Homebrew.
3. Install **Homebrew**:

   ```
   /bin/bash -c "$(curl -fsSL https://
   raw.githubusercontent.com/Homebrew/install/HEAD/
   install.sh)"
   ```

   The script will walk you step by step through the installation process.
4. Once Homebrew is installed, install `git`:

   ```
   brew install git
   ```
5. Check if `git` was successfully installed:

   ```
   git --version
   ```

   This time, you should see a version number in the output. If for some reason it does not show the version number, close Terminal, restart, and run the command again.

Create a GitHub Account, and Configure It Locally

1. Go to https://github.com/ to sign up for an account.

2. Verify your email address. Reference the GitHub documentation if you do not receive a verification email.

3. Go back to Terminal, and configure `git` to use your credentials. Replace `Your_Name` with your actual name. Replace `Your_Email` with the email address you used to create your GitHub account (keep the quotes around these items):

```
git config --global user.name "Your_Name"
```

```
git config --global user.email "Your_Email"
```

Create a Personal Access Token

1. From the upper-right corner of your GitHub account, click your user icon and select **Settings**.
2. Scroll down, and on the left, click **<> Developer settings**.
3. From the left menu, select **Personal access tokens**.
4. Click **Generate new token**.
5. For **Note**, enter any name you want.
6. For **Expiration**, choose whatever date you want. Once this token expires, you will have to create a new one.
7. For **Select scopes**, check the box next to **repo**.
8. Click **Generate token**.
9. On the next page, you will see your new Personal Access Token. Copy that token to a text file — you will be unable to retrieve the token again after this screen. When you push files to GitHub from your machine, you will need to use this token when prompted for a password. Have the token handy at that time.

Terminology

When working with `git` and GitHub, you will come across a lot of new terminology. The following terms are some of the most common terms you will see:

Term	Definition
`add`	Moves a file to the *staging* space to prepare it for a commit.
Branch	Think of a tree. Repositories start with a `main` branch (historically referred to as `master`). As you continue to develop, you can create new branches for features or fixes. Branching allows you to independently work on a new feature without adding it to the original codebase until you are ready. You can experiment within a branch without having to worry about breaking the original code.
`clone`	Creates a local copy of a repository on your system.
Commit	A snapshot of a point in time where your changes are saved. Commits contain a message indicating what changed.
Diff	Compares your working copy of a file to the original version of the file.
Fork	A copy of someone else's repository, saved to your account. This is a GitHub/GitLab concept — not a concept related to the actual `git` software.
`origin`	This is the location of the repository on your GitHub account. When you

Term	Definition
	fork a repository, the forked repository becomes the `origin`.
`git`	Version-control software used to track file changes. `git` enables multiple developers to work together simultaneously and track file updates. `git` directories form repositories. There are a number of companies that offer Repository-as-a-Service functionality, such as GitHub, GitLab, and Bitbucket.
`push`	Uploads your local changes to a remote repository.
`pull`	Fetches and updates content from a remote repository.
Pull request (PR)	A request to merge your changes with a remote repository.
Repository (also repo)	Contains all project files with a history of all changes and commits.
Remote	Original versions of the repository, hosted on another server. `upstream` and `origin` are both remotes.
`upstream`	This is the original location of the repository you forked.

Fork and Clone Repos

When you are working with repos, you have the option to directly clone the repo and work on it locally, or you can fork the repo and then clone it.

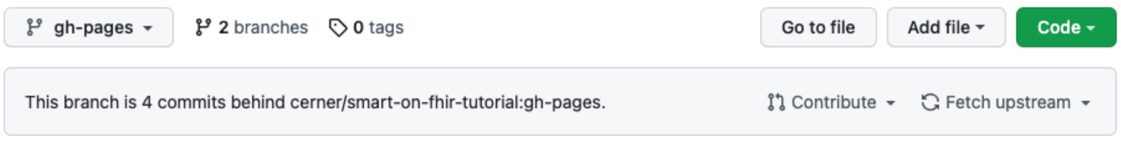
- **Fork, clone, work locally**: You would typically use this method when you are working with open-source projects where you do not have direct access or permissions to update the source repo. You can make changes without affecting the original repo. You can delete forked repos without deleting the original repo.
- **Clone, work locally**: Use this workflow when working on your own projects. Create a repo on GitHub, and clone it to your system to work locally. You could also use this workflow if you are given direct access to a repo at a company.

Fork a Repo

1. Go to the GitHub repo (e.g., `https://github.com/vansia43/github-git-demo`).
2. At the top, click **Fork**.
3. If prompted, select your username as the location you want to fork to.
4. You will have to option to update the **Repository name**; however, keep it the same as the original repo if you are going to later open a pull request.
5. Click **Create fork**.

This will create a copy of the repo in your GitHub account. The URL for the forked repo will include your username (i.e., `github.com/your_username/original_repo_name.git`).

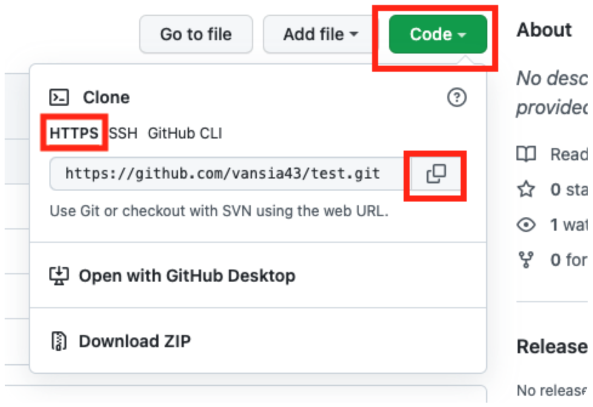
On your forked repo, you can see a how the current version of your repo compares to the original repo. When you first fork it, your version should be the same.



This branch is 4 commits behind cerner/smart-on-fhir-tutorial:gh-pages.

Clone a Repo

1. From your forked repo on GitHub, click **Clone**.
2. Ensure **HTTPS** is selected.
3. Copy the link using the copy icon.



```
https://github.com/vansia43/test.git
```

Use Git or checkout with SVN using the web URL.

4. Open a Terminal session.
5. Change directory to the folder where you want your cloned repo to reside (e.g., `Documents`):

```
cd Documents
```

6. Run the `clone` command, and paste the link to the forked repo:

```
git clone https://github.com/your_username/repo_name.git
```

Note: You may be prompted to enter your username and password. Enter your GitHub username, and use your Personal Access Token as the password.

7. You should now have a local copy of the GitHub repo on your computer. You can navigate to the main repo folder in Finder and view the files, or you can change directory to access the repo in Terminal:

```
cd github-git-demo
```

Configure Your Local Repo

This section will discuss how to connect your local repo to remotes (`origin` and `upstream`) as well as how to create and check out branches.

Add Remotes

Add remotes to push changes back to your forked copy on GitHub (the `origin`) as well as pull updates from the original repo (the `upstream`).

1. Check for existing remotes:

   ```
   git remote -v
   ```

   If you cloned your repo directly from GitHub, then you will most likely see the URL for your forked copy in the output. It will look something like this:

   ```
   origin  https://github.com/your_username/repo_name.git
   (fetch)
   origin  https://github.com/your_username/repo_name.git
   (push)
   ```

2. Add the original/`upstream` repo so you can fetch and pull changes later on. You can obtain this URL by going to the original repo and clicking the **Clone** button:

   ```
   git remote add upstream https://github.com/original_owner/
   original_repo_name.git
   ```

Create and Check Out Branches

When you are working on a project in a repo, you start with a `main` branch. (You may see this historically referred to as `master`.) As you work on projects, also called features, you should create a new branch (e.g., `dev` or `doc-updates`) that tracks the work for each feature. The new branch will mirror your `main` branch until you start making changes. Once you commit your changes to your feature branch, push them to GitHub, and open a pull request, your feature branch will merge with the original repo's `main` branch.

1. Check for existing branches:

   ```
   git branch
   ```

   If you have no other branches, you should see `* main` in the output.

2. To create a new branch that mirrors the branch you are currently on and then switch to that new branch, use:

   ```
   git switch -c branch_name
   ```

   Replace `branch_name` with something descriptive to represent what you are working on. You should see `Switched to a new branch 'branch_name'` in the output.

   > **Note:** This is a newer command. If you have an older version of `git`, you can use `git checkout -b <branch_name>`.

3. To move between branches, use:

   ```
   git switch <branch_name>
   ```

   > **Note:** This is a newer command. If you have an older version of `git`, you can use `git checkout <branch_name>`.

4. Check the status of `git`:

   ```
   git status
   ```

   You should see `On branch branch_name` and `nothing to commit, working tree clean` in the output (if you have not yet made any changes). **Ensure that you have switched to your feature branch before making changes to the files.**

Add, Commit, and Push Files

Now that you have created a feature branch, you can edit and update files within the repo. You can make file changes using any GUI-based editor, such as Atom, VSCode, etc. You can also edit files directly in Terminal using Vim, Nano, etc.

Add Files to the Staging Space

1. Once you have saved your work, check the status of `git` using `git status`. You should see something similar to this in the output:

   ```
   On branch branch_name
   Changes not staged for commit:
     (use "git add <file>..." to update what will be
   committed)
     (use "git restore <file>..." to discard changes in
   working directory)
             modified:   file.md

   no changes added to commit (use "git add" and/or "git
   commit -a")
   ```

   Observe that `file.md` was edited.

2. Add the updated files to the *staging* space:

   ```
   git add .
   ```

   The `.` at the end of the command means that you are adding *all* files you edited. If you want to add a single file, use `git add` with the file name (i.e., `git add file.md`).

3. You should see something like this in the output:

   ```
   On branch branch_name
   Changes to be committed:
     (use "git restore --staged <file>..." to unstage)
             modified:   file.md
   ```

   This means that your file is now in the staging space and is ready to be committed.

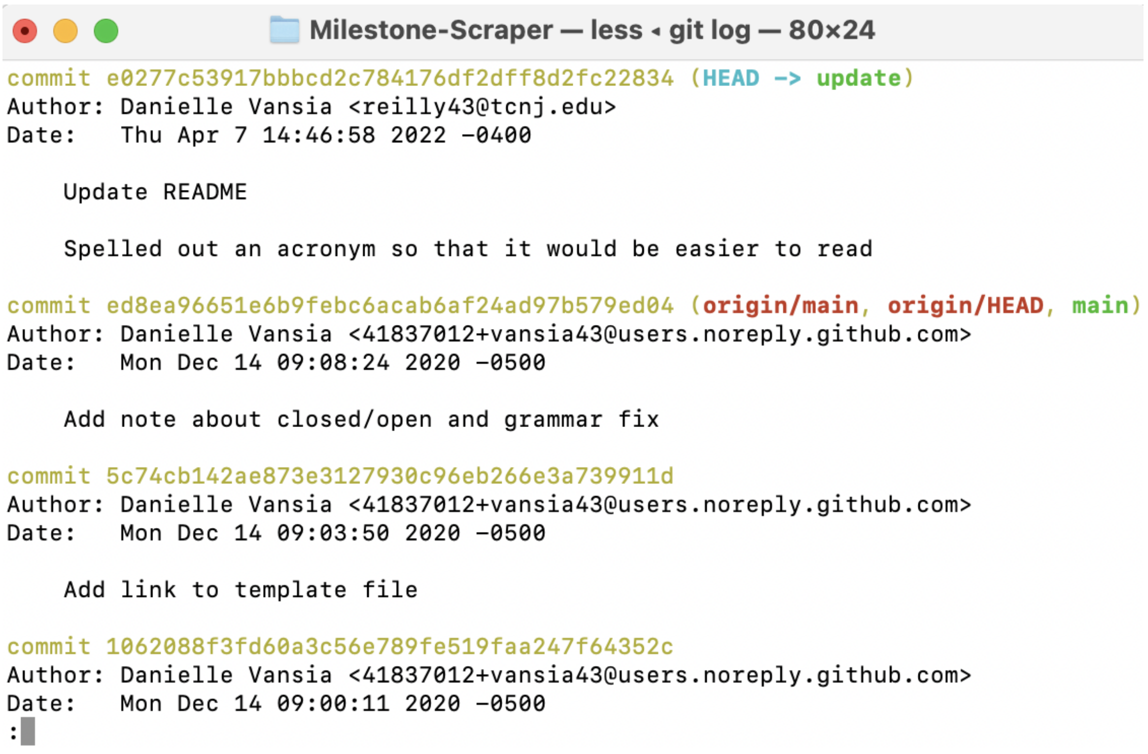
Commit Files

In `git`, you can view a history of commits:

```
git log
```

This will provide you with a history of the changes that were made.

The output will look like this:



Milestone-Scraper — less ◂ git log — 80×24

```
commit e0277c53917bbbcd2c784176df2dff8d2fc22834 (HEAD -> update)
Author: Danielle Vansia <reilly43@tcnj.edu>
Date:   Thu Apr 7 14:46:58 2022 -0400

    Update README

    Spelled out an acronym so that it would be easier to read

commit ed8ea96651e6b9febc6acab6af24ad97b579ed04 (origin/main, origin/HEAD, main)
Author: Danielle Vansia <41837012+vansia43@users.noreply.github.com>
Date:   Mon Dec 14 09:08:24 2020 -0500

    Add note about closed/open and grammar fix

commit 5c74cb142ae873e3127930c96eb266e3a739911d
Author: Danielle Vansia <41837012+vansia43@users.noreply.github.com>
Date:   Mon Dec 14 09:03:50 2020 -0500

    Add link to template file

commit 1062088f3fd60a3c56e789fe519faa247f64352c
Author: Danielle Vansia <41837012+vansia43@users.noreply.github.com>
Date:   Mon Dec 14 09:00:11 2020 -0500
:
```

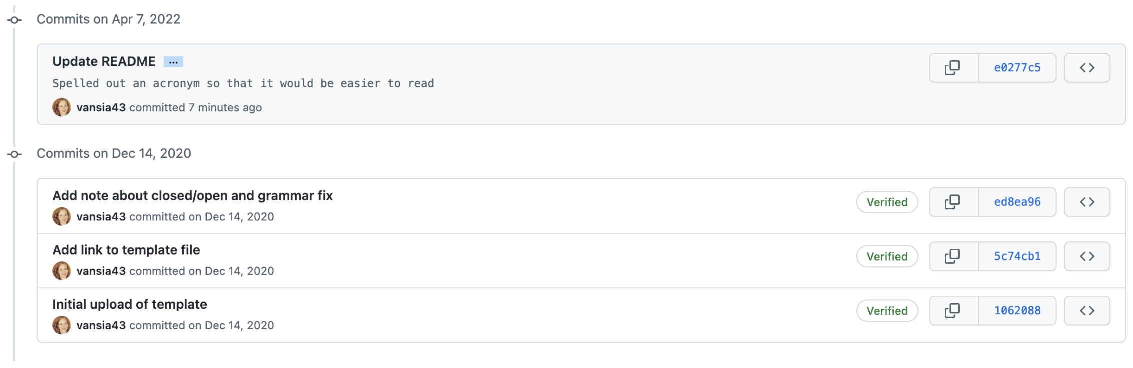
Observe the first commit has two lines of text. The first line is the *summary line*, which should say why you made the change. The second line of commit text is an explanation. You can make a commit with only one line, but if you want to provide a detailed explanation of why you made the change or update, you should make a multi-line commit.

The first line of a commit message should follow these parameters:

- Is 50 characters or less
- Uses the imperative mood (e.g., "Fix grammar and typos" or "Add new section for troubleshooting")
- Provides a good, quick summary of the change
- Does not include a period at the end of the line

See the *Additional Resources* section of this guide for some more tips on best practices when writing commits.

In GitHub, the commit history looks like this:



Commits on Apr 7, 2022

Update README ...
Spelled out an acronym so that it would be easier to read
vansia43 committed 7 minutes ago — e0277c5

Commits on Dec 14, 2020

Add note about closed/open and grammar fix
vansia43 committed on Dec 14, 2020 — Verified — ed8ea96

Add link to template file
vansia43 committed on Dec 14, 2020 — Verified — 5c74cb1

Initial upload of template
vansia43 committed on Dec 14, 2020 — Verified — 1062088

Notice how the second explanation line is accessed via the ellipses (...). The first commit line needs to be short so that when other developers review the commit history for the repo, they can view a quick synopsis of the changes without the text being cut off.

Use the following command to commit a file with a single-line message (ensure it's wrapped in double quotes):

```
git commit -m "Commit short message"
```

Use the following command to commit a file with a multi-line message (ensure both messages are wrapped in double quotes):

```
git commit -m "Commit short message" -m "Longer message
description"
```

Push Commits to GitHub

Now that you have committed your changes, it is time to push them back up to GitHub. Pushing your changes will add them to your GitHub `origin` repo.

To push changes to the `origin`, use:

```
git push origin branch_name
```

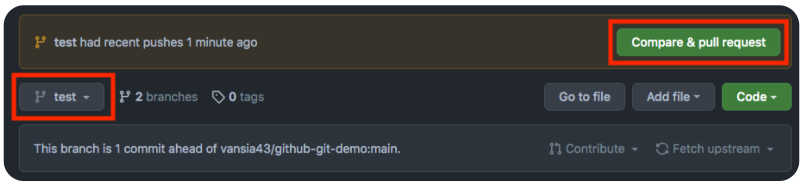
Replace `branch_name` with the name of your feature branch.

This command will add your changes — as well as your new feature branch — to your GitHub account. You may be prompted to enter your username and password. Use your GitHub username and Personal Access Token for your password.

Open a Pull Request (PR)

Now you are ready to open a pull request — or PR — on GitHub. A PR is a request to merge your changes with the `upstream` repo's `main` branch. A reviewer will look at your requested changes, potentially request additional changes, and then merge them.

1. Go back to your forked copy of the repo in your GitHub account, and use the dropdown above the file list to select your feature branch. (It may default to `main`.)

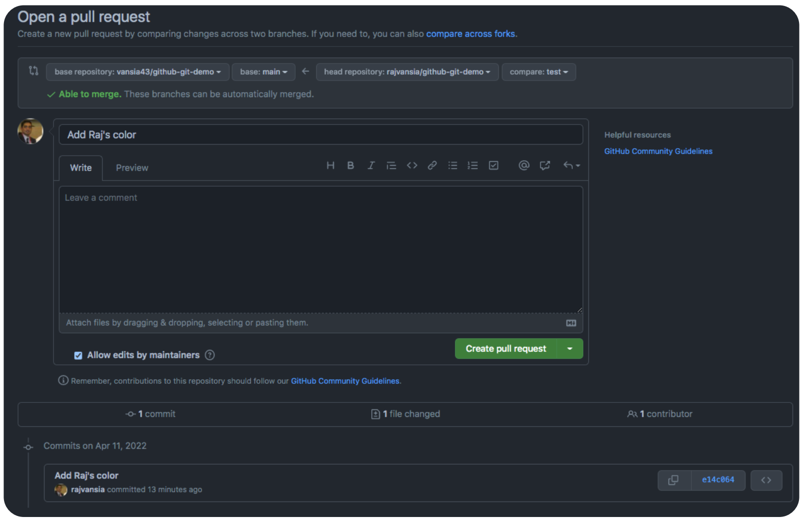


test had recent pushes 1 minute ago

This branch is 1 commit ahead of vansia43/github-git-demo:main.

2. At the top, you'll see a message indicating your branch has recent pushes. Click **Compare & pull request** to start a PR.

3. The UI will let you know if there are any conflicts between your feature branch version and the `upstream` repo. Once you see an **Able to merge** message, you can open a PR.



Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.

base repository: vansia43/github-git-demo · base: main ← head repository: rajvansia/github-git-demo · compare: test

✓ Able to merge. These branches can be automatically merged.

Add Raj's color

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

[✓] Allow edits by maintainers

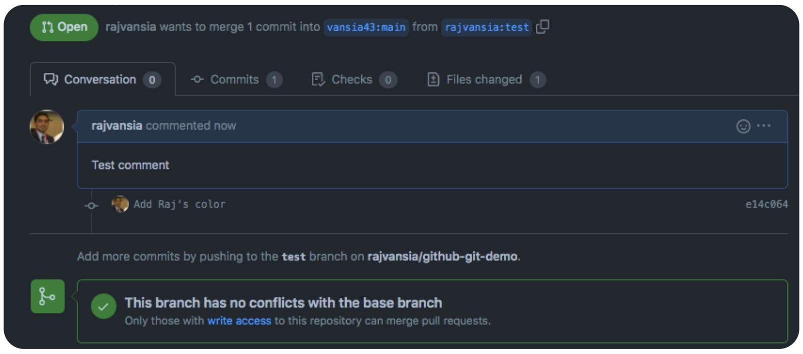
Remember, contributions to this repository should follow our GitHub Community Guidelines.

1 commit · 1 file changed · 1 contributor

Commits on Apr 11, 2022

Add Raj's color
rajvansia committed 13 minutes ago · e14c064

4. Next to your user icon, enter a PR title and a description. For the description, you can repeat your commit message, or you can enter something more detailed. This message will not show in the `git log` history, so it is best to use this description to facilitate chat or reference if this is solving a ticket/issue in GitHub.


Open rajvansia wants to merge 1 commit into `vansia43:main` from `rajvansia:test`

rajvansia commented now

Test comment

`Add Raj's color` e14c064

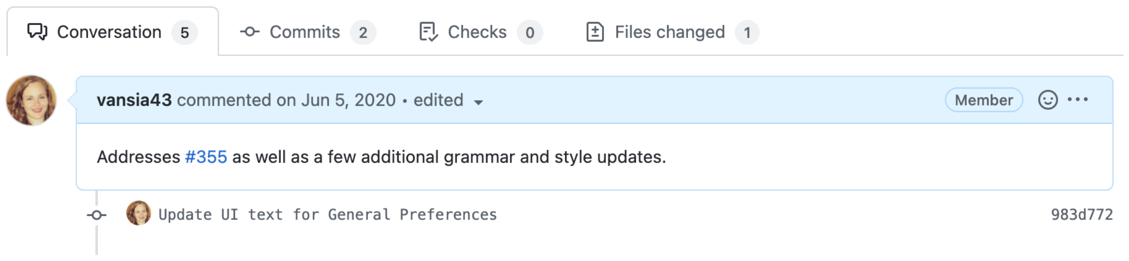
Add more commits by pushing to the **test** branch on **rajvansia/github-git-demo**.

This branch has no conflicts with the base branch
Only those with write access to this repository can merge pull requests.

5. Click **Create pull request**.

Review the Diff

After you create your PR, a reviewer will review your changes and look at the *diff*. Open your PR, and observe a list of commits. In the example below, there is one commit: "Update UI text for General Preferences". The comment above the commit was a comment that was directly added in the GitHub UI. Each commit links to the updated file and will give a visual history of what changed.

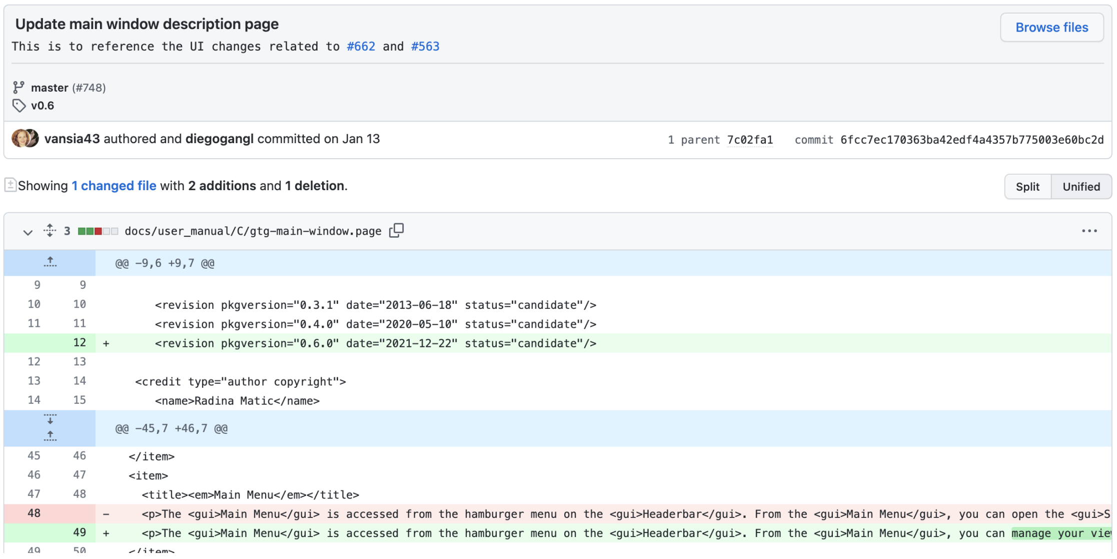


vansia43 commented on Jun 5, 2020 • edited

Addresses #355 as well as a few additional grammar and style updates.

`Update UI text for General Preferences` 983d772

When viewing the commit in GitHub, observe line deletions in red and line additions in green. Reviewers will look at this diff to determine what changed before merging your PR. They may also review your work locally and test it.


Update main window description page
`This is to reference the UI changes related to #662 and #563`

master (#748)
v0.6

vansia43 authored and **diegogangl** committed on Jan 13 — 1 parent 7c02fa1 — commit 6fcc7ec170363ba42edf4a4357b775003e60bc2d

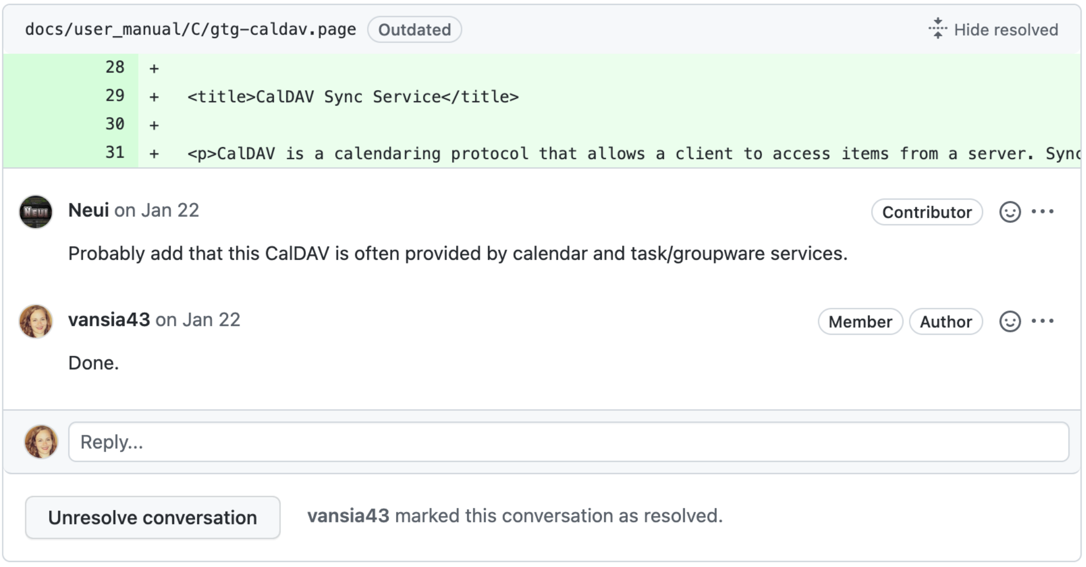
Showing 1 changed file with **2 additions** and **1 deletion**.

```
3 docs/user_manual/C/gtg-main-window.page
@@ -9,6 +9,7 @@

       <revision pkgversion="0.3.1" date="2013-06-18" status="candidate"/>
       <revision pkgversion="0.4.0" date="2020-05-10" status="candidate"/>
+      <revision pkgversion="0.6.0" date="2021-12-22" status="candidate"/>

    <credit type="author copyright">
       <name>Radina Matic</name>
@@ -45,7 +46,7 @@
   </item>
   <item>
     <title><em>Main Menu</em></title>
-    <p>The <gui>Main Menu</gui> is accessed from the hamburger menu on the <gui>Headerbar</gui>. From the <gui>Main Menu</gui>, you can open the <gui>S
+    <p>The <gui>Main Menu</gui> is accessed from the hamburger menu on the <gui>Headerbar</gui>. From the <gui>Main Menu</gui>, you can manage your vie
```

If you are signed up to receive GitHub emails and notifications, you will get an email letting you know that your PR was merged. Reviewers may also chat with you via GitHub in the comments or identify lines of code where they might change something.



docs/user_manual/C/gtg-caldav.page Outdated

```
+
+   <title>CalDAV Sync Service</title>
+
+   <p>CalDAV is a calendaring protocol that allows a client to access items from a server. Sync
```

Neui on Jan 22

Probably add that this CalDAV is often provided by calendar and task/groupware services.

vansia43 on Jan 22

Done.

vansia43 marked this conversation as resolved.

All of the collaboration and comment work can occur in GitHub, but if you need to make changes, you will need to go back to your local copy and make additional commits.

Note: As long as you continue to work on the same feature branch on your local copy, any additional commits you push to GitHub will be added to your PR.

Pull Changes and Update Your Repo

This demo has only scratched the surface on how to work with `git` and GitHub. Once your PR is submitted and merged, your work becomes part of the original project. If you are going to keep contributing to the project, you will need to pull all of the other changes that were made between the time you cloned the repo and now.

After you make your first PR, your `main` branch becomes outdated because it doesn't contain the changes you made on your feature branch, but it also doesn't contain changes that may have been added to the `upstream` repo while you were working. Follow the steps below to pull all `upstream` changes and update your repo both locally and on GitHub.

Important Note: Many organizations and companies have a specific method for pulling changes and updating local copies of repos. The below workflow is one method for doing this; however, some organizations may require that you `rebase` instead of `pull`. After you complete the below steps, look at the `git log`. You will see an additional *merge commit*, and many developers prefer a "clean" history — without this commit. The `rebase` method goes beyond the scope of this demo. Refer to the *Additional Resources* section for information on advanced `git` options.

1. To update your local copy, first switch to the `main` branch:

```
git switch main
```

2. Pull the `upstream` changes:

```
git pull upstream main
```

`git pull` is a combination of two commands — `git fetch` and `git merge`. With `git pull`, you are fetching the updates from the `upstream` repo and then merging them into the `main` branch of your local copy.

3. Delete your old feature branch:

```
git branch -d <branch_name>
```

4. Update the `main` branch on your forked GitHub repo:

   ```
   git push origin main
   ```

5. Delete your old feature branch from GitHub:

   ```
   git push --delete origin <branch_name>
   ```

   Now you can create and switch to a new feature branch as well as work on the next feature or bug.

Additional Resources

There is plenty more to learn about `git`. There are numerous resources online, and we have some great courses on Pluralsight if you want to dive deeper into some advanced topics.

- The ***Managing Source Code with Git*** path on Pluralsight provides you with multiple courses to learn more about some of the complex features of `git`. You can take a Skills IQ test to see how much you know about `git`.
- GitHub also offers a bunch of **free, hands-on courses through their Learning Lab**. The Learning Lab will create repos in your GitHub account for you to experiment and work with.
- The official **GitHub** and **GitLab** documentation is very useful for learning about these services. Many of the basic features of both services are the same, so once you learn one, it's not too hard to learn the other.
- This article provides a lot of useful information for **writing good commit messages**.
- We used HTTPS for cloning repos and then pushed via HTTPS with a Personal Access Token. **Pulling and authenticating with SSH** requires some extra steps, but some organizations may require that you use this method instead.

Contribute to Open-Source Projects

The best way to get practice on GitHub is to contribute to open-source projects. There are a number of projects out there that you can jump in and contribute to. Many projects have a contributor's manual, which will give you information on the project's development workflow as well as how to open a PR or respond to issues.

- **r/technicalwriting** has a great list of open-source projects that are always looking for documentation volunteers.
- You can also perform **advanced searches on GitHub** to look for issues with the tags `documentation` and `good-first-issue`. The `good-first-issue` tag is typically reserved for small fixes that allow you to dive in and help with an existing project. You can also search by the `help-wanted` tag as this is reserved for open-source volunteers.