GitHub Workflow Skill Building

Vanessa Quintana

2025-06-19

Table of contents

Preface

Welcome to **GitHub Workflow Skill Building**, a practical tutorial designed to help you develop confidence using Git and GitHub through real-world tasks. Whether you're an independent researcher, student, or team collaborator, this guide will help you navigate version control effectively using tools like **Git GUI**, **VSCode**, and **RStudio**.

This book walks you through the GitHub workflow step-by-step, with separate chapters for each tool and task. You'll learn not just the "how," but also the "why" behind key practices in collaborative version control.

In these pages, you'll learn how to:

- **Clone** repositories using the browser, **Git GUI**, **VSCode**, and **RStudio**
- **Create** new local repositories and connect them to GitHub
- **Fork** other people's projects and contribute using pull requests
- **Resolve** version conflicts and recover lost or stashed changes
- **Troubleshoot** common problems with Git and GitHub
- **Maintain** a healthy repository through everyday best practices

Each chapter is structured around learning objectives, visual instructions, video walkthroughs, and clear troubleshooting tips. You'll find code snippets and guidance tailored to the platform you prefer to work in, whether you're coding in R, Python, or just managing documentation.

By the end of this book, you'll be able to:

- Start and maintain your own GitHub repositories
- Confidently contribute to others' projects using forks and branches
- Prevent and resolve version issues
- Use GitHub to support reproducible, collaborative, and well-documented work

About the Author

I am an ecological modeler with a focus on aquatic ecosystems, collaborative science, and accessible tools for interdisciplinary work. I created this tutorial to support learners from all backgrounds, whether you are new to version control or looking to build confidence across different platforms. My goal is to make GitHub feel approachable and useful, especially for those working in research, data analysis, or team-based projects. I regularly use Git, R, and Python in my own workflows and hope this guide helps you feel more empowered in yours.

1 Introduction to GitHub

GitHub is a platform that supports collaborative work and version control, making it easier for individuals and teams to manage changes in a project over time. It is built on Git, a system that tracks edits, enables branching, and helps users integrate changes without losing work or overwriting each other.

At its core, GitHub allows users to:

- Track changes across project files
- Safely test new ideas using branches
- Propose and review edits before they're added to a shared project
- Work in parallel with others while keeping a complete history of updates

While GitHub is commonly used in software development, it is just as valuable for researchers, analysts, students, and interdisciplinary teams working with code, data, or shared documentation.

1.1 Why Use GitHub?

Benefit	What It Enables
Version tracking	Maintains a full history of what changed and when
Safe experimentation	Allows testing and editing without disrupting the main work
Collaboration	Supports shared contributions across individuals or teams
Transparency	Provides clear records of authorship and revision details
Integration	Works with tools like RStudio, VSCode, and GitHub Desktop

Using GitHub keeps your work organized and traceable, whether you're collaborating on a project, contributing to open-source software, or managing a professional workflow.

1.2 Key Concepts and Commands

Repository (Repo)

A repository is a project folder that contains your files and their version history. Everything related to the project (e.g., code, documents, presentations, models, and applications) can be stored in a repository.

There are size limitations to keep in mind:

- Individual files should be kept under 100 MB
- Repositories should stay under 1 GB for best performance
- For larger files such as datasets or media, GitHub recommends using Git Large File Storage (LFS)

Repositories can be public (anyone can view or contribute) or private (access is restricted to selected collaborators). On a GitHub team or organization page, repositories can be managed by multiple team members, each with assigned roles and permissions. This setup makes it easy for groups to collaborate on shared projects, assign tasks, and track progress in one place.

Clone

Cloning means creating a local copy of a GitHub repository so you can work on it from your own computer.

Fork

Forking copies someone else's repository to your own GitHub account. It's typically used when you want to propose changes without editing the original project directly.

Branch

A branch is an independent workspace for editing or adding features. You can create and work on a branch without affecting the main project.

Commit

A commit is a snapshot of your changes, often grouped with a short message that describes what you did.

Push

Pushing sends your committed changes from your local machine to GitHub.

Pull Request (PR)

A pull request is a formal request to merge your changes into another branch or repository. This is how collaboration, review, and integration happen on GitHub.

Merge

Merging takes the changes from one branch and integrates them into another once it's completed and reviewed.

When to Clone vs. When to Fork

Understanding whether to **clone** or **fork** a repository depends on how you plan to interact with the project:

Situation	Action	Why
You are working on **your own project** (e.g., creating a personal website, analysis, or codebase)	**Clone**	You are the owner and want a local copy of your own repository to edit and manage
You are part of a **team or organization** and contributing directly to a shared repository	**Clone**	You have direct access and permissions, so you can push updates to the main repo
You want to **suggest edits to someone else's public project** but do **not** have write access	**Fork**	Forking lets you copy the project to your account, make changes, and submit a pull request without affecting the original

Situation	Action	Why
You want to **use another repository as a template** for your own version	**Fork** (or "Use this template")	Forking preserves project history and attribution, giving you a starting point with full editing rights

- **Clone** when you're working on a repository you own or have permission to contribute to directly
- **Fork** when you want to contribute to someone else's project or experiment without changing the original

Both actions can help you get a local copy of the repository, but **forking** creates a separate relationship to the original project with a clear record of where it came from.

1.3 Summary

GitHub helps you:

- Maintain clean, version records of your work

- Collaborate without losing progress or conflicting with others

- Contribute to shared projects in a structured and transparent way

Throughout this tutorial, you'll learn how to apply these concepts using tools like Git GUI, RStudio, and VSCode for working on your own or with a team.

2 Best Practices to Maintain a GitHub Repository

Learning Objectives

- Understand habits that improve collaboration and reduce version control errors
- Learn effective practices for managing changes in GitHub repositories
- Apply consistent workflows for using Git across different platforms

Why Do Best Practices Matter?

Using Git and GitHub effectively is more than just committing and pushing code. It involves managing your repository in a way that keeps it organized, understandable, and collaborative. These practices apply whether you are working alone or with a team.

Daily Workflow Best Practices

1. Pull Before You Start

Always begin your work session by pulling the latest changes:

```
git pull
```

This ensures your local files are up-to-date with the remote repository and helps avoid merge conflicts later.

2. Use Meaningful Commit Messages

Each commit should explain what and why you made changes:

```
git commit -m "Fix typo in data summary and update README"
```

Avoid vague messages like “update” or “fix.”

3. Push Before You Leave

Before ending your work session, commit any final changes and push to GitHub:

```
git add . git commit -m "Wrap up session and push changes" git push
```

This keeps your work synced and backed up in the remote repository.

Organizational Practices

- **Use a clear README**: Include instructions for how to run the project, data sources, dependencies, and project purpose.
- **Maintain file structure**: Use directories (e.g., `/data`, `/scripts`, `/docs`) to keep files organized.
- **Document dependencies**: Use `requirements.txt` (Python) or `DESCRIPTION` (R) to list required packages.
- **Use issues and pull requests**: Track bugs, suggestions, and feature development using GitHub Issues. Use Pull Requests (PRs) to review and integrate changes.

Collaboration Tips

- **Branch for Features or Fixes**: Create a new branch for each major feature or change:

  ```
  git checkout -b feature-login-form
  ```

- **Review Before Merge**: Always review changes in a Pull Request before merging into `main` or `master`. Leave comments and suggestions when working with others.
- **Avoid Committing Sensitive Files**: Use a `.gitignore` file to exclude data files, credentials, or temporary files that shouldn't be tracked.

By following these habits, you can prevent conflicts, improve collaboration, and make your GitHub projects easier to maintain over time.

3 Cloning a Repository through a Web Browser

Learning Objectives

- Understand what it means to clone or download a GitHub repository
- Successfully copy a repository to your local machine from your browser using the GitHub website
- Identify common errors and learn how to fix them

Step-by-Step Instructions

1. Open the repository you want to copy on GitHub.
2. Click the blue **Code** button.
3. Select **Download ZIP**.
4. Save and unzip the file to your computer.
5. This option gives you the files but does not connect to version control.

Video Tutorial

Tutorial_Videos/Clone_Browser/01_Download_Existing_Repo.mp4

Troubleshooting

- **Authentication failed:** Use a personal access token (PAT) if prompted for a password.
- **Downloaded ZIP doesn't update:** ZIP files are static and will not reflect future changes to the repository.

Additional Resources

- GitHub Docs: Cloning a repository
- Happy Git with R

4 Cloning a Repository with Git GUI

Learning Objectives

- Understand what it means to clone or download a GitHub repository
- Successfully copy a repository to your local machine using the GitHub GUI
- Identify common errors and learn how to fix them

Step-by-Step Instructions

1. Open Git GUI.
2. Click "**Clone Repository**".
3. Paste the repository URL into the **Source Location** field.
   Example: `https://github.com/vmahan1998/GitHub_Basic_Skills.git`
4. Choose a **Target Directory** where the local copy should be saved.
5. Name the directory (typically the same as the repo name "**GitHub_Basic_Skills**".
6. Click "**Clone**". Git GUI will create a local copy of the repository on your machine.

Video Tutorial

Tutorial_Videos/Clone_Git_GUI/01_Clone_Git_GUI.mp4

Troubleshooting

- **Authentication failed:** Git GUI may still ask for a GitHub username and password, which no longer work with HTTPS. Use a personal access token (PAT) instead. You can generate one at https://github.com/settings/tokens.
- **Nothing happens after cloning:** Double-check the Source URL and make sure it's spelled correctly. Also, confirm that the folder you selected as the Target Directory is empty or available.
- **Can't see the files in File Explorer:** Git GUI only manages version control. You'll need to open the folder using your file browser or another tool like VSCode to view and edit files.

Additional Resources

- GitHub Docs: Cloning a repository
- GitHub Docs: About Git GUI Clients
- Git for Windows (includes Git GUI)

5 Cloning a Repository with VSCode

Learning Objectives

- Understand what it means to clone or download a GitHub repository
- Successfully copy a repository to your local machine using VSCode
- Identify common errors and learn how to fix them

Step-by-Step Instructions

1. Open Visual Studio Code (VSCode).
2. On the **Welcome page**, click **Clone Git Repository**.
    - If you don't see this option, install the **GitHub Extension** from the VSCode marketplace.
3. Paste the repository URL:
   Example: `https://github.com/vmahan1998/GitHub_Basic_Skills.git`
4. Choose a local folder where the repository should be saved.
5. Name the directory (typically the same as the repo name "**GitHub_Basic_Skills**").
6. When prompted, click "**Open"** to begin working inside the cloned repository.

Video Tutorial

Tutorial_Videos/Clone_VSCode/01_Clone_Repo.mp4

Troubleshooting

- **Authentication failed:** If prompted for login credentials, use a personal access token (PAT) instead of a password. Generate one at https://github.com/settings/tokens.
- **Git not installed:** Make sure Git is installed on your computer and that VSCode can detect it. You can download Git from https://git-scm.com.
- **Extension missing:** Install the GitHub or Git integration extension from the Extensions Marketplace.

Additonal Resources

- GitHub Docs: Cloning a repository
- Happy Git with R
- Using Git in VSCode
- Git for Windows

6 Cloning a Repository with RStudio

Learning Objectives

- Understand what it means to clone or download a GitHub repository
- Successfully copy a repository to your local machine using RStudio
- Identify common errors and learn how to fix them

Step-by-Step Instructions

1. Open RStudio.
2. Go to **File > New Project > Version Control > Git**.
3. Paste the URL for the repository you want to clone:
   Example: `https://github.com/vmahan1998/GitHub_Basic_Skills.git`
4. Choose a directory where you want to store the project.
5. Name the directory (typically the same as the repo name "**GitHub_Basic_Skills**").
6. Click **Create Project**. RStudio will open the cloned repository in a new project workspace.

Video Tutorial

Tutorial_Videos/Clone_RStudio/01_Clone_RStudio.mp4

Troubleshooting

- **Authentication failed:** RStudio may prompt you for GitHub credentials. If using HTTPS, you must enter a personal access token (PAT) instead of your password. You can create one at https://github.com/settings/tokens.
- **Git not configured in RStudio:** If Git is not detected, go to **Tools > Global Options > Git/SVN** and set the path to your Git executable (usually `git.exe`).
- **Missing Git option in New Project menu:** Install Git for Windows or macOS and restart RStudio.
- **Project won't open or clone fails:** Double-check the URL and ensure that you selected the correct folder and have internet access.

Additional Resources

- GitHub Docs: Cloning a repository
- Happy Git with R
- RStudio Git Integration
- Git for Windows

7 Creating a New Repository with Git GUI

Learning Objectives

- Understand what it means to create a new repository on GitHub
- Successfully create a repository in GitHub using Git GUI
- Identify common errors and learn how to fix them

Step-by-Step Instructions

1. Open Git GUI.
2. Click **Create New Repository**.
3. Choose or create a folder where the new repository will be initialized.
4. Git GUI will create an empty repository in that folder. Open the folder in a file browser and add your project files or create a new file (e.g., `README.md`).
5. To push the new repository to GitHub:
   - First, create a new repository on GitHub.com without initializing it with a README.
   - Back in Git GUI, go to **Remote > Add** and paste the GitHub repo URL.
   - Set the remote name to `master`.
   - Go to **Remote > Push** to upload your local repository to GitHub.
6. In Git GUI, click "**Rescan**" to see added files.
7. Stage changes by clicking "**Stage Changed**".
8. Write a commit message (e.g., "Initial commit") and click "**Commit**".

Video Tutorial

Create the Repository

Tutorial_Videos/Create_Repo_Git_GUI/01_Create_Repo_Git_GUI.mp4

Initialize the Repository on GitHub

Tutorial_Videos/Create_Repo_Git_GUI/02_Initialize_Repo_in_Browser.mp4

Push the Repository to GitHub

Tutorial_Videos/Create_Repo_Git_GUI/03_Push_Repo_to_Github.mp4

Troubleshooting

- **Remote repository not found:** Make sure you created the repository on GitHub and that the URL is correct. Git GUI does **not** include functionality to create a new repository on GitHub.

  It can only:

  - Create a local Git repository on your computer
  - Connect to an existing remote repository (one that already exists on GitHub)
  - Push local changes to that existing remote

- **Authentication failed:** Use a personal access token (PAT) if prompted for your GitHub credentials.
- **Push fails due to README conflict:** Ensure your GitHub repo was created without a README, license, or `.gitignore` to avoid merge conflicts.
- **Changes not appearing:** Click "**Rescan**" to refresh the list of changed files in Git GUI.

Additional Resources

- GitHub Docs: Creating a new repository
- GitHub Docs: Adding a remote
- Git for Windows (includes Git GUI)
- Happy Git with R

8 Creating a New Repository with VSCode

Learning Objectives

- Understand what it means to create a new repository on GitHub
- Successfully create a local Git repository in VSCode and push it to GitHub
- Identify common errors and learn how to fix them

Step-by-Step Instructions

Part 1: Create a Local Repository in VSCode

1. Open Visual Studio Code (VSCode).
2. Create a new folder on your computer and open it in VSCode.
3. Inside VSCode, go to **View > Source Control** or click the Source Control icon on the left sidebar.
4. Click **Initialize Repository** to turn the folder into a local Git repository.
5. Create or add your project files (e.g., `README.md`, `.py`, `.R`, or `.ipynb` files).

Part 2: Create a Repository on GitHub

1. In your browser, go to GitHub.com and sign in.
2. Click the + icon in the upper-right and select **New Repository**.
3. Name your repository (e.g., `my-new-repo`).
4. **Important**: *Do not* initialize the repo with a README, `.gitignore`, or license (this prevents conflicts).
5. Click **Create Repository**.

Part 3: Link and Push to GitHub

1. In VSCode, open the **Terminal** (View > Terminal).
2. In the **Terminal** type:
   - git remote add origin https://github.com/vmahan1998/GitHub_Basic_Skills_Create_Repo_Test_VSCode.git
   - git branch -M main
   - git push -u origin main

Video Tutorial

Create the Repository

Tutorial_Videos/Create_Repo_VSCode/01_Create_Repo_VSCode.mp4

Push the Repository to GitHub

Tutorial_Videos/Create_Repo_VSCode/02_Connect_Repo_Push_Changes_VSCode.mp4

Troubleshooting

- **Remote repository not found:**
  Make sure you created the repository on GitHub and copied the correct URL. RStudio does not create remote repositories.
- **RStudio can:**
  - Create a local Git project on your computer
  - Connect to an existing remote repository (one that already exists on GitHub)
  - Push local changes to that remote
- **Authentication failed:**
  GitHub no longer accepts passwords for HTTPS. If prompted, use a personal access token (PAT) in place of your password.
- **Push fails due to README or file conflicts:**
  Make sure your GitHub repository was created **without** initializing it with a README, .gitignore, or license. These files can create merge conflicts when pushing from RStudio.
- **Missing Git in RStudio:**
  If Git options don't appear when creating a new project, go to `Tools > Global Options > Git/SVN` and verify that RStudio detects Git. You may need to install Git from git-scm.com and restart RStudio.
- **Pushed changes not showing on GitHub:**
  Make sure you pushed to the correct remote and branch. Use `Git > Push` again in RStudio and check your GitHub repository online to confirm.

Additonal Resources

- GitHub Docs: Cloning a repository
- Happy Git with R
- Using Git in VSCode
- Git for Windows

9 Creating a New Repository with Rstudio

Learning Objectives

- Understand what it means to create a new repository on GitHub
- Successfully create a local Git repository in RStudio and push it to GitHub
- Identify common errors and learn how to fix them

Step-by-Step Instructions

1. Open RStudio.
2. Go to **File > New Project > New Directory > Empty Project**.
3. Choose a folder name and directory location for your new project.
4. Check the box for **"Create a git repository"** (this initializes Git locally).
5. Click **Create Project**. Your new project will open with Git enabled.
6. Create or add your project files (e.g., `README.md`, `.R` scripts, data).
7. In your browser, create a new repository on GitHub.com.
    - Do **not** initialize it with a README, license, or `.gitignore`.
8. Back in RStudio, go to the **Terminal** pane and run:
    - git remote add origin https://github.com/yourusername/your-repo-name.git
    - git branch -M main
    - git push -u origin main

Video Tutorial

Create the Repository

Tutorial_Videos/Create_Repo_RStudio/01_Create_Repo_RStudio.mp4

Push the Repository to GitHub

Tutorial_Videos/Create_Repo_RStudio/02_Create_Branch_Push_Repo.mp4

Troubleshooting

- **Remote repository not found:**
  Make sure you created the repository on GitHub and copied the correct URL. RStudio does not create remote repositories.

  RStudio can:

  - Create a local Git project on your computer
  - Connect to an existing remote repository (one that already exists on GitHub)
  - Push local changes to that remote

  Authentication failed:
  GitHub no longer accepts passwords for HTTPS. If prompted, use a personal access token (PAT) in place of your password.

  Push fails due to README or file conflicts:
  Make sure your GitHub repository was created **without** initializing it with a README, .gitignore, or license. These files can create merge conflicts when pushing from RStudio.

  Missing Git in RStudio:
  If Git options don't appear when creating a new project, go to `Tools > Global Options > Git/SVN` and verify that RStudio detects Git. You may need to install Git from git-scm.com and restart RStudio.

  Pushed changes not showing on GitHub:
  Make sure you pushed to the correct remote and branch. Use `Git > Push` again in RStudio and check your GitHub repository online to confirm.

Additional Resources

- GitHub Docs: Creating a repository
- Happy Git with R
- RStudio Git Integration
- Git for Windows

10 Forking a Repository and Making Changes with Git GUI

Learning Objectives

- Understand what it means to fork a repository on GitHub
- Successfully fork a repository on GitHub and make changes locally using Git GUI
- Identify common errors and learn how to fix them

Step-by-Step Instructions

Part 1: Fork the Repository on GitHub

1. Navigate to the public repository you want to contribute to. (e.g., https://github.com/vmahan1998/GitHu
2. Click the "**Fork** button in the top-right corner of the page.
3. Choose your GitHub account as the destination for the fork.
4. GitHub will create a copy of the repository under your account.

Part 2: Clone the Forked Repository Locally

1. Open Git GUI.
2. Click "**Clone Existing Repository**".
3. Copy the URL of your forked repository from GitHub (e.g., `https://github.com/yourusername/repo-na`
4. Paste the URL into the **Source Location** field.
5. Choose a **Target Directory** and name the folder (e.g., `repo-name`).
6. Click "**Clone**".

Part 3: Create New Branch

1. In Git GUI, go to **Branch > Create**.
2. Name your new branch (e.g., fix-typo, feature-new-header, or update-readme).
3. Click "**Create**" to switch to the new branch.

Part 4: Make and Commit Changes Locally

1. Open the cloned folder in File Explorer and make changes to the project files (e.g., `README.md` or code).
2. In Git GUI, click "**Rescan** to detect changes.
3. Stage your changes by clicking "**Stage Changed**".
4. Write a commit message describing your changes and click "**Commit**".

Part 5: Push Changes to Your Fork on GitHub

1. Go to **Remote** > **Push**.
2. Make sure you are pushing to your fork (`new-message`) and to the appropriate branch (`main` or `master`).
3. Click "**Push**" to upload your changes.

Part 6: Create a Pull Request

1. Visit your fork on GitHub.
2. Click "**Compare & pull request**".
3. Add a title and description of your changes.
4. Click "**Create pull request**" to submit your contribution to the original repository.

Part 7: Merge Pull Request (for repository maintainers)

1. In the original repository (or in your fork if you're an owner), open the pull request.
2. Click "**Merge pull request**".
3. Confirm by clicking "**Confirm**".
4. After merging, you can safely delete the branch you created.

Video Tutorial

Fork the Repository

Tutorial_Videos/Fork_Repo_Git_GUI/01_Fork_GitHub_Repo.mp4

Clone the Repository

Tutorial_Videos/Fork_Repo_Git_GUI/02_Clone_GitHub_Repo.mp4

Create a New Branch and Edit Files

Tutorial_Videos/Fork_Repo_Git_GUI/03_Create_Branch_Push_Changes.mp4

Create a Pull Request

Tutorial_Videos/Fork_Repo_Git_GUI/04_Create_Pull_Request.mp4

Merge a Pull Request

Tutorial_Videos/Fork_Repo_Git_GUI/05_Merge_Pull_Request.mp4

Troubleshooting

- **Authentication failed:** Use a personal access token (PAT) if prompted for your GitHub credentials.
- **Remote repository not found:** Double-check the URL you copied from your fork.
- **Unable to push changes:** Ensure you are pushing to your fork, not the original repository (you won't have write access to the original).
- **Changes not showing in Git GUI:** Click **Rescan** to detect uncommitted changes.
- **Wrong branch:** Make sure you're working on the correct branch, especially if the original project uses `main` instead of `master`.

11 Additional Resources

- GitHub Docs: Forking a repository
- GitHub Docs: About pull requests
- Git for Windows (includes Git GUI)
- Happy Git with R

12 Fork a New Repository with VSCode

Learning Objectives

- Understand what it means to fork a repository on GitHub
- Successfully fork a repository on GitHub and make changes locally using VSCode
- Identify common errors and learn how to fix them

Step-by-Step Instructions

Part 1: Fork the Repository on GitHub

1. Navigate to the public repository you want to contribute to. (e.g., https://github.com/vmahan1998/GitHu
2. Click the "**Fork** button in the top-right corner of the page.
3. Choose your GitHub account as the destination for the fork.
4. GitHub will create a copy of the repository under your account.

Part 2: Clone the Forked Repository Locally

1. Open Visual Studio Code (VSCode).
2. On the **Welcome page**, click **Clone Git Repository**.
    - If you don't see this option, install the **GitHub Extension** from the VSCode marketplace.
3. Paste the repository URL.
4. Choose a local folder where the repository should be saved.
5. Name the directory (typically the same as the repo name "**GitHub_Basic_Skills**").
6. When prompted, click "**Open"** to begin working inside the cloned repository.

Part 3: Create New Branch

1. In VSCode, open the **Source Control** sidebar (click the branch icon or press `Ctrl+Shift+G`).
2. Click the branch name (e.g., `main`) in the lower-left corner or at the top of the Source Control pane.
3. Select "**Create new branch"** from the dropdown.
4. Enter a name for your branch (e.g., `new-message`) and press "**Enter"**.
5. VSCode will automatically switch you to the new branch.

Part 4: Make and Commit Changes Locally

- Edit the files in VSCode as needed (e.g., modify `EDIT_ME.md` or source code).
- Save your changes (`Ctrl+S` or File > Save).
- Open the **Source Control** pane.
- You should see your changed files listed.
- Hover over the file and click the + icon to stage it, or click the + icon next to **Changes** to stage all files.
- In the message box at the top, write a clear and descriptive commit message.
- Click the " **" Commit** button (or use the check mark icon).
- Click the " **" Push** icon in the bottom bar or from the Source Control menu to push your branch to GitHub.

Part 6: Create a Pull Request

1. Go to your forked repository on GitHub in a web browser.
2. GitHub will prompt you to compare & open a pull request.
3. Click "**Compare & pull request"**.
4. Add a descriptive title and explanation of your changes.
5. Click "**Create pull request"** to propose your contribution to the original repository.

Part 7: Merge Pull Request (for repository maintainers)

1. If you have permission, go to the pull request on the original repository.
2. Review the changes and click "**Merge pull request"**.
3. Confirm the merge and delete the feature branch if no longer needed.

Video Tutorial

Fork the Repository

Tutorial_Videos/Fork_Repo_VSCode/01_Create_Branch_Push_Changes_VSCode.mp4

Create a Pull Request

Tutorial_Videos/Fork_Repo_VSCode/02_Create_Pull_Request_VSCode.mp4

Merge a Pull Request

Tutorial_Videos/Fork_Repo_VSCode/03_Merge_Pull_Request_VSCode.mp4

Troubleshooting

- **Remote repository not found**
  Make sure you created the repository on GitHub and copied the correct HTTPS or SSH URL. VSCode does not create remote repositories — it only connects to them.
- **VSCode can:**
  - Clone a remote repository from GitHub
  - Connect your local changes to a remote repo via Git
  - Push changes to an existing remote repository
- **Authentication failed**
  GitHub no longer supports passwords for HTTPS. If prompted, use a personal access token (PAT) instead. You can generate one at https://github.com/settings/tokens.
- **Push fails due to README or file conflicts**
  If the GitHub repo was initialized with a README, license, or `.gitignore`, and your local repo wasn't, this can cause merge conflicts. Create your GitHub repo without those options when pushing an existing local project.
- **Git not detected**
  If Git features don't appear in VSCode, install Git from https://git-scm.com and restart VSCode. Go to **View > Command Palette > Git: Show Git Output** to check if it's recognized.
- **Pushed changes not showing on GitHub**
  Make sure you pushed to the correct remote (usually `origin`) and correct branch (`main` or `master`). You can check the branch in the bottom-left corner of VSCode.

Additonal Resources

- GitHub Docs: Cloning a repository
- Using Git in VSCode
- GitHub Docs: Creating a personal access token

13 Fork a New Repository with RStudio

Learning Objectives

- Understand what it means to fork a repository on GitHub
- Successfully fork a repository on GitHub and make changes locally using RStudio
- Identify common errors and learn how to fix them

Step-by-Step Instructions

Part 1: Fork the Repository on GitHub

1. Navigate to the public repository you want to contribute to. (e.g., https://github.com/vmahan1998/GitHu
2. Click the "**Fork** button in the top-right corner of the page.
3. Choose your GitHub account as the destination for the fork.
4. GitHub will create a copy of the repository under your account.

Part 2: Clone the Forked Repository Locally

1. Open RStudio.
2. Go to **File > New Project > Version Control > Git**.
3. Paste the URL for the repository you want to clone:
   Example: "https://github.com/yourusername/your-repo-name.git"
4. Choose a directory where you want to store the project.

Part 3: Create New Branch

1. In the **Terminal** in RStudio type:

```
-   git checkout -b new-message
```

Note: "**new-message**" is the name of the new branch.

Part 4: Make and Commit Changes Locally

1. Edit files in the RStudio editor as needed (e.g., modify `EDIT_ME.md` or code scripts).
2. In the **Git** tab (upper-right pane), click "**Refresh"** or"**Commit"**.
3. Stage the files you changed by checking the boxes next to them.
4. Add a descriptive commit message in the message box.
5. Click "**Commit"**.
6. Click the **green arrow** pointed north to push changes to GitHub Repo.

Part 6: Create a Pull Request

1. Go to your forked repository on GitHub in a web browser.
2. GitHub will prompt you to compare & open a pull request.
3. Click "**Compare & pull request"**.
4. Add a descriptive title and explanation of your changes.
5. Click "**Create pull request"** to propose your contribution to the original repository.

Part 7: Merge Pull Request (for repository maintainers)

1. If you have permission, go to the pull request on the original repository.
2. Review the changes and click "**Merge pull request"**.
3. Confirm the merge and delete the feature branch if no longer needed.

Video Tutorial

Fork the Repository

Tutorial_Videos/Fork_Repo_RStudio/01_Fork_Repo_RStudio.mp4

Clone the Repository, Create a New Branch, and Edit Files

Tutorial_Videos/Fork_Repo_RStudio/02_Clone_Repo_Create_Branch_RStudio.mp4

Create a Pull Request

Tutorial_Videos/Fork_Repo_RStudio/03_Create_Pull_Request_RStudio.mp4

Merge a Pull Request

Tutorial_Videos/Fork_Repo_RStudio/04_Merge_Pull_Request_RStudio.mp4

Troubleshooting

- **Remote repository not found:**
  Make sure you created the repository on GitHub and copied the correct URL. RStudio does not create remote repositories.

  RStudio can:

  - Create a local Git project on your computer
  - Connect to an existing remote repository (one that already exists on GitHub)
  - Push local changes to that remote

  Authentication failed:
  GitHub no longer accepts passwords for HTTPS. If prompted, use a personal access token (PAT) in place of your password.

  Push fails due to README or file conflicts:
  Make sure your GitHub repository was created **without** initializing it with a README, .gitignore, or license. These files can create merge conflicts when pushing from RStudio.

  Missing Git in RStudio:
  If Git options don't appear when creating a new project, go to `Tools > Global Options > Git/SVN` and verify that RStudio detects Git. You may need to install Git from git-scm.com and restart RStudio.

  Pushed changes not showing on GitHub:
  Make sure you pushed to the correct remote and branch. Use `Git > Push` again in RStudio and check your GitHub repository online to confirm.

Additional Resources

- GitHub Docs: Creating a repository
- Happy Git with R
- RStudio Git Integration
- Git for Windows

14 Addressing Version Conflicts

Learning Objectives

- Understand what causes version conflicts in Git
- Learn how to identify and resolve conflicts in your code editor or Git interface
- Gain confidence in committing resolved changes and continuing collaboration workflows

What Are Version Conflicts?

A version conflict happens when two people (or two branches) change the same line of a file differently, and Git cannot decide which version to keep. This typically occurs during merging or pulling changes.

When Do Conflicts Occur?

- Pulling changes from the remote repository that modify the same file you changed locally
- Merging branches that have diverged with overlapping changes
- Rebasing a branch with changes that conflict with the main branch

Step-by-Step Instructions to Resolve Conflicts

Step 1: Identify the Conflict

Git will tell you which files are in conflict. Open your Git interface or terminal and look for a message like:

```
CONFLICT (content): Merge conflict in filename.ext
```

The file will now contain conflict markers like:

```
<<<<<<< HEAD Your local changes ======= Incoming changes from the other branch >>>>>>> branch
```

Step 2: Edit the File

- Open the file in your editor (RStudio, VSCode, or any text editor).
- Decide what the correct version should be. You can:
  - Keep your changes
  - Keep the incoming changes
  - Combine both
- Delete the conflict markers (`<<<<<<<`, `=======`, `>>>>>>>`) after resolving.
- Stage file with conflicts resolved.

14.1 Step 4: Complete the Merge or Pull

If you were merging:

```
git commit -m "Resolve merge conflict in filename.ext"
```

If you were pulling:

```
git pull --continue
```

Tool-Specific Tips

RStudio

- Conflicted files appear in the Git tab
- Edit the file, then click the checkbox next to it
- Click Commit to finalize

VSCode

- Conflict sections are highlighted with options to accept current, incoming, both, or compare
- Click the appropriate resolution button for each conflict block
- Save the file and commit

Git GUI

- After a merge conflict, click “Rescan”
- Stage files after resolving
- Add a commit message and click Commit

Troubleshooting

- **Still see conflict markers?** Ensure all markers were removed from the file.
- **Commit rejected after resolving?** You may have staged but not committed. Try again with `git commit`.
- **Can't push?** Make sure your branch is ahead of the remote and conflict-free.

Additional Resources

- GitHub Docs: Resolving a merge conflict
- VSCode Docs: Resolve merge conflicts

15 Recovering Stashed Changes

Learning Objectives

- Understand what it means to stash changes in Git
- Learn how to view, apply, or drop stashed changes
- Restore work safely after interruptions or switching branches

What Is Git Stash?

Stashing temporarily saves your uncommitted changes so you can work on something else without losing progress. Git stores these changes in a stack, which you can apply later to continue where you left off.

When to Use Stash

You are in the middle of editing files but need to:

- Switch branches
- Pull new changes from the remote
- Run or test something cleanly without committing unfinished work

Step-by-Step Instructions

Step 1: Stash Your Changes

In your terminal or Git interface, run:

```
git stash
```

This command stashes all tracked changes. If you also want to stash untracked files:

```
git stash -u
```

You will see a confirmation message like:

```
Saved working directory and index state WIP on main: abc1234 Update index file
```

Step 2: List All Stashes

You can view all stored stashes with:

```
git stash list
```

Output will look like:

```
stash@{0}: WIP on main: abc1234 Update index file stash@{1}: WIP on feature-branch: def5678 A
```

Step 3: Apply a Stash

To apply the most recent stash:

```
git stash apply
```

To apply a specific stash from the list:

```
git stash apply stash@{1}
```

To both apply and remove the stash:

```
git stash pop
```

Step 4: Continue Your Work

After applying the stash:

- Open your files
- Make any additional edits
- Stage and commit your changes as usual

Tool-Specific Tips

RStudio

- Use the Terminal tab to run stash commands
- There is no Git stash functionality in the Git tab

VSCode

- Use the built-in Terminal to stash or apply
- Extensions like GitLens allow visual stash management

Git GUI

- Stashing is not supported in the interface
- Use Git Bash or another terminal to stash or apply changes

Troubleshooting

- **Stash doesn't apply**: Check for merge conflicts or try a specific stash with `git stash apply stash@{0}`
- **Changes disappear after popping**: Use `git stash list` to confirm the stash was removed
- **Forgot what you stashed**: Use `git stash show -p` to review details of a stash

Additional Resources

- GitHub Docs: git stash
- Happy Git with R: Stashing
- Pro Git Book: Git Stashing

16 Troubleshooting Common Problems

Learning Objectives

- Recognize several common problems when using GitHub
- Learn how to troubleshoot common Git issues with step-by-step solutions
- Build confidence navigating authentication, syncing, and collaboration challenges

What Are Common GitHub Problems?

Even experienced users encounter problems when working with Git and GitHub. These issues usually occur during setup, collaboration, or when switching between tools like RStudio, VSCode, and Git GUI.

When Do These Problems Occur?

- Pushing changes without proper authentication
- Pulling updates when your local files have diverged
- Conflicting file versions due to multiple contributors
- Forgetting to stage, commit, or sync your changes
- Not configuring Git properly the first time

Step-by-Step Guide to Fix Common Issues

Problem 1: Authentication Failed

What it looks like: You're prompted for a password, but login fails.
Why it happens: GitHub no longer supports password authentication for HTTPS.

Solution:

1. Generate a Personal Access Token (PAT).
2. Use the token in place of your GitHub password when prompted.
3. Consider using an SSH key for future sessions (especially in VSCode).

Problem 2: Repo Exists on GitHub, But Push Fails

What it looks like: You get an error saying remote repository not found or push is rejected.
Why it happens: The GitHub repo is initialized with a README or other files that conflict with your local setup.

Solution:

1. When creating the repo on GitHub, do not initialize with README, license, or .gitignore.
2. If conflicts exist, pull first:

   ```
   git pull origin main --allow-unrelated-histories
   ```

3. Resolve any conflicts, then push.

Problem 3: Git Not Detected

What it looks like: Git features are missing in RStudio or VSCode.
Why it happens: Git is not installed or not properly linked to your IDE.

Solution:

1. Install Git from git-scm.com.
2. Restart your IDE.
3. In RStudio: Go to **Tools > Global Options > Git/SVN** and verify the Git executable path.
4. In VSCode: Git should be detected automatically. If not, check your system path.

Problem 4: Changes Aren't Showing on GitHub

What it looks like: You committed locally, but nothing appears in your GitHub repo.
Why it happens: You forgot to push your changes.

Solution:

1. After committing, push using:

   ```
   git push origin main
   ```

2. Refresh the GitHub page to confirm.

Problem 5: You Cloned the Wrong Repo or Fork

What it looks like: You're editing a repo, but you can't push changes.
Why it happens: You cloned the original repo instead of your fork.

Solution:

1. Fork the original repository on GitHub.
2. Clone your fork, not the original project.
3. Confirm your remote URL with:

   ```
   git remote -v
   ```

Tool-Specific Tips

RStudio

- Use the Git tab to commit and push
- If Git tab is missing, configure Git in **Tools > Global Options > Git/SVN**

VSCode

- Use the Source Control tab
- Look for inline errors or tooltips when actions fail

Git GUI

- Use “Rescan” and “Push” manually
- Add remote via **Remote > Add**

Summary

Problem	Fix
Authentication fails	Use PAT or set up SSH
Push fails due to README conflict	Don't initialize GitHub repo with README
Git not recognized	Install Git and restart IDE
Can't push changes	Make sure you're on the correct branch and have committed
Wrong repo cloned	Confirm remote URL matches your GitHub fork

Additional Resources

- GitHub Docs: Troubleshooting Git
- Happy Git with R
- Pro Git Book
- VSCode Docs: Git Support