Have fun with Git without going mental!

Git Fun(damentals)

Having fun with Git without going mental!

Online version of the presentation:

- Git Fun(damentals) - Sean on IT
- Taming Git and GitHub with PowerShell - Sean on IT

What is Git?

In 2005, Linus Torvalds, the creator of Linux, needed to replace the version control system used to manage the source code of the Linux kernel. He, and the Linux community, created Git. Some of the goals of the new system included:

- Speed
- Simple design
- Strong support for non-linear development (thousands of parallel branches)
- Fully distributed Able to handle large projects like the Linux kernel efficiently (speed and data size)

Git is distributed

- Nearly every operation is local
- In centralized version control system (CVCS), you must connect to the central store to access and update the history. With Git you have the entire history of the project right there on your local disk, most operations seem almost instantaneous.
- This also means that there is very little you can't do if you're offline.

Git is safe

Integrity

- Git uses SHA-1 hashes of data to verify integrity

- The SHA-1 hash is used to identify every object in the database

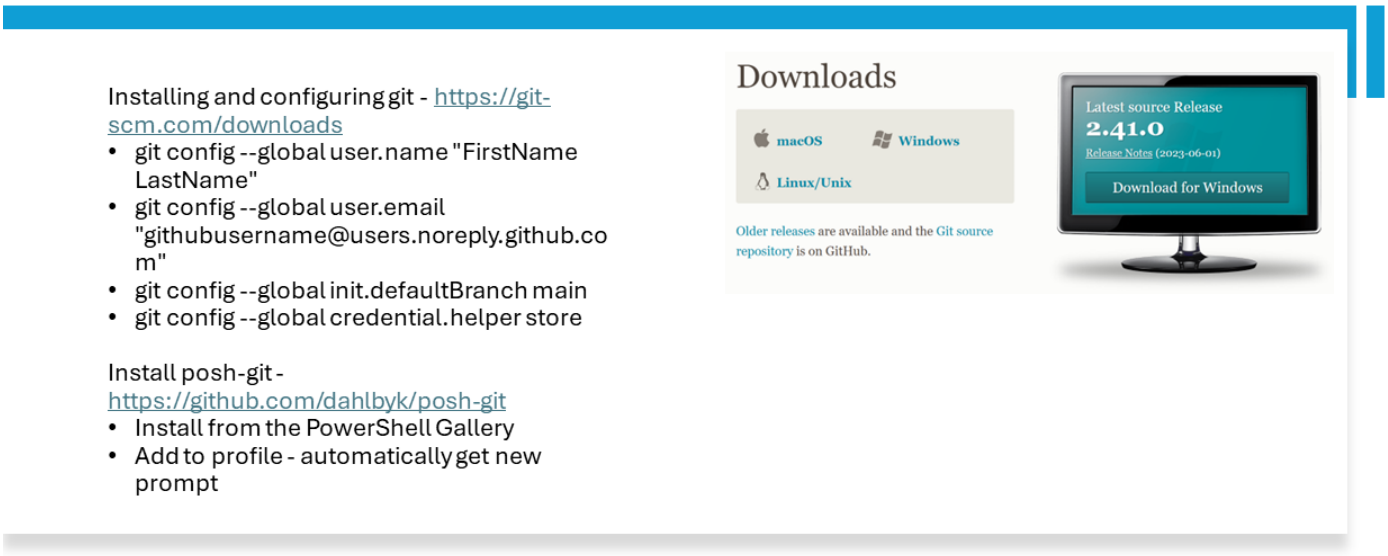
Git only adds data

- Nearly every operation adds information to the database
- It's nearly impossible to delete anything
- This allows you to easily recover a previous state

What is GitHub?

GitHub hosts Git repositories and provides developers with tools to ship better code through command line features, issues (threaded discussions), pull requests, and code review. GitHub builds collaboration directly into the development process. Work is organized into repositories where developers can outline requirements or direction and set expectations for team members. Then, using the GitHub flow, developers simply create a branch to work on updates, commit changes to save them, open a pull request to propose and discuss changes, and merge pull requests once everyone is on the same page.

Setting up Git for PowerShell



Installing Git for Windows

Download the latest version of Git for the platform you are using. There are versions for Windows, macOS, and Linux. https://git-scm.com/downloads

Installing posh-git

posh-git is a PowerShell module that integrates Git with PowerShell. It provides Git status summary information that's displayed in a customized PowerShell prompt. **posh-git** also provides tab completion support for git commands, branch names, paths, and more.

For more information, see:

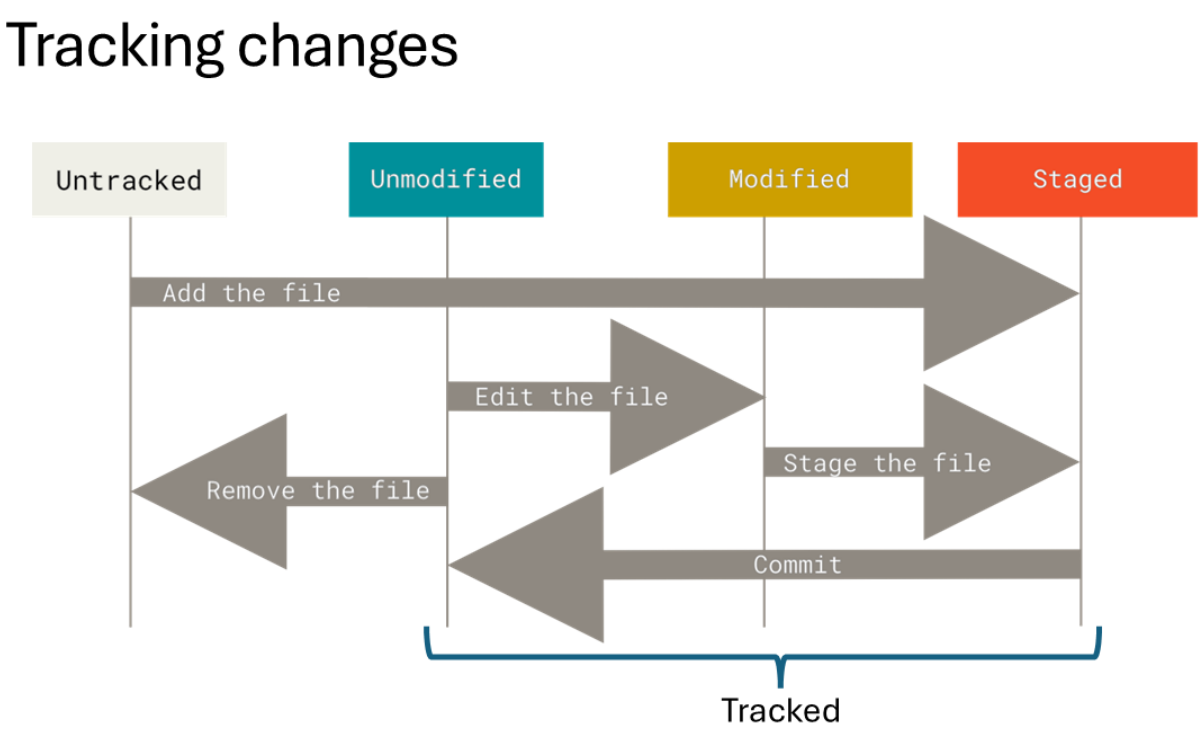
- posh-git on the PowerShell Gallery
- posh-git on GitHub

Install posh-git using the following command:

```
Install-Module posh-git
```

Add the following command to your profile script.

```
Import-Module posh-git
```



Tracking changes

The files in your working directory can be in one of two states: tracked or untracked.

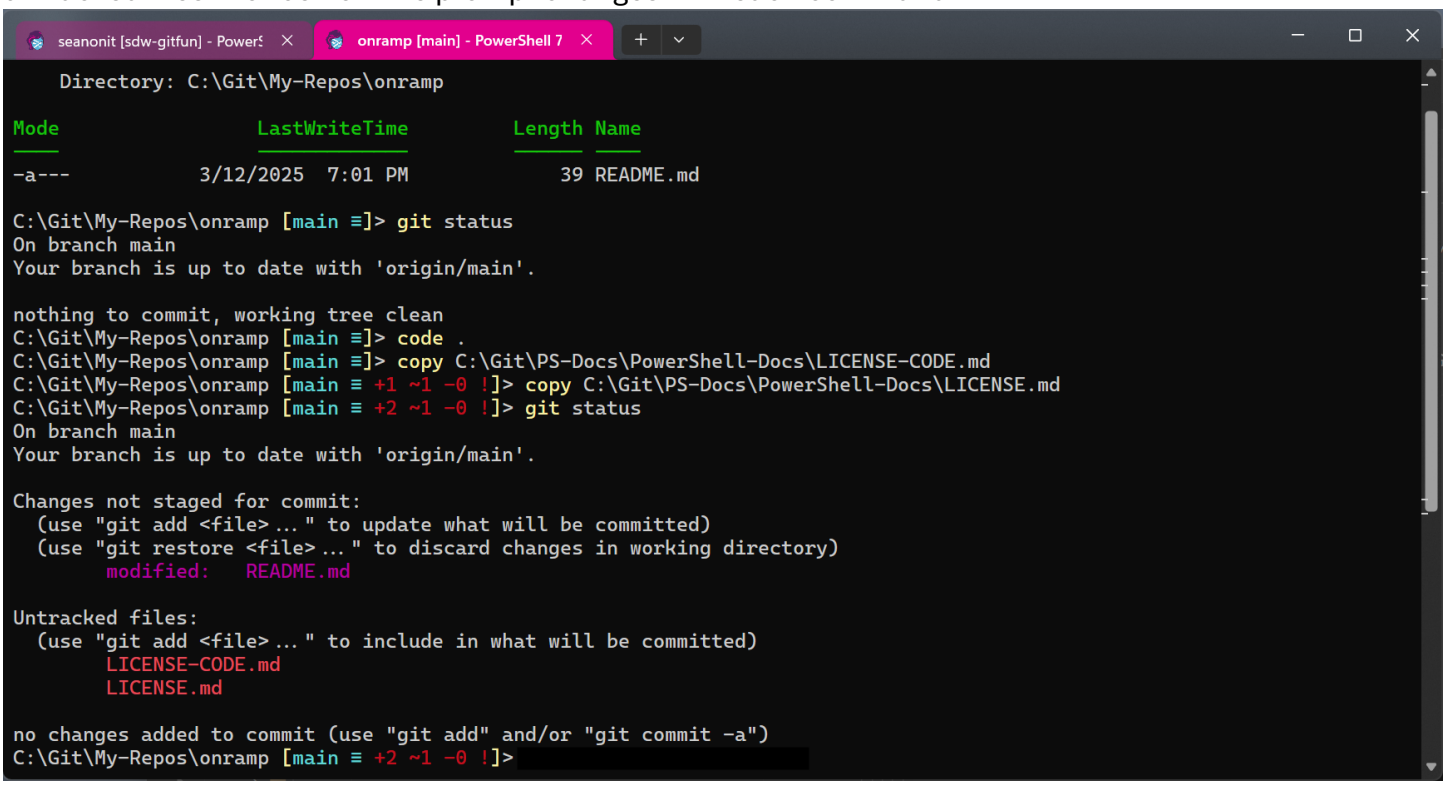
- Tracked files are files that were included in the last snapshot. They can be unmodified, modified, or staged. These files are tracked by the Git index (staging area).
- Untracked files are any files in your working directory that were not in your last snapshot and are not in your staging area. When you first clone a repository, all of your files will be tracked and unmodified because they were in the last snapshot. As you modify files, Git sees them as modified because you have changed them since your last commit. You stage these modified files and then commit them to save the snapshot permanently in your Git directory.

Git commands to track changes

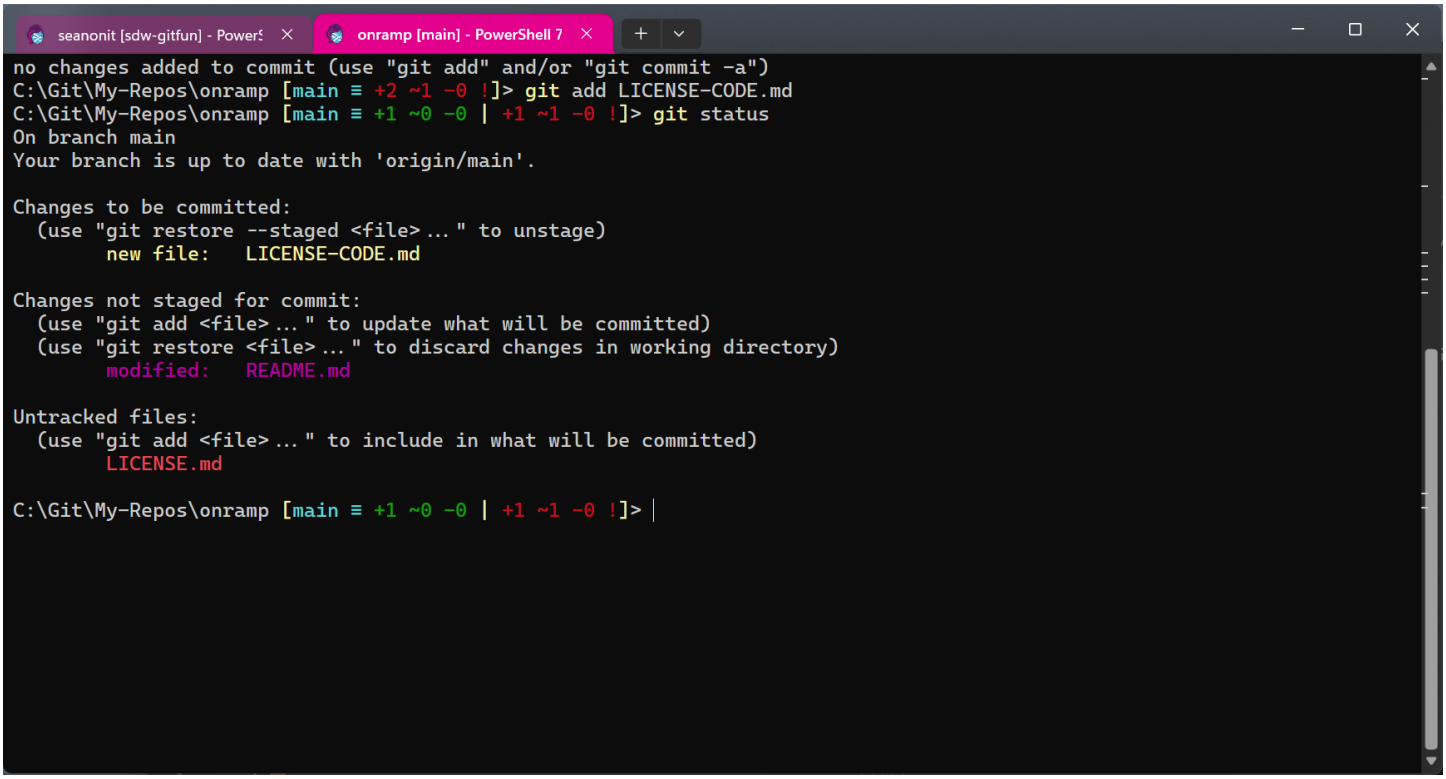
- `git status` - Shows the status of changes as untracked, modified, or staged
- `git add` - Adds changes to the staging area
- `git commit` - Saves the snapshot permanently in your Git directory

The following screenshots show the progression of changes to the working directory and how they are tracked by Git.

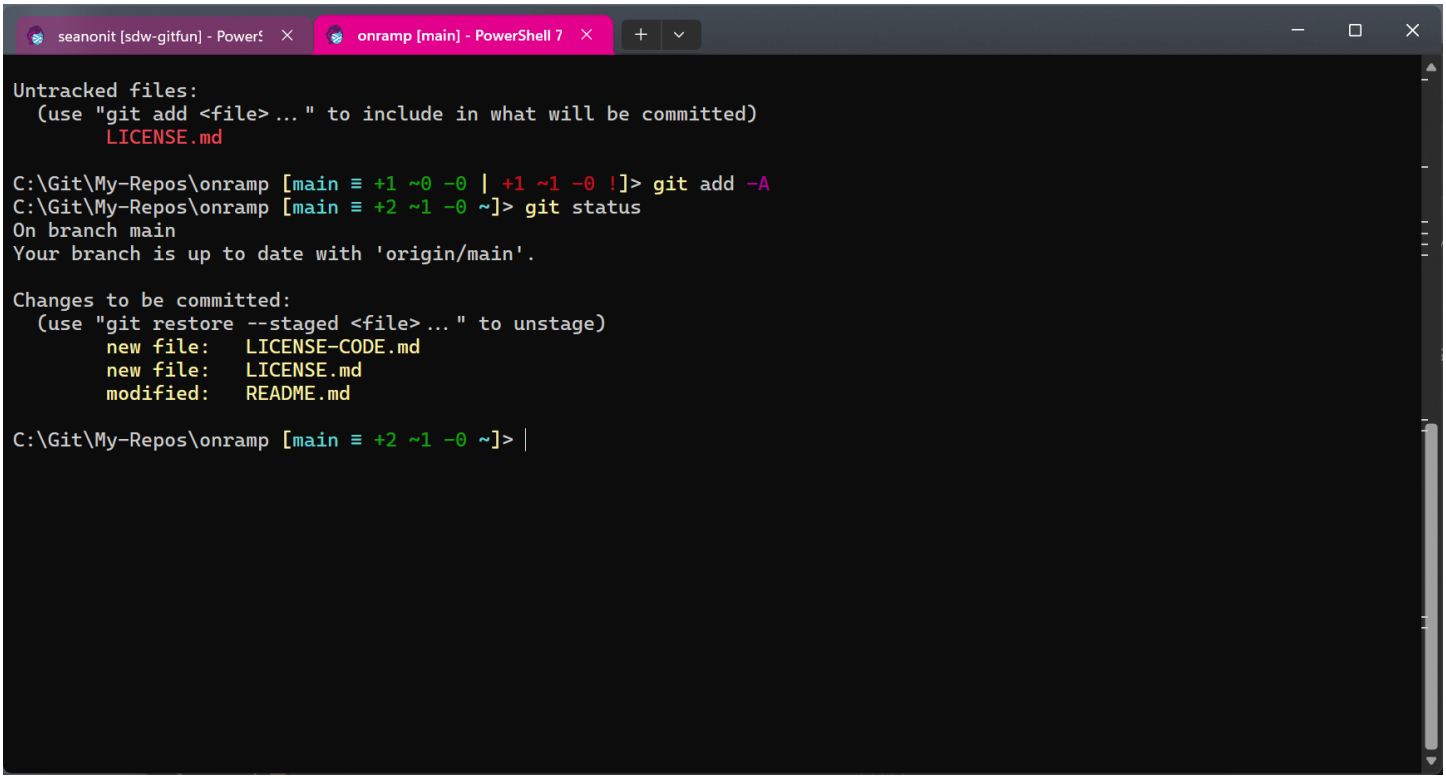
In the first screenshot, you can see that the local working directory contains one file, README.md. The git status command shows the working tree is clean. Next, the README.md files is edited and two license files are copied from another location. The git status command shows one modified file and two untracked files. Notice how the prompt changes with each command.



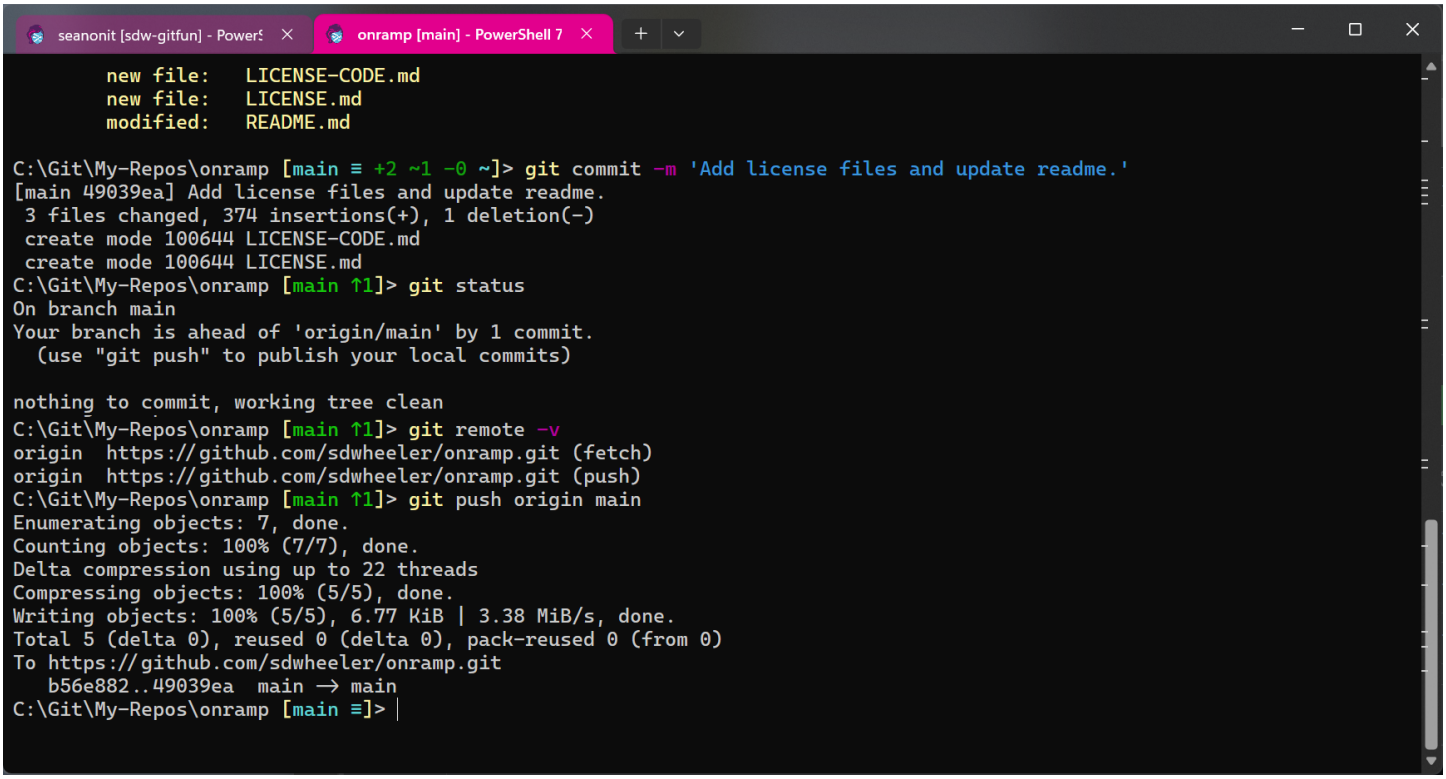
The git add command is used to add a license file to the staging area. The git status command shows that `LICENSE-CODE.md` is now staged.



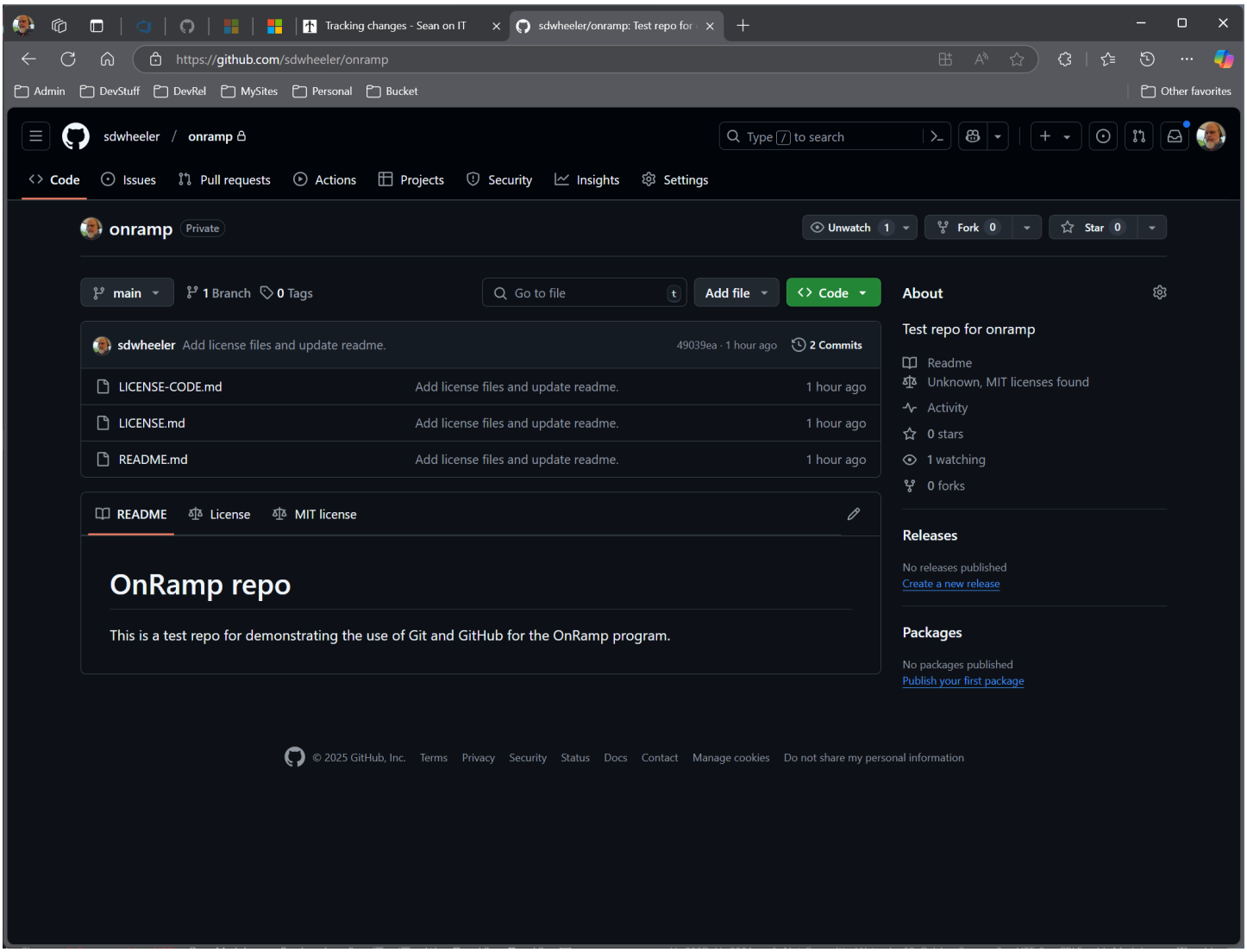
The git add -A command adds all files to the staging area. The git status command shows that all three files are now staged.



The git commit command is used to save the changes to the local repository. Finally, the git push command is used to push the changes to GitHub.



Now we can see the changes in GitHub. The README.md file is shown with the changes and the two license files are shown as new files.



What is a branch?

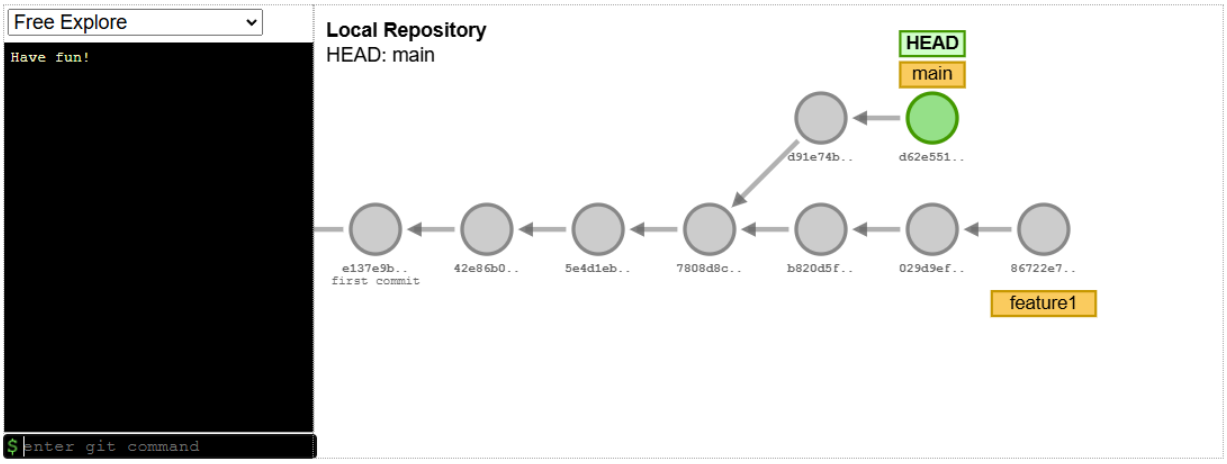


Image from https://git-school.github.io/visualizing-git/

What is a branch?

Branching means you diverge from the main line of development and continue to do work without messing with that main line. In many VCS tools, this is a somewhat expensive process, often requiring you to create a new copy of your source code directory, which can take a long time for large projects. When you make a commit, Git stores a commit object that contains a pointer to the snapshot of the content you staged. This object also contains the author's name and email address, the message that you typed, and pointers to the commit or commits that directly came before this commit (its parent or parents): zero parents for the initial commit, one parent for a normal commit, and multiple parents for a commit that results from a merge of two or more branches.

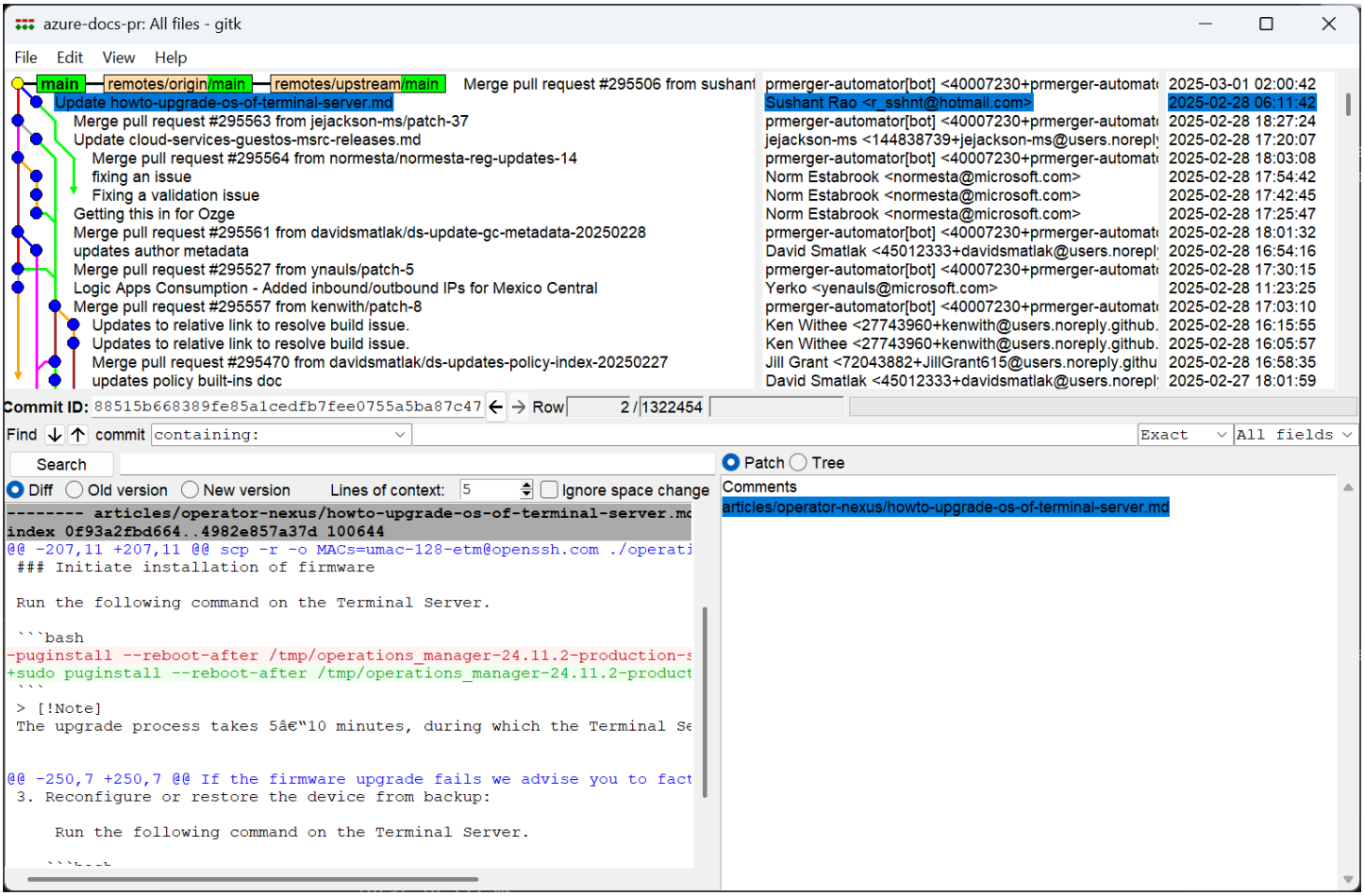
The website Visualizing Git provides a great simulation of how branching works in Git. You can see how the commit history is represented as a graph and how branches are created and merged.

Branch commands

- `git branch <branch-name>` - Create a new branch
- `git checkout -b <branch-name>` - Create a new branch and switch to it
- `git checkout <branch-name>` - Switch to a branch
- `git merge-base main working-branch` - Find the common ancestor of two branches
- `git merge <branch-name>` - Merge a branch into the current branch
- `git rebase <branch-name>` - Reapply commits on top of another base tip
- `git branch -d <branch-name>` - Delete a branch
- `git branch -v` - List local branches in your repository
- `git remote -vr` - List remote branches in your repository
- `git branch -all -v` - List local and remote branches in your repository

Tools for visualizing commits

There are several tools available to visualize commits and branches in Git. The following screenshot shows the `gitk` tool included with Git. It displays the commit history as a graph and includes the commit message, author, date, and diffs. You can also see the branches and tags in the repository.



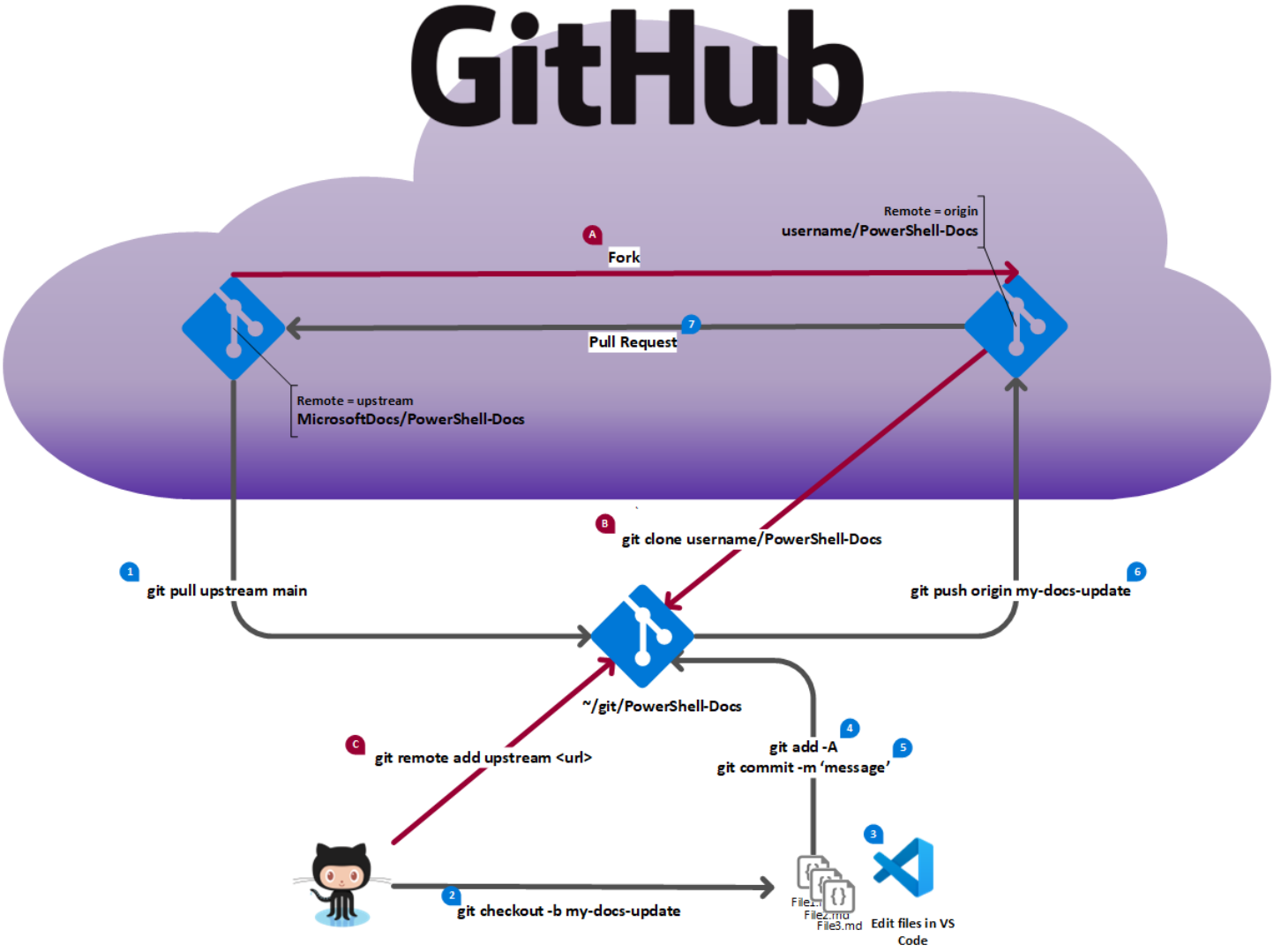
- VS Code - Use the built-in Git extension to visualize commits and branches
- `git log --oneline --graph` - Show a one-line summary of each commit in a graph format
- `gitk` - Show a graphical representation of the commit history
- `tig` - Show a text-based graphical representation of the commit history

GitHub workflow for a forked repository

GitHub flow is a lightweight, branch-based workflow. The GitHub flow is useful for everyone, not just developers.

The GitHub documentation has a good overview of the process and the reasons for using it. See GitHub flow. If you want to contribute to someone else's project but don't have write access to the repository, you can use a "fork and pull request" workflow.

The following image illustrates the workflow for using Git and GitHub to update files in the PowerShell-Docs repository. The steps shown in red are a one-time action for each new repository you fork.



Steps	Description of steps	Git command / GitHub actions
A	Fork the repo	Browse to `https://github.com/MicrosoftDocs/PowerShell-Docs` and select the **Fork** button (top right). Open the **Choose and owner** dropdown and select your personal account. Select the **Create fork** button.
B	Clone the repo (once per machine)	`git clone https://github.com/<your-account>/PowerShell-Docs.git`
C	Add the upstream remote	`git remote add upstream https://github.com/MicrosoftDocs/PowerShell-Docs.git`

For a full discussion of forking, see Contributing to projects in the GitHub documentation.

The numbered steps (in black) are described in the table below.

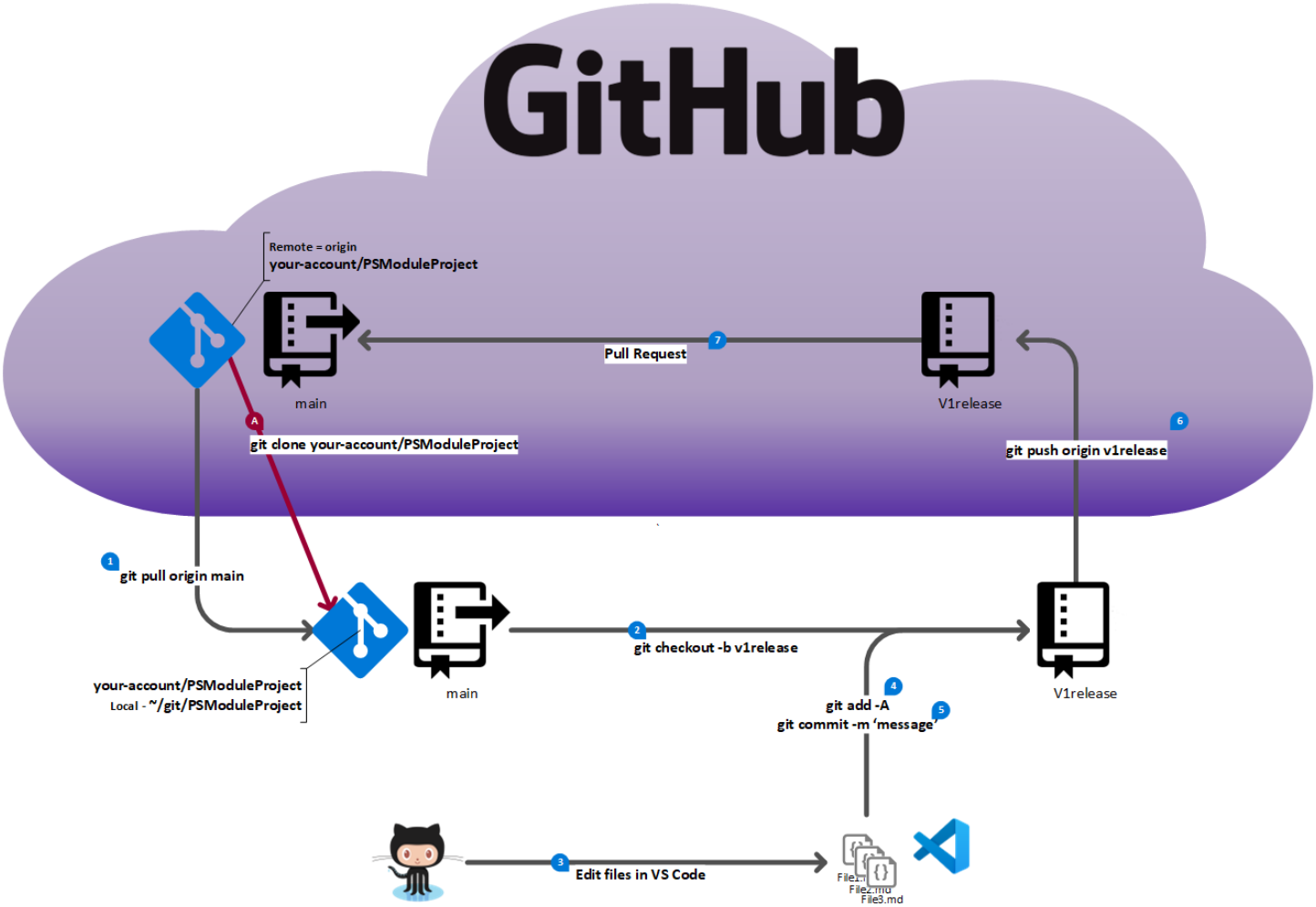
Steps	Description of steps	Git command / GitHub actions
0	Checkout the main branch	`git checkout main`
1	Sync the main branch	`git pull upstream main` `git push origin main`
2	Create a new working branch	`git checkout -b workingbranch`
3	Create new content	Use VS Code to create or edit files
4	Add changes for Git tracking	`git add -A`
5	Commit changes to local repo	`git commit -m 'commit message'`
6	Push working branch to fork	`git push origin workingbranch`
7	Submit pull request	Go to `https://github.com/MicrosoftDocs/PowerShell-Docs/pulls` and click the **New pull request** button. **Base repository**: `MicrosoftDocs/PowerShell-Docs` **base**: `main <- head` **repository**: `username/PowerShell-Docs` **compare**: `workingbranch` Fill out the pull request description and click **Submit**.
8	PR is reviewed	Make the necessary changes based on the review feedback.
9	PR is merged	Go to step 10

10	Cleanup unneeded branch info	`git checkout main`
		`git push origin --delete workingbranch`
		`git branch -D workingbranch`
		
		The git push command deletes the branch in your fork and deletes the tracking branch from your local repo. The git branch command deletes the branch from your local repo.

11	Start new change	Go to step 0

GitHub workflow for single clone

The following image illustrates the workflow for using Git and GitHub to add or change content using a working branch for a single repository. The step shown in red is a one-time action for each machine you work on. The numbered steps (in black) are described in the table below.



Steps	Description of steps	Git command / GitHub actions
A	Clone the repo (once per machine)	`git clone https://github.com/<your-account>/PSModuleProject`
0	Checkout the main branch	`git checkout main`
1	Sync the main branch	`git pull origin main`
2	Create a new working branch	`git checkout -b v1release`
3	Create new content	Use VS Code to create or edit files
4	Add changes for Git tracking	`git add -A`
5	Commit changes to local repo	`git commit -m 'commit message'`
6	Push working branch to origin	`git push origin v1release`
7	Submit pull request	Go to `https://github.com/<your-account>/PSModuleProject/pulls` and click the **New pull request** button. **Base repository**: `your-account/PSModuleProject` **base**: `main <- head` **repository**: `your-account/PSModuleProject` **compare**: `v1release` Fill out the pull request description and click **Submit**.
8	PR is reviewed	Make the necessary changes based on the review feedback.
9	PR is merged	Go to step 10

10	Cleanup unneeded branch info	`git checkout main`
		`git push origin --delete v1release`
		`git branch -D v1release`
		The git push command deletes the branch in your fork and deletes the tracking branch from your local repo. The `git branch` command deletes the branch from your local repo.

11	Start new change	Go to step 0

Appendix - GitHub CLI commands

Many of the tasks you do in GitHub can be done using the GitHub CLI. The following list contains GitHub CLI commands for some of the most common tasks.

Create a new repository

The following command creates a new repository in your GitHub account.

```
PS> gh repo create <repo-name> -d 'Test repo for class' --public --add-readme --
license MIT –clone
```

- The `-d` option allows you to add a description to the repository.
- The `--public` option makes the repository public.
- The `--add-readme` option creates a README file in the new repository.
- The `--license` option adds a license to the repository.
- The `--clone` option clones the new repository to your local machine in the current location in a folder with the same name as the repository.

Create a new GitHub repository from the current local repository

The following command creates a new GitHub repository from the current local repository.

```
PS> gh repo create <repo-name> -d 'Test repo for OnRamp class' --public -s .\ --push
--remote origin
```

- The `-d` option allows you to add a description to the repository.
- The `--public` option makes the repository public.
- The `-s` option specifies the source of the repository. In this case, it is the current directory.
- The `--push` option pushes the current local repository to the new GitHub repository.
- The `--remote` option specifies the name of the remote. In this case, it is origin.

Clone an existing GitHub repository

The following command clones an existing GitHub repository to your local machine.

```
gh repo clone <repo-name>
```

The GH CLI will automatically create a folder with the same name as the repository in the current directory. It also creates a remote called origin that points to the GitHub repository. If the repository you cloned is a fork, it will also create a remote called upstream that points to the original repository.

Checkout a PR submitted by someone else

The following command checks out a pull request submitted by someone else.

```
gh pr checkout <pr-number>
```

This command creates a new branch in your local repository with the same name as the pull request branch and checks it out. The new branch will be based on the branch that the pull request is being merged into. The command also creates a new remote named for the account that submitted the pull request. This remote is used to fetch the pull request branch from the original repository. If you have sufficient permissions, you can also push changes to the pull request branch.

Other available commands

The GitHub CLI has many other commands that you can use to manage your GitHub repositories.

- Manage issues: `gh issue -h`
- Manage labels: `gh label -h`
- Manage pull requests: `gh pr -h`
- Manage releases: `gh release -h`
- Manage repos: `gh repo -h`

For a complete list of commands, see the GitHub CLI documentation.

The GitHub CLI also provides an extension system that allows you to add custom commands to the CLI. The dash extension provides a text-based dashboard for your GitHub repositories. The dashboard allows you to view issues and pull requests and perform basic tasks.