Week 3

Last week, we used JavaScript to make a button display a fun fact. So far, all of our work has lived in one folder on our own computer. This week, we will start saving that work with Git and sharing it through GitHub.

We will use two parts of VS Code. The Source Control view shows our changed files and gives us buttons for common Git actions. The terminal lets us give Git commands directly. Both work with the same folder and the same history.

We will still open the page in a browser to test it. Our code stays in VS Code, and we will publish it through Source Control.

By the end of this hour, your local `main` branch should be published and a second branch should contain one tested, pushed change. We will also walk through a pull request and partner review. If your pair needs more time for that part, use the review steps below to finish it during catch-up before the shared project starts in Week 9.

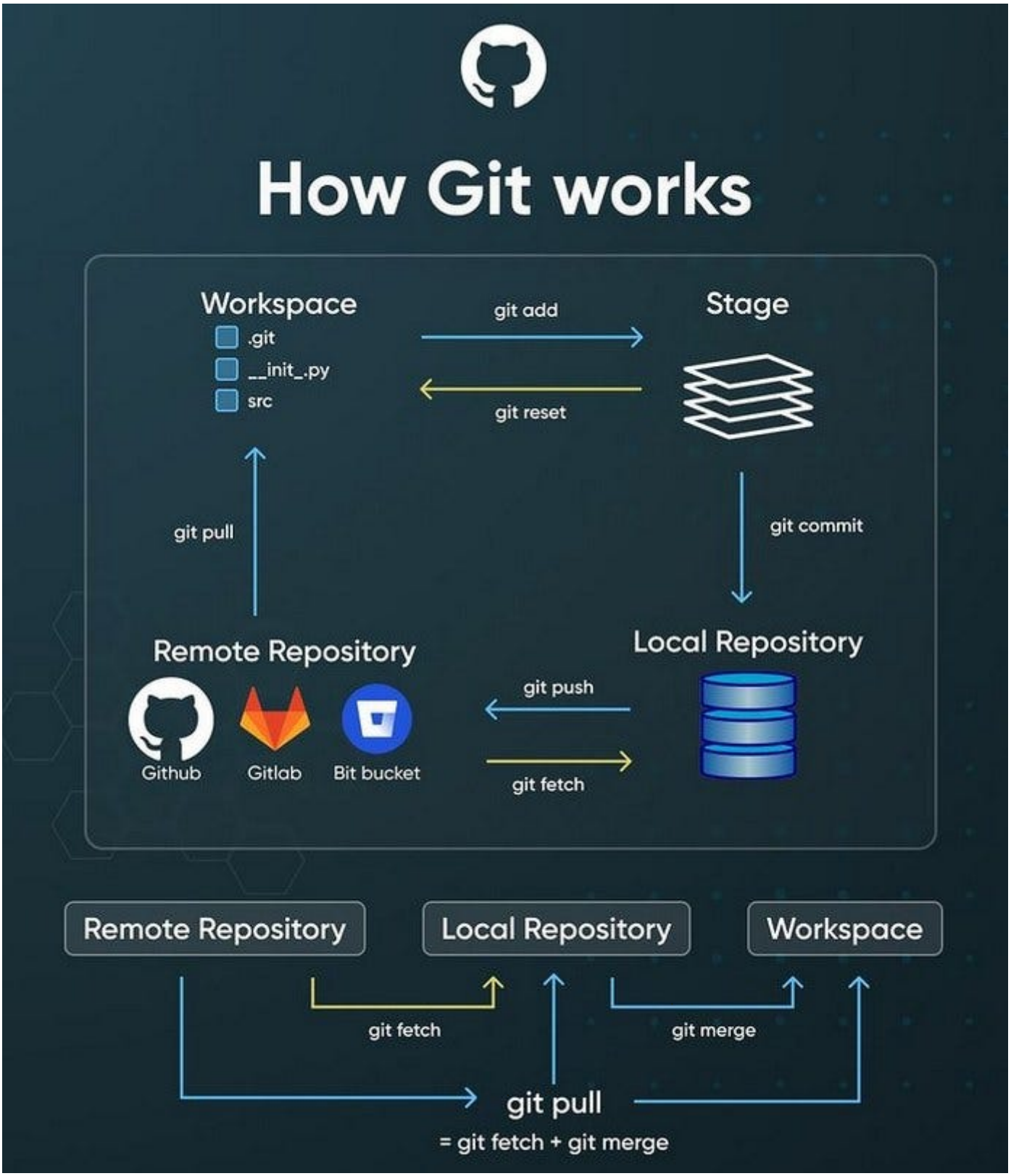
Git and GitHub

Git is a version control system. It records versions of our files so we can see what changed over time. If a button stops working, that history can help us find the change that caused the problem.

A project stored with Git is called a repository, or repo. Each saved version is called a commit. A commit contains the changes we chose and a short message that explains them.

GitHub stores Git repositories online so other people can work with us. Git records the history on our computer. GitHub gives the team a shared copy of that history.

The copy on GitHub is called a remote repository. Git usually gives the main remote the short name `origin`. Pushing sends local commits to the remote. Pulling brings remote commits back to our computer.



Opening the terminal

Open the `practice-page` folder in desktop VS Code. You should see `index.html`, `styles.css`, `script.js`, and `avatar.svg` in the Explorer.

Choose **Terminal > New Terminal**. The terminal opens at the bottom of VS Code. A terminal lets us type commands for the computer to run. Type each command after the prompt and press Enter. Do not include the word `sh` from the labels above the code blocks.

Start by checking that Git and GitHub CLI are available and that GitHub CLI knows which account to use:

```
git --version
gh --version
gh auth status
```

The first two commands should show version numbers. The last command should show your GitHub username and say that you are logged in to `github.com`. If it does not, return to the GitHub CLI setup before continuing.

Next, check the name and email Git will store with your commits:

```
git config --global --get user.name
git config --global --get user.email
```

If either command shows nothing, return to the Week 3 setup and add the missing value before continuing.

Creating the repository

Select the **Source Control** icon on the left side of VS Code. It looks like a line with connected circles. Choose **Initialize Repository**.

Initializing creates Git's history inside the folder. It does not publish the files or change what the page looks like. You may notice letters beside the filenames in Source Control. `U` means a file is untracked, so Git has not saved a version of it yet.

Return to the terminal and run:

```
git status
```

`status` tells us which branch we are on and which files have changed. The four page files should appear under **Untracked files**. If you see files from outside `practice-page`, stop and ask for help. Git may have been initialized in the wrong folder.

Staging and committing the files

Before we make a commit, we choose which changes belong in it. This step is called staging. It stops us from saving an unrelated or unfinished change by accident.

In the Source Control view, find **Changes**. Select the plus button beside `index.html`, `styles.css`, `script.js`, and `avatar.svg`. Each file should move to **Staged Changes**.

Enter this message in the box at the top:

```
Add my practice page
```

Select **Commit**. We now have a saved version in our local repository. Run these commands in the terminal:

```
git status
git log --oneline
```

`status` should say there is nothing to commit. `log` shows the commit we just made. The letters and numbers at the start of its line identify that commit.

Creating the remote and publishing from VS Code

Our commit still exists only on this computer. First, use GitHub CLI to create a private repository on GitHub and connect it to this local repository:

```
gh repo create practice-page --private --source=. --remote=origin
git remote -v
```

`gh repo create` creates the private remote repository. `--source=.` means the repository in the current folder, and `--remote=origin` connects the remote using the usual name `origin`. We left out the `--push` option, so the commit is still only on this computer.

Return to Source Control and select **Publish Branch**. If you do not see it, open the **...** menu and choose **Push**. Source Control uses the GitHub CLI credentials to upload the `main` branch.

Run `git status`. It should say that `main` is up to date with `origin/main`. You can open the repository address from `git remote -v` to confirm the four files are there, but keep editing the local files in VS

Code.

Give your partner access

A private repository can only be opened by people you invite. This access setting is the one part we need to complete on GitHub's website.

Open the repository, choose **Settings > Collaborators**, and invite your partner's GitHub username. Your partner should invite you to their repository too. Both people need to accept the invitation before the review. Copy the repository address shown by this terminal command and send it to your partner:

```
git remote get-url origin
```

Making a branch

Our first commit is on a branch called `main`. We will use `main` for the version we have checked and agreed to keep.

A branch lets us work on a change separately. When we create a branch from `main`, it starts with the same files. We can change the fun fact while `main` keeps the original. Once someone has reviewed the change, we can merge it into `main`.

Create a branch in the terminal:

```
git switch -c improve-fact
```

`switch` moves us to another branch. The `-c` option tells Git to create the branch first. The branch name appears in the lower-left corner of VS Code. You can also check it with:

```
git branch --show-current
```

It should say `improve-fact`. The files still look the same because we have only created the branch.

Changing and reviewing a file

Open `script.js`. The Week 2 example used this line:

```
fact.textContent = "I can solve a Rubik's Cube.";
```

Add a little more detail:

```
fact.textContent = "I can solve a Rubik's Cube in under two minutes.";
```

If you wrote a different fact last week, add a detail to that one instead. Save the file, then run:

```
git status
git diff
```

`status` should list `script.js` as modified. `diff` shows the exact lines that changed. A line beginning with `-` is the old version, and a line beginning with `+` is the new version. Red and green show removal and addition; they do not tell us whether the code is good or bad.

You can see the same comparison by selecting `script.js` under **Changes** in Source Control. Read it before staging. Make sure only the sentence changed.

Testing the branch

Open the same local `index.html` you have been using. Click the button. The new sentence should appear. Click again to confirm it stays, then refresh and check that the starting message returns. The picture and styles should still load.

Staging and committing in the terminal

This time, we will stage and commit with commands:

```
git add script.js
git diff --staged
git commit -m "Add the solving time to the Rubik's Cube fact"
```

`git add` stages the file. After staging, plain `git diff` no longer shows that change, so `git diff --staged` shows what the next commit will contain.

The `-m` option gives the commit its message. A message like `update` tells us very little. Naming the change helps a teammate understand the commit without opening it.

Run `git status` again. It should say the working tree is clean. The new commit is local until we push the branch:

```
git push -u origin improve-fact
```

`push` sends the commit to GitHub. `origin` identifies the remote repository, and `improve-fact` identifies the branch. The `-u` connects the local and remote branches, so later we can use the shorter command `git push`.

Creating a pull request in VS Code

We now have a change on `improve-fact` that we want to add to `main`. A pull request, usually shortened to PR, is how we propose that change on GitHub. Opening a PR does not change `main`.

Select the GitHub icon in VS Code to open **GitHub Pull Requests**. The extension may ask you to connect a GitHub account the first time you open it. Choose the same account shown by `gh auth status`. This connection lets the extension display pull requests; Git still uses GitHub CLI when it pushes and pulls.

Make sure the current branch is `improve-fact`, then choose **Create Pull Request** and set the base branch to `main`. The base receives the change from our current branch.

Read the diff and confirm it shows the sentence you meant to change. Use a title such as:

```
Add the solving time to the Rubik's Cube fact
```

In the description, explain what changed and what happened when you tested it. For example:

> I added how long it takes to solve the Rubik's Cube. I opened the page from my improve-fact branch and tried the button. It showed the new sentence, and the picture and styles still loaded.

Write what you actually checked. Add your partner as the reviewer, then create the PR.

Reviewing a pull request in VS Code

Your partner needs a local copy of your repository to try the change. Cloning downloads a repository, including its files and Git history.

1. Accept your partner's collaborator invitation.
2. In VS Code, open the Command Palette with **View > Command Palette** and run **Git: Clone**.
3. Paste the HTTPS repository address your partner sent you. Git will use the account shown by `gh auth status`.
4. Choose a parent folder that is separate from your own `practice-page`. Open the cloned repository in a new VS Code window.
5. Select the GitHub Pull Requests icon, find the PR assigned to you, and open it.
6. Select **Checkout** on the PR. VS Code switches the local files to the branch from that PR.

Read the description and open **Changes in Pull Request**. Select `script.js` to read the diff.

Open `index.html` from the cloned folder and try the button. The sentence on the page should match the added line in the diff. If the change makes sense and works, describe what you checked and approve it. If something is unclear or broken, comment on the line or request a change. Submit the review so the author can see it.

You do not have to invent a problem. A useful approval says what you read and tested.

Updating the pull request

If your reviewer asks for a change, return to your own `practice-page` window. Make sure the lower-left corner says `improve-fact`, then edit and test the file. Save another commit on the same branch:

```
git add script.js
git commit -m "Clarify the fun fact"
git push
```

The existing PR updates when the new commit reaches the same remote branch. Do not open another PR for the correction. Ask your partner to make sure the PR branch is checked out, run `git pull` in that repository's terminal, and test it again.

Merging and updating `main`

Once the change works and your partner has approved it, return to your own `practice-page` window. Open the PR description in VS Code and select **Merge**. Merging brings the branch's change into `main` on GitHub.

Your local `main` still needs that remote change. Return to the terminal in your own repository and run:

```
git switch main
git pull
```

Open `script.js` and find the new sentence. Now the local and remote copies of `main` both contain the reviewed change.

In Week 9, teams will receive different tasks in one shared merch repository. The repository will already exist, so you will clone it instead of publishing it. You will then use the same branch, commit, push, PR, review, merge, and pull steps from this lesson. Before that, we will learn React, product data, APIs, carts, and checkout through smaller examples.

Before next week

Complete the Week 4 setup before our next meeting. We will rebuild part of the practice page with React.

Common problems

What you see	What to check
The terminal cannot find `git`	Install Git, then completely close and reopen VS Code. Run `git --version` again.
The terminal cannot find `gh`	Install GitHub CLI, then completely close and reopen VS Code. Run `gh --version` again.
`gh auth status` says you are not logged in	Repeat the GitHub CLI sign-in steps, then run `gh auth status` again.
GitHub CLI shows the wrong account	Run `gh auth switch`. If the account you need is not listed, repeat the GitHub CLI sign-in steps. Then run `gh auth setup-git`.

A commit asks for your identity	Run the two `git config --global` commands from setup with your own name and commit email.
Source Control shows files outside `practice-page`	Stop before committing. Check which folder is open in VS Code and ask for help finding where Git was initialized.
`gh repo create` says the repository already exists	Stop before choosing another command. Check the existing repository with the facilitator so you do not overwrite or connect to the wrong project.
Publish Branch is missing	Run `git remote -v` and confirm that `origin` exists. Make sure the first commit is on `main`, then look for **Push** in the Source Control **...** menu.
The wrong sentence appears	Run `git branch --show-current`, save `script.js`, and refresh the local page.
The PR has nothing to compare	Confirm you committed on `improve-fact` and pushed it. Run `git status` and `git log --oneline`.
A partner cannot clone the repository	Check that the invitation was accepted by the account shown by `gh auth status`, then run `gh auth setup-git` and try the HTTPS address again.
The Pull Requests view shows the wrong account	Sign out of that extension account and connect the same account shown by `gh auth status`.
`main` still has the old sentence after merging	Switch to `main`, run `git pull`, then reopen or refresh the file.

Resources

- Install Git
- Install GitHub CLI
- GitHub CLI: Sign in
- GitHub CLI: Configure Git credentials
- GitHub CLI: Create a repository
- VS Code: Source Control
- VS Code: Git quickstart
- VS Code: Repositories and remotes
- VS Code: Work with GitHub
- Git: Recording changes
- GitHub: Invite a collaborator

If you forget a command, start with the result you need. For example, search Google for `GitHub CLI create repository from existing folder` or `Git show current branch`. Open a result from the VS Code, Git, or GitHub documentation and compare its example with your current folder and branch before running it.

Revision #6
Created 2026-10-09 17:10:05 UTC by Nishit Sharma
Updated 2026-10-14 17:18:51 UTC by Nishit Sharma