

Session 10 — Publishing Your Game

Version Control & Deployment

You're about to take your trivia game from local development to the live internet! This guide walks you through version control concepts, Git workflows, and automated deployment using GitHub Pages. Ready to share your creation with the world? Let's go!

Table of Contents

- [Understanding Version Control](#)
- [Understanding CI/CD](#)
- [Making This Project Yours](#)
- [Learning the Git Workflow](#)
- [Customizing Your Game Title](#)
- [Watching Your Deployment](#)
- [Sharing Your Live Game](#)
- [Essential Terms](#)
- [Ask the AI](#)

Accessing Your Codespace

Visit github.com/codespaces to relaunch your Codespace from Session 9.

Understanding Version Control

Before we publish your game, let's understand **version control** — the system that tracks every change to your code and lets you collaborate safely with other developers.

Version control is a system that tracks every change you make to your project files. Every time you make changes, you can create a snapshot (called a **commit**) that records exactly what you did and when. This creates a complete history of your project's evolution — you can always see what changed and when.

Why Version Control Exists

Problem	Solution
“My code worked yesterday!”	Git history shows exactly what changed
“I broke everything!”	Rollback to any previous working version
“Who changed this file?”	Blame/history shows author and reason
“How do we work together?”	Branches let multiple people code simultaneously

How Version Control Works

Version control systems like **Git** work by tracking changes to files in a special folder called a **repository**. When you're ready to save your progress, you create a **commit** — a permanent snapshot of your project at that moment.

Think of **Git** as your project's memory system. It remembers every change you've ever made, who made it, and when. If you need to work on a new feature without breaking your main code, you can create a **branch** — like a parallel universe where you can experiment safely.

When you're ready to share your work or back it up online, you **push** your commits to a remote repository on platforms like GitHub. This uploads your entire project history, not just the current files.

Version control is essential for any real development work. It tracks every decision and change you make — you can experiment confidently knowing you can always reference or return to any previous state. This tracking transforms how you approach development, making you more willing to try new ideas and refactor existing code.

Understanding CI/CD

Now let's understand **CI/CD** — the automated processes that build and deploy your code whenever you push changes.

CI/CD stands for **Continuous Integration/Continuous Deployment**. It's like having a robot assistant that automatically builds and publishes your app every time you save changes to GitHub.

CI/CD Components

Component	Purpose	Your Setup
Continuous Integration (CI)	Automatically test and build code when changes are pushed	GitHub Actions runs your build process
Continuous Deployment (CD)	Automatically deploy successful builds to production	GitHub Pages hosts your built app
Build Process	Convert source code into deployable files	Vite bundles your React app
Hosting Platform	Serve your app to users on the internet	GitHub Pages provides free hosting



From Push to Publish

This diagram shows what happens behind the scenes when you push code. GitHub detects the change, runs your build process, and deploys your updated game automatically:

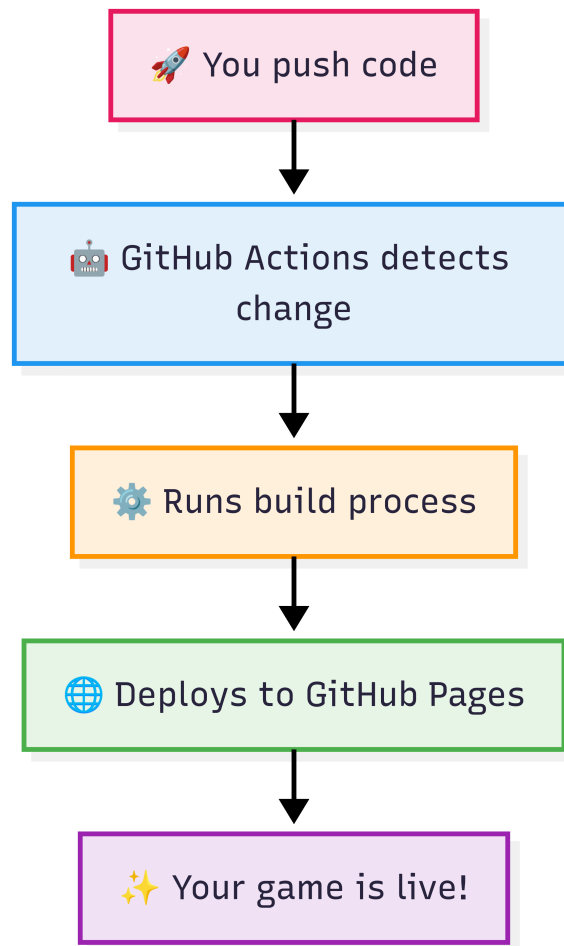


Figure: CI/CD Pipeline — GitHub Actions builds and deploys your app automatically

CI/CD eliminates manual deployment work. Instead of building and uploading files yourself, the system automatically handles everything when you push code. This is how teams deploy apps multiple times per day.

Making This Project Yours

All `git` commands should be run in your Codespace terminal

1. Check Your Git Remote Status

First, let's **see** if your Codespace has any remote connections

```
git remote -v
```

If this returns nothing (which is typical for Codespaces), you can **skip** to step 2. If it shows a remote, you'll need to **remove** it first

```
git remote remove origin
```

2. Create a New Repo on GitHub

- Go to github.com/new
- Name your repo, e.g., `trivia-quest`
- Do not add a `README` or `.gitignore` — your project already has those!
- Click “Create repository”

Upon completion, you will be redirected to your new repo page.

3. Update the Vite Build Path

Open `package.json` and update the `build` script to match your repo name

```
"build": "vite build --base=/your-repo-name/"
```

Example:

```
"build": "vite build --base=/trivia-quest/"
```

This ensures your site works correctly when deployed to GitHub Pages by matching the repo’s name to the URL path.

4. Create a Personal Access Token

Since GitHub requires authentication, you need to create a Personal Access Token (PAT):

- Go to github.com/settings/tokens
- Click “Generate new token” → “Generate new token (classic)”
- Note by entering “Codespace Git Access” or similar
- Expiration by choosing “30 days” (or longer if preferred)
- Scopes by checking both “repo” and “workflow” (needed to push GitHub Actions files)
- Click “Generate token”
- Copy the token immediately - you won’t see it again!

5. Connect Your Project with Authentication

Replace the placeholders with your actual values

```
git remote add origin \  
https://USERNAME:TOKEN@github.com/USERNAME/REPO-NAME.git
```

Example:

```
git remote add origin \  
https://babalugats76:ghp_abc123xyz@github.com/babalugats76/trivia-quest.git
```

Make a mistake typing the URL? Just remove it using `git remote remove origin` and try again. You can always use `git remote -v` to verify this connection.

Important: Keep your token private! Don't share it with anyone.

6. Push Your Code to GitHub

```
git push -u origin main
```

You should see output like this

```
Enumerating objects: 80, done.  
Counting objects: 100% (80/80), done.  
Delta compression using up to 2 threads  
Compressing objects: 100% (73/73), done.  
Writing objects: 100% (80/80), 1.80 MiB | 7.07 MiB/s, done.  
Total 80 (delta 5), reused 0 (delta 0), pack-reused 0 (from 0)  
remote: Resolving deltas: 100% (5/5), done.  
To https://github.com/your-username/your-repo-name.git  
* [new branch]      main -> main  
branch 'main' set up to track 'origin/main'.
```

Note This first push will trigger GitHub Actions, but deployment will fail because Pages isn't enabled yet. That's expected!

7. Verify Your Code Is On GitHub

- Go to your GitHub profile <https://github.com/your-username>

- Click on your new repo (e.g., trivia-quest)
- If you see your files, your push worked! You now officially own your project!

8. Enable GitHub Pages

- Go to your repo → Settings → Pages
- Under “Build and deployment”, find the “Source” dropdown
- Select “GitHub Actions”
- Save (this happens automatically)

9. Commit Your Course Progress

Now let’s **commit** all the amazing work you’ve done throughout the course! You likely have changes from previous sessions that need to be saved

```
# Stage all your changes from the entire course
git add .

# Commit your progress with a meaningful message
git commit -m "feat: complete trivia game with all course features"

# Push your complete game to trigger deployment
git push
```

This will trigger a successful deployment with all your course work since Pages is now enabled!

10. Find Your Live Site

Your site should now be live at <https://your-username.github.io/your-repo-name/>

Note It may take a few minutes for the deployment to complete. Check the Actions tab to monitor progress.

Once you complete the Git workflow in the next section, your site will be live at <https://your-username.github.io/your-repo-name/>

These steps transform your project from a shared template into your personal creation. Now you can make updates, publish changes to the web, and share your unique version with the world. Your trivia game is officially yours to customize and deploy!

Note Your game uses a special `getAssetPath()` utility function to ensure images and audio work correctly both in development and when deployed to GitHub Pages. This automatically handles the different URL paths needed for deployment.

Learning the Git Workflow

Whether you're fixing bugs, adding new features, or updating the design, you'll need a reliable way to manage and publish changes. That's where **Git** comes in.

Git is a version control tool that helps you track your progress, explain what changed, and safely update your project. It works alongside GitHub (where your code lives) and GitHub Actions (which automatically rebuilds and redeploys your site every time you push changes). In this section, you'll learn the three essential Git commands that power this workflow.

The Three Essential Commands

Every update follows this simple pattern:

```
# 1. Stage your changes (get them ready to save)
git add .

# 2. Save a snapshot with a message (like a journal entry)
git commit -m "feat(logo): customize game title to Trivia Quest"

# 3. Upload to GitHub (share with the world and trigger deployment)
git push
```

How Your Changes Go Live

This diagram shows how your changes move from your computer to the live website, step by step. Once you push your code, GitHub takes care of the rest and automatically updates your game online:

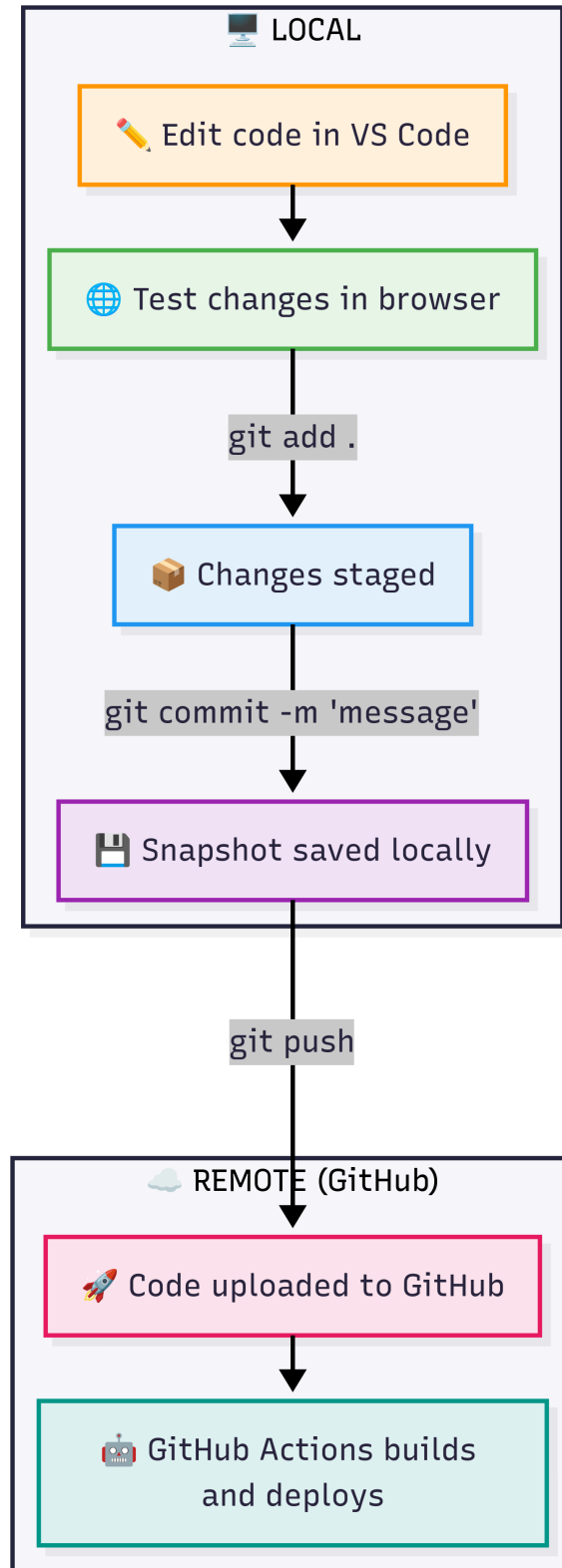


Figure: Git Development Workflow

Writing Good Commit Messages

Every commit requires a message (any text string) that explains what changed. These messages create a readable history - like a diary of your project's evolution.

While you can write any message you want, we'll use a common convention that many teams find helpful: `type(scope): description`

The message `feat(logo): customize game title to Trivia Quest` follows this format:

- `feat`: Type of change (other types: `fix`, `style`, `docs`)
- `logo`: What part of the app changed
- `customize game title to Trivia Quest`: What you actually did

Common Message Types

Type	Example	What It Means
feat	<code>feat(quiz): add timer</code>	New feature
fix	<code>fix(scoring): resolve bug</code>	Bug fix
style	<code>style(button): update colors</code>	Visual changes

Git connects your local code to your live site. With just three commands — `add`, `commit`, and `push` — you can track changes, explain what you did, and publish updates instantly. It's the core workflow behind every improvement you'll make from here on out.



Customizing Your Game Title

Now let's practice the Git workflow by making a real change to your game. We'll personalize your game's title and see the complete process in action.

Make Your Change

Let's personalize your game by changing the title that appears on the splash screen:

1. Open `src/components/GameLogo.jsx`
2. Find these two text elements:

```
<text id="title-first-line">
  Wizcamp
</text>
<text id="title-second-line">
  Realms
</text>
```

3. Change them to something that matches your game:

```
<text id="title-first-line">
  Trivia
</text>
<text id="title-second-line">
  Quest
</text>
```

4. Check your browser - you should see the new title!

Apply the Git Workflow

Now let's commit and share this change using the workflow you just learned

```
# 1. Stage your changes (prepare them for committing)
git add .

# 2. Commit your changes (create a permanent snapshot)
git commit -m "feat(logo): customize game title to Trivia Quest"

# 3. Push your changes (share with the world and trigger deployment)
git push
```

You just experienced the complete developer workflow - from making a change to deploying it live. This same pattern applies whether you're fixing a bug, adding a feature, or updating styles.

Watching Your Deployment

Now let's track your title change through the automated deployment process!

Monitor the Build Process

1. **Go** to your repository on GitHub
2. **Click** the “Actions” tab
3. **Click** on your latest workflow run (should show your commit message)
4. **Click** on “build-and-deploy” to see the deployment steps
5. **Watch** the status - wait for the green checkmark!

Test Your Live Game

- Visit your live game at `https://your-username.github.io/your-repo-name/`
- Check the splash screen - you should see your new title!
- Test the game - make sure everything still works

You just experienced the complete developer workflow - from code change to live deployment. This automation is how teams ship updates multiple times per day.

Sharing Your Live Game

Congratulations! Your trivia game is now live on the internet. Here’s how to share it with others.

Your Game's Public URL

Your game is available at

```
https://your-username.github.io/your-repo-name/
```

Example: `https://babalugats76.github.io/trivia-quest/`






Sharing Your Achievement


- **Copy** the URL and share it with friends and family
- **Test** on different devices — phones, tablets, computers
- **Share** on social media with screenshots of your game
- **Add** the URL to your GitHub profile or portfolio

Having your game live on the internet transforms it from a learning exercise into something real that others can actually use. You've built something that exists on the web — that's a genuine accomplishment!

Essential Terms

Quick reference for all the version control and deployment concepts you just learned:

Term	Definition	Why it matters
 version control	A system that tracks changes to files over time, allowing you to see history, revert changes, and collaborate safely.	Essential for any serious development work — lets you experiment fearlessly and work with others.
 Git	The version control tool that tracks changes in your code — like a save system for your project with complete history.	The industry standard that every developer uses daily — powers the add, commit, push workflow you just learned.
 repository	A folder containing your project files and their complete change history, managed by Git.	Your project's home base where all code and history live — can be local (on your computer) or remote (on GitHub).
 commit	A snapshot of your project at a specific point in time, with a message describing what changed.	Creates permanent save points you can return to — like checkpoints in a video game.
 push	Upload your local commits to a remote repository like GitHub, making them available to others.	How you share your work and trigger automated deployments — essential for collaboration and publishing.

 CI/CD	Continuous Integration/Continuous Deployment — automated processes that build and deploy code when changes are made.	Eliminates manual deployment work — your code automatically becomes a live website when you push changes.
---	--	---



Ask the AI — Publishing Your Game

You just published your trivia game to the internet using development workflows — excellent work!

Now let's deepen your understanding of version control, Git workflows, and deployment automation. Here are the most impactful questions to ask your AI assistant about today's session:

- Why is version control essential for software development, and what problems does it solve?
- What makes a good commit message, and why do they matter?
- How does the Git workflow (add, commit, push) relate to saving files on my computer?
- What are the benefits of automated deployment over manual file uploads?
- How do GitHub Actions and GitHub Pages work together to deploy websites?
- What's the difference between local repositories and remote repositories?
- How can I use version control for future projects and collaboration?