

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](#)
- [Make This Project Yours](#)
- [The Git Workflow](#)
- [Customize Your Game Title](#)
- [Watch Your Deployment](#)
- [Share Your Live Game](#)
- [Essential Terms](#)
- [Ask the AI](#)

☁️ Access 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 like having a detailed history book for your project. Every time you make changes, you can create a snapshot (called a **commit**) that you can return to later. Think of it as “save points” in a video game — you can always go back to a previous state if something goes wrong.

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.



Why This Matters

Version control is essential for any serious development work. It's like having a time machine for your code — you can experiment fearlessly knowing you can always return to a working state. This safety net transforms how you approach coding, 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



Automated Deployment Pipeline

```
---
config:
  layout: elk
  look: neo
---
flowchart LR
  A["🚀 You push code"] --> B["🤖 GitHub Actions detects change"]
  B --> C["⚙️ Runs build process"]
  C --> D["🌐 Deploys to GitHub Pages"]
  D --> E["✨ Your game is live!"]

  A:::push
  B:::detect
  C:::build
  D:::deploy
  E:::live

  classDef push fill:#fce4ec,stroke:#e91e63,stroke-width:2px
  classDef detect fill:#e3f2fd,stroke:#2196f3,stroke-width:2px
  classDef build fill:#fff3e0,stroke:#ff9800,stroke-width:2px
  classDef deploy fill:#e8f5e8,stroke:#4caf50,stroke-width:2px
  classDef live fill:#f3e5f5,stroke:#9c27b0,stroke-width:2px
```



Why This Matters

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.



Make 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/" // Update with 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:** Enter “Codespace Git Access” or similar
- **Expiration:** Choose “30 days” (or longer if preferred)
- **Scopes:** Check 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/>

💡 Why This Matters

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.

🔄 The Git Workflow

Before making changes, let's understand the essential daily workflow that every developer uses. This pattern stays consistent across all development work.

📊 The Git Workflow

```
---
config:
  layout: elk
  look: neo
---
flowchart TD
    A["📝 Edit code in VS Code"] --> B["🌐 Test changes in browser"]
    B --> C["📦 Stage with git add ."]
    C --> D["💾 Commit with descriptive message"]
    D --> E["🚀 Push to share and deploy"]
    E --> F["🤖 GitHub Actions builds and deploys"]
    F --> A

    A:::edit
    B:::test
    C:::stage
    D:::commit
    E:::push
    F:::deploy

    classDef edit fill:#fff3e0,stroke:#ff9800,stroke-width:2px
    classDef test fill:#e8f5e8,stroke:#4caf50,stroke-width:2px
    classDef stage fill:#e3f2fd,stroke:#2196f3,stroke-width:2px
    classDef commit fill:#f3e5f5,stroke:#9c27b0,stroke-width:2px
    classDef push fill:#fce4ec,stroke:#e91e63,stroke-width:2px
    classDef deploy fill:#e0f2f1,stroke:#009688,stroke-width:2px
```


The Essential Git Commands

Every change follows this three-step pattern:

```
# 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
```

Understanding Commit Messages

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

The message `feat(logo): customize game title to Trivia Quest` follows a helpful pattern: - **feat**: Type of change (other types: `fix`, `style`, `docs`) - **(logo)**: What part of the app changed - **customize game title**: 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

Why This Matters

This workflow stays consistent across all development - games, websites, mobile apps. The predictable pattern helps you build good habits and work confidently with any codebase.



Customize 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
```

💡 Why This Matters

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.

👁️ Watch 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

💡 Why This Matters

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

Share 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




Why This Matters







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.

 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.
 branch	A separate copy of your code where you can experiment with new features without breaking your main code.	Lets you try new ideas safely — if something goes wrong, your main code stays untouched.
 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.
 GitHub Actions	GitHub's automation platform that runs workflows (like building and deploying) when you push code.	Powers your automated deployment — builds your React app and publishes it without manual work.

 GitHub Pages	Free web hosting service that automatically publishes websites from GitHub repositories.	Turns your repository into a live website – perfect for React apps and portfolio projects.
--	--	--



Ask the AI – Version Control & Deployment

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?**