A Creative Technologist's Guide to GitHub AI-Powered Workflows

Tutorial Summary

Roo Code Assistant (Generated)

Derived from the Workshop Tutorial

May 17, 2025

Outline

Chapter 0: Introduction Workshop Overview

- Welcome to the comprehensive guide for creative technologists.
- Focus: GitHub, Git, VS Code, and AI-driven coding assistants.
- Goal: Establish a robust, cost-free, and powerful toolchain.
- Benefits: Version control, collaboration, free web hosting (GitHub Pages), AI assistance.

Original Workshop Agenda Highlights

- Understanding Git Version Control
- Accounts Installations (GitHub, Git, VS Code, API Keys)
- Core Git Workflow (Clone, Edit, Commit, Push)
- Collaboration, Recovery, GitHub Pages
- AI Helpers (Roo Code)

The Rationale: A Powerful Ecosystem

- **GitHub:** Centralised hub for code, portfolios, collaboration, and free hosting.
- **VS Code:** Versatile code editor with seamless Git/GitHub integration.
- **AI (Roo Code with Gemini):** Code generation, explanation, refactoring, vibe coding, persistent context.
- Collectively: Manage complex projects, foster collaboration, leverage AI with minimal cost.

Chapter 1: Why Git? Understanding Version Control

- Git addresses challenges in complex projects:
  - Safe experimentation (branches).
  - Tracking and reverting changes (commits).
  - Efficient collaboration.
  - Showcasing work.

Core Git Concepts

- **Repository (Repo):** Directory tracking all changes.
- **Commit:** Snapshot of your project at a point in time.
- **Branch:** Parallel version for independent development.
- **Remote:** Cloud-hosted copy (e.g., on GitHub).

Diagram: Local Computer ↔ Staging Area ↔ Local Repo ↔ Remote Repo (GitHub). Branches diverge and merge from Local Repo.

Git vs. GitHub

- **Git:** The underlying version control software (local, offline capable).
- **GitHub:** Web-based platform hosting Git repositories with additional collaboration tools.
- Analogy: Git is the engine, GitHub is a popular car model built around that engine.

Chapter 2: Essential Setup

- One-time process to establish your digital toolkit.
- Provides a robust, free, and powerful environment.
- Key steps:
  1. Create GitHub Account
  2. Install Git & VS Code
  3. Set up Google Cloud API Key (for AI)
  4. Configure Roo Code in VS Code

Diagram: Setup workflow from GitHub account to Roo Code config.

2a: Creating Your GitHub Account

- Go to `github.com/signup`.
- Provide email, create password, choose username (professional & memorable).
- Verify email address.
- Select the **Free** plan.
- Importance: Professional visibility, community engagement, identity.

2b: Installing Git & VS Code

Installing Git:

- macOS: Check with ‘git –version‘; install/update via Homebrew (‘brew install git‘).
- Windows: Download from ‘gitforwindows.org‘; use default options.
- Verify: ‘git –version‘.

2b: Installing Git & VS Code

Installing Git:

- macOS: Check with ‘git –version‘; install/update via Homebrew (‘brew install git‘).
- Windows: Download from ‘gitforwindows.org‘; use default options.
- Verify: ‘git –version‘.

Initial Git Configuration:

Terminal Commands

```
git config --global user.name "Your Name"
git config --global user.email "your@email.com"
git config --global init.defaultBranch main
```

2b: Installing Git & VS Code (Continued)

Installing VS Code:

- macOS: Homebrew (‘brew install –cask visual-studio-code‘) or download from ‘code.visualstudio.com‘.
- Windows: Download from ‘code.visualstudio.com‘; use default options.

2b: Installing Git & VS Code (Continued)

Installing VS Code:

- macOS: Homebrew (‘brew install –cask visual-studio-code‘) or download from ‘code.visualstudio.com‘.
- Windows: Download from ‘code.visualstudio.com‘; use default options.

Recommended VS Code Extensions:

- **Roo Code (AI Assistant):** For AI-powered coding.
- **Git Graph:** Visualise Git history.
- **Markdown Preview Mermaid Support:** For diagrams in Markdown.

2c: Setting up Google Cloud API Key

- Needed for AI features in Roo Code (Google Gemini models).
- Google Cloud Free Tier: Often includes free credits (e.g., $300 for 90 days). Payment info required for activation.
- Steps:
  1. Sign in to Google Cloud Console.
  2. Create/Select a Project.
  3. Navigate to APIs & Services ¿ Credentials.
  4. Create API Key (copy and store securely).
  5. Enable "Gemini API" in the Library.
  6. Restrict API Key (recommended for security).

2d: Configuring Roo Code in VS Code

- Open Roo Code panel (kangaroo icon). Click settings (cogwheel).
- **Profile Configuration (for Google Gemini):**
  - Profile Name (e.g., "Gemini Pro Rate Limited").
  - API Provider: Google Gemini.
  - Paste your API Key.
  - Select AI Model (e.g., "Gemini 2.5 Pro Preview").
  - Set Rate Limit (e.g., 30000ms) to manage costs.
  - Save.
- **Modes:** Ask, Code (for generation/modification), Architect.
- **Auto-Approve:** Enable Read/Write for file operations.
- Test with a simple prompt (e.g., "Explain Git").

Chapter 3: The Core Git Workflow

- Introduces fundamental Git operations.
- Cloning repositories, essential commands, basic version control tasks.

3.1 Cloning a Repository

- Creates a local copy of an existing repository (all files & history).
- **Steps:**
  1. Find Repository URL on GitHub (<> `Code` button).
  2. Clone via VS Code (Recommended: "Clone Git Repository...").
  3. Alternative: Command line ('git clone ¡url¿').

3.2 Essential Git Commands

- `git status`: Show changes.
- `git add <file> | .`: Stage changes.
- `git commit -m "msg"`: Commit staged changes.
- `git log`: Display history.
- `git pull`: Fetch & merge remote changes.
- `git push`: Upload local commits.
- `git branch`: List/create/delete branches.
- `git checkout -b <name>`: Create & switch branch.
- `git merge <branch>`: Merge branch.

Diagram: Working Directory $\xrightarrow{add}$ *Staging Area* $\xrightarrow{commit}$ *Local Repo* $\xrightarrow{push}$ *Remote Repo. Remote* $\xrightarrow{pull}$ *Local.*

3.3 Guided Exercise: Your First Git Project

- Create a project folder.
- Initialise Git repository (‘git init‘ or via VS Code).
- Create/modify a file (e.g., ‘README.md‘).
- Stage (‘git add‘) and commit (‘git commit‘) changes.
- Create a new branch (‘git checkout -b feature-branch‘).
- Make changes, stage, and commit on the new branch.
- Switch back to ‘main‘ (‘git checkout main‘).
- Merge the feature branch (‘git merge feature-branch‘).
- View history (e.g., using Git Graph extension).

Chapter 4: Collaboration & Recovery

- Explores effective collaboration using GitHub.
- Covers methods for recovering from common mistakes.

4.1 Forks vs. Branches

- **Branches (Private Team Work):**
  - Collaborators have direct write access to the same repository.
  - Work on features in separate branches within the shared repo.
  - Integrate changes via Pull Requests (PRs) within the same repo.
- **Forks (Public/Open-Source Contributions):**
  - For projects you don't have write access to.
  - A "fork" is your personal copy of another's repository.
  - Make changes in your fork, then open a PR to the original (upstream) repo.

Diagrams: Branching model (shared repo) vs. Forking model (contributing to external repo).

4.2 Opening a Pull Request (PR)

- A formal proposal to merge changes from your branch/fork into another (often ‘main‘).
- Central to code review and collaboration on GitHub.
- **Typical Workflow:**
  1. Push your feature branch to GitHub (‘git push -u origin ¡branch-name¿‘).
  2. Create PR on GitHub: Compare & pull request, select base/compare branches, add title/description.
  3. Review & Discussion: Team members comment, suggest changes. Iterate with more commits if needed.
  4. Merge PR: After approval, merge changes into the base branch. Delete feature branch (good practice).
- VS Code extensions (e.g., "GitHub Pull Requests and Issues") allow managing PRs in-editor.

4.3 Undo Recipes: Recovering from Mistakes

- **Undo last commit (keep changes):** `git reset --soft HEAD~1`
- **Discard unstaged edits (specific file):** `git checkout -- <file>`
- **Discard all unstaged edits:** `git checkout .` or `git restore .`
- **Unstage a file:** `git reset HEAD <file>` or `git restore --staged <file>`
- **Amend last commit message:** `git commit --amend -m "New msg"` (Avoid on pushed commits)
- **Revert a pushed commit (new commit):** `git revert <commit_sha>` (Safer for shared history)
- **Recover deleted branch (find SHA with `git reflog`):** `git branch <new_name> <sha>`

Undo Recipes: Important Considerations

- `git reset` **vs.** `git revert`**:**
  - `git reset` (especially `--hard`) rewrites history. Safe for local, unpushed commits. **Avoid on pushed/shared history.**
  - `git revert` creates a new commit that undoes previous changes. Safer for shared history.
- Many undo operations are available via VS Code's Source Control panel UI.

Chapter 5: GitHub Pages

- Host static websites directly from GitHub repositories for free.
- Ideal for portfolios, project showcases, documentation.
- Serves HTML, CSS, JS (and Markdown converted to HTML).
- URL Structures:
  - User/Org Site: `<username>.github.io` (from repo named `<username>.github.io`)
  - Project Site: `<username>.github.io/<repository-name>`

5.2 Setting Up GitHub Pages for a Repository

1. **Prepare Files:** Website needs at least an `index.html`. Often placed in root or `/docs` folder.
2. **Push to GitHub:** Commit and push your website files.
3. **Configure Settings:** In repo ‘Settings‘ ¿ ‘Pages‘.
4. **Choose Source:** ”Deploy from a branch”. Select branch (e.g., ‘main‘) and folder (e.g., ‘/(root)‘ or ‘/docs‘). Save.
5. **Wait for Deployment:** Site will be live at the provided URL.

5.4 Using Custom Domains

- Use your own domain (e.g., `www.yourproject.com`).
- **Steps:**
  1. Add custom domain in GitHub Pages settings.
  2. Configure DNS records with your domain registrar:
     - **Apex domain (`yourdomain.com`):** Four 'A' records pointing to GitHub IPs (e.g., 185.199.108.153) OR 'ALIAS'/'ANAME' record to `username.github.io`.
     - **Subdomain (`www.yourdomain.com`):** 'CNAME' record pointing to `username.github.io`.
  3. Wait for DNS propagation (can take up to 48 hours).
  4. Verify and Enforce HTTPS in GitHub settings.
- Roo Code can help explain DNS concepts and troubleshoot.

5.5 Advanced: Publishing from Private Repo

- Keep source code private, publish built static site to a public repo's GitHub Pages.
- Uses GitHub Actions.
- **Overview:**
  1. Private source repo, public hosting repo.
  2. GitHub Action in private repo: builds site, pushes to public repo's 'gh-pages' branch.
  3. Public repo serves Pages from 'gh-pages'.

1. **Create Personal Access Token (PAT):** With ‘repo‘ and ‘workflow‘ scopes. Store securely.
2. **Add PAT as Secret in Private Repo:** E.g., ‘DEPLOY$_T$*OKEN*‘.**Create GitHub Actions Workflow File**(*e.g.*, `.gith`
3.
    4. Checks out code.
    5. Sets up build environment (e.g., Node.js, Hugo).
    6. Runs build commands.
7. Uses an action like `peaceiris/actions-gh-pages` to deploy to ‘external$_r$*epository*‘(*publicrepo*)*usingthePAT*.*Specify*‘*publish*$_d$*ir*‘*and*‘*publish*$_b$*r*

Enable GitHub Pages in Public Repo: Serve from the deployment branch (e.g., ‘gh-pages‘).
Roo Code can help generate workflow YAML, explain syntax, and troubleshoot.

Chapter 6: AI-Powered Workflows with Roo Code

- Integrating Git/GitHub & VS Code with AI (Roo Code extension + Gemini API).
- Creates an AI-powered coding assistant directly in the editor.

6.1 What is Roo Code? (Recap)

- VS Code extension acting as an AI co-pilot.
- Capabilities:
  - Generate, explain, refactor, debug code.
  - Answer questions, generate documentation.
- Uses configured API key (e.g., Google Gemini).
- Connects AI to your file system with checks and balances.

6.2 How to Use Roo Code: "Vibe Coding"

- Iterative, exploratory, conversational approach.
- **Process:**
  1. Open Roo Code sidebar (kangaroo icon).
  2. Interact via chat: Be specific, conversational, iterate.
  3. Leverage editor context: Use code selection, active file/project.
  4. Apply suggestions, manage output (copy, or direct write if permitted).
  5. **Crucially, use Git for version control with AI changes.**
  6. Control Roo Code Modes (Ask, Code, Architect).
  7. Manage rate limiting & costs (e.g., 30s limit for free tier).

6.3 Prompt Ideas & 6.4 Local File Access

Prompt Ideas for Creative Technologists:

- Project scaffolding (e.g., "Lay out a Python eye tracker project...").
- Documentation (e.g., "Refactor README to beginner tutorial.").
- Creative asset generation/modification (e.g., "Suggest color-blind safe palette...").
- Complex document/diagram creation (e.g., Gantt charts with Mermaid).
- Understanding existing code.

6.3 Prompt Ideas & 6.4 Local File Access

Prompt Ideas for Creative Technologists:

- Project scaffolding (e.g., "Lay out a Python eye tracker project...").
- Documentation (e.g., "Refactor README to beginner tutorial.").
- Creative asset generation/modification (e.g., "Suggest color-blind safe palette...").
- Complex document/diagram creation (e.g., Gantt charts with Mermaid).
- Understanding existing code.

Power of AI with Local File Access:

- Superior contextual understanding (reads multiple project files).
- Bulk operations (generate directory structures, refactor across files).
- Persistent memory within a session.

This local integration + Git = dynamic and powerful development.

Chapter 8: LaTeX with WSL2, TeXLive, & VS Code

- For high-quality documents (complex formulae, structured layouts).
- Setup: WSL2 (Ubuntu), TeXLive, VS Code + LaTeX Workshop extension.
- Goal: Seamless LaTeX development in a familiar coding environment.

8.1-8.3: WSL2 & TeXLive Setup

Setting up WSL2 & Ubuntu:

- Install Ubuntu from Microsoft Store (e.g., Ubuntu 22.04 LTS).
- Install Windows Terminal (recommended).
- Basic Linux commands: ‘cd‘, ‘ls‘, ‘mkdir‘, ‘sudo apt update/upgrade‘.

8.1-8.3: WSL2 & TeXLive Setup

Setting up WSL2 & Ubuntu:

- Install Ubuntu from Microsoft Store (e.g., Ubuntu 22.04 LTS).
- Install Windows Terminal (recommended).
- Basic Linux commands: ‘cd‘, ‘ls‘, ‘mkdir‘, ‘sudo apt update/upgrade‘.

Installing TeXLive on Ubuntu:

- Update package lists: ‘sudo apt update‘.
- Install TeXLive: ‘sudo apt install texlive texlive-latex-extra‘.
- (Note: Installs Ubuntu repo version, generally stable).
- Explore CTAN (`ctan.org`) for packages/documentation.

8.4-8.5: VS Code & LaTeX Workshop

Connecting VS Code to WSL2:

- Install "WSL" extension (by Microsoft) in VS Code.
- Connect to WSL (Cmd/Ctrl+Shift+P ¿ "WSL: Connect to WSL").
- Open project folder within WSL environment.

8.4-8.5: VS Code & LaTeX Workshop

Connecting VS Code to WSL2:

- Install "WSL" extension (by Microsoft) in VS Code.
- Connect to WSL (Cmd/Ctrl+Shift+P ¿ "WSL: Connect to WSL").
- Open project folder within WSL environment.

Setting up LaTeX Workshop Extension:

- Install "LaTeX Workshop" (by James Yu) in WSL-connected VS Code.
- Configure build tools in workspace `.vscode/settings.json` for 'pdflatex'.
- Key settings: 'latex-workshop.latex.tools' and 'latex-workshop.latex.recipes'.
- Features: Auto-compile on save, PDF viewer, SyncTeX.
- Consider "Code Spell Checker" extension.

8.6-8.7: Roo Code for LaTeX & Manual Compilation

Enhancing LaTeX Workflow with Roo Code:

- Understand complex LaTeX syntax.
- Generate LaTeX structures (articles, beamer slides).
- Debug compilation errors.
- Find relevant LaTeX packages.
- Refine content.
- *Use Git for version control with AI-generated LaTeX.*

8.6-8.7: Roo Code for LaTeX & Manual Compilation

Enhancing LaTeX Workflow with Roo Code:

- Understand complex LaTeX syntax.
- Generate LaTeX structures (articles, beamer slides).
- Debug compilation errors.
- Find relevant LaTeX packages.
- Refine content.
- *Use Git for version control with AI-generated LaTeX.*

Manual Compilation (Understanding):

- Create simple `.tex` file.
- Compile from WSL terminal: `pdflatex mydocument.tex`.
- Confirms TeXLive setup and basic process.

Chapter 9: Reference Cheat Sheet (Git Commands)

- **Configuration (Once):**
  - `git config --global user.name "Name"`
  - `git config --global user.email "email@example.com"`
  - `git config --global init.defaultBranch main`

Chapter 9: Reference Cheat Sheet (Git Commands)

- **Configuration (Once):**
  - `git config --global user.name "Name"`
  - `git config --global user.email "email@example.com"`
  - `git config --global init.defaultBranch main`
- **Starting Project:**
  - `mkdir project; cd project; git init`

Chapter 9: Reference Cheat Sheet (Git Commands)

- **Configuration (Once):**
  - `git config --global user.name "Name"`
  - `git config --global user.email "email@example.com"`
  - `git config --global init.defaultBranch main`
- **Starting Project:**
  - `mkdir project; cd project; git init`
- **Connecting to Remote:**
  - `git remote add origin <url>`
  - `git push -u origin main`

Cheat Sheet: Everyday Workflow

- `git status` - Check changes
- `git add .` or `git add <file>` - Stage changes
- `git commit -m "message"` - Commit staged
- `git push` - Push to remote
- `git pull` - Pull from remote

Cheat Sheet: Everyday Workflow

- `git status` - Check changes
- `git add .` or `git add <file>` - Stage changes
- `git commit -m "message"` - Commit staged
- `git push` - Push to remote
- `git pull` - Pull from remote

Branching:

- `git branch` - List branches
- `git branch <name>` - Create branch
- `git checkout <name>` - Switch branch
- `git checkout -b <name>` - Create & switch
- `git merge <name>` - Merge branch
- `git branch -d <name>` - Delete merged branch

Cheat Sheet: Viewing History & Undoing

Viewing History:

- `git log` - Basic history
- `git log --oneline --graph --decorate --all` - Compact graph

Cheat Sheet: Viewing History & Undoing

Viewing History:

- `git log` - Basic history
- `git log --oneline --graph --decorate --all` - Compact graph

Undoing Things (Use with care):

- Unstage: `git reset HEAD <file>` or `git restore --staged <file>`
- Discard working dir changes: `git checkout -- <file>` or `git restore <file>`
- Amend last commit: `git commit --amend -m "new message"`

Refer to official Git documentation for more.

Conclusion Next Steps

- This tutorial provides a foundation for a modern creative technology workflow.
- Key tools: Git, GitHub, VS Code, and AI (Roo Code).
- Practice these concepts to enhance your projects and collaboration.
- Explore further: Advanced Git, specific AI prompts, deeper LaTeX integration.

Thank you for following this guide!

AI Assistance for This Presentation

- This Beamer presentation structure and initial content were generated with AI assistance (like Roo Code).
- Roo Code can help:
  - Summarise text for slides.
  - Suggest Beamer code structures.
  - Explain LaTeX or Beamer syntax.
  - Refine slide content for clarity and conciseness.
  - Generate draft content for new sections based on prompts.
- Remember to review and refine AI-generated content for accuracy and style.