GitHub

GitHub Basics for Newbie Developers

GitHub vs Git: The Difference

- **Git** (Local):

```
git add, git commit, git branch, git merge
```

- **GitHub** (Remote):

```
Push, Pull, Pull Requests, Issues, Teams, Actions
```

GitHub is

- Git + Social Coding + Project Management.
- Your portfolio management.
- **Your web server!**
    - YOUR_GITHUB_ID.github.io

Remote Management

- `origin` - Your fork/repository
- Many GitHub-related commands.

```
# View current remotes
git remote -v

# Add a remote
git remote add origin https://github.com/user/repo.git
```

Use GUI Tools (Recommended)

- In most cases, VSC GitHub features and extensions will be enough for daily work.
- Git Tower will make GitHub usage simple and easy.
- However, in some cases, you must use CLI tools; so, you need to understand how to use Git CLI commands.

Git Fetch

- Downloads **new commits/branches/tags** from remote (e.g., `origin`)
- **Does not** modify your working files or current branch
- Lets you review before integrating

```
git fetch origin
git log origin/main --oneline --graph --decorate
```

Git Pull = Fetch + Merge

- First fetches from remote
- Then merges into your current branch
- Updates your working files

```
git pull origin main
```

Equivalent to:

```
git fetch origin
git merge origin/main
```

Git Push

- Uploads your local commits to the remote
- Makes your changes available to others
- Fails if remote has new commits you don’t have (use pull/rebase first)

```
git push origin main
```

Use Cases

- Preview remote changes safely → git fetch then inspect
- Sync your branch with remote → git pull
- Publish your work → git push

Tip: You can use `git pull --rebase` to keep history linear (after git fetch).

GitHub Organizations

When we develop software in a team using GitHub, we create an organizational repository.

1. Go to GitHub.com and sign in
2. Click your profile picture → **Your organizations**

3. Click the "New organization" button.
4. Choose the Free organization plan.
5. Choose organization name and contact email.
6. Complete setup.

Teams

In the organization, we can make teams.

1. On the Tab: Teams → New Team
2. Invite your team members using their GitHub ID.
3. Set the role in the organization.

- Member/Owner
- Teams

Fork

Instead of cloning a repository, you can fork the repository.

- Your copy on GitHub
- Can modify freely
- Mainly used for open source contributions

When we don't use a fork:

- We *make a branch*, make a change, and make a pull request (PR).

When we use fork:

- We *fork* the open source project, make changes, and make a pull request (PR).

Pull Requests (PR)

Create PR

- From branch to branch (usually to `main`)
- Can be within the same repo in an organization/team
- Cross-repository PRs for open source

PR Components

- Title and description
- Reviewers and assignees
- Labels and milestones
- Linked issues

PR Creation Best Practices

Title Format: [Type] Brief description

```
feat: Add user authentication
fix: Resolve login redirect issue
docs: Update API documentation
```

Description Template (Example)

```
## Summary
Brief description of changes

## Changes Made
- Specific change 1
- Specific change 2

## Testing
- [x] Unit tests pass
- [x] Manual testing completed

## Related Issues
Closes #123
```

Code Review Process

Before merging the PR, team members should do the code review.

Reviewer Actions:

- **Comment**: General feedback
- **Suggest changes**: Inline code suggestions
- **Approve**: Ready to merge
- **Request changes**: Needs fixes

GitHub Tools for PRs

- We can use the `gh` command line tools.

```
# Create PR from command line
gh pr create --title "Add feature" --body "Description"
# List PRs
gh pr list
# Check out a PR locally
gh pr checkout 123
# Review PR
gh pr review --approve
gh pr review --request-changes --body "Needs tests".
# Merge PR
gh pr merge 123 --squash
```

- We can use the VSC Pull Request extension.
- We can even use the GitHub.com site for code review.
- The focus is the `process` of doing code review, not the tool for the process.

GitHub Projects

- Using GitHub projects, we can manage projects.
- Project Types:
    - **Table View**: Spreadsheet-like organization
    - **Board View**: Kanban-style workflow

Project Management

- Milestones:
  - Group issues by release/sprint
  - Track progress toward goals
  - Set due dates
- Automation:
  - Auto-move issues when PR created
  - Auto-close when PR merged
  - Assign based on labels

Issues Management

Issue Types:

- **Bug reports**: Something broken
- **Feature requests**: New functionality
- **Documentation**: Improve docs
- **Questions**: Need help/clarification

Issue Templates:

- Repository Settings → Features → Issues → Set up templates
- Standardizes bug reports and feature requests

Issue Reporting

Labels:

```
Priority: high, medium, low
Type: bug, enhancement, documentation
Status: needs-review, in-progress, blocked
Difficulty: good-first-issue, help-wanted
```

Workflow Example

```
📋 Backlog → 🔄 In Progress → 👀 Review → ✅ Done
```

Linking Issues and PRs

In commit messages:

```
git commit -m "Fix login bug Closes #123"
```

In PR descriptions:

```
Fixes #123
Resolves #456
Closes #789
```

Benefits of GitHub Project:

- Automatic issue closure
- Traceability between code and requirements
- Better project tracking

GitHub Actions (CI/CD)

Modern application uses CI/CD

- CI (Continuous Integration)
- CD (Continuous Development)
- GitHub supports CI/DC using Actions
- GitHub provides a virtual computer to process the requests in the workflow.

Workflow Example: .github/workflows/ci.yml

```yaml
name: CI
on: [push, pull_request]
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Run tests
        run: npm test
      - name: Deploy
        if: github.ref == 'refs/heads/main'
        run: npm run deploy
```

In this example:

- **Triggers:** Push, PR, schedule, manual
- **Use cases:** Testing, deployment, notifications

GitHub Pages

Free static website hosting directly from your GitHub repository

Perfect for:

- Personal portfolios
- Project documentation
- Course websites
- Blog sites (using Hugo or similar)

Types of GitHub Pages Sites

> **Important:** Replace `yourusername` with your actual GitHub username!

1. User/Organization Pages

- Repository name: `username.github.io`
- URL: `https://username.github.io`
- One per GitHub account

2. Project Pages

- Repository name: `any-name`
- URL: `https://username.github.io/any-name`
- Unlimited per account

Creating a User Site

Step 1: Create a Repository

1. Go to **GitHub.com** and sign in
2. Click **"New repository"** (green button)
3. Repository name: `yourusername.github.io`
4. Make it **Public**

Create an `index.html` and other files, but you can add any static files.

```
<!DOCTYPE html>
<html lang="en">
<head>
  ...
</head>
<body>
  ...
</body>
</html>
```

Create Any Repository

1. Create a **new repository** with any name (e.g., `my-project`)
2. Make it **Public**
3. Create a directory (i.e., `\docs`) to publish its content.

Enable GitHub Pages

1. Go to your repository on **GitHub.com**
2. Click **"Settings"** tab
3. Scroll down to **"Pages"** section

4. Under **"Source"**, select:

- **Deploy from a branch**
- Branch: **main**
- Folder: **/docs**

5. Click **"Save"**

Deploy Project Site

1. Add static files in the directory.
2. **Your project site will be live at:**

   `https://username.github.io/my-project`

Best Practices

- File Organization (Example)

```
yourusername.github.io/
├── index.html              # Homepage
├── about.html              # About page
├── css/
│   └── style.css           # Stylesheets
├── js/
│   └── script.js           # JavaScript files
├── images/
│   └── logo.png            # Images
└── README.md               # Repository documentation
```

Tips

- Always use lowercase filenames
- Use hyphens instead of spaces
- Include a `404.html` for error pages
- Optimize images for the web
- Test locally before pushing

Static Site Generator

- GitHub supports Jekyll, written in Ruby.
- In this course, we use Hugo, written in Go.
    - We use Markdown to make its content.

Coding with GitHub.com

- Visual Studio Code in GitHub
- GitHub Codespaces
- Gitpod allows us to use Visual Studio Code in GitHub.com

Visual Studio Code in GitHub

- We can turn GitHub into a VSCode Editor.
    - Open any file in GitHub.com
    - Change the github.com with github.dev.
    - Visual Studio Code will open the file in the web browser.

GitHub Codespaces

- In GitHub, click the green "<> Code" button.
    - Click the "Create codespaces" button.
    - Visual Studio Code will turn the GitHub repository into a VSCode project.

Gitpod

- Gitpod is "VS Code in the cloud."
- Think of it as a virtual computer specifically set up for coding.
- Gitpod is a cloud-based development environment from your Git repository.

GitHub Ecosystem

Actions Marketplace:

- Pre-built workflow actions
- Community-contributed tools
- Popular categories: CI/CD, code quality, deployment

Apps & Integrations:

- Slack notifications
- Jira integration
- Code quality tools (SonarCloud)
- Project management tools