<Git-Github-Manual>

<Training Blackbook for Developers>

Table of Content

Below Syllabus will be updated soon .....

<Setting up your local work environment>

Step 1: Set Up Your GitHub.com Account

For this class, we will utilise a public account on https://github.com/. This approach serves several purposes:

- We avoid practicing in repositories that contain genuine code.
- We deliberately introduce issues so we can instruct you on resolving them (hence, refer to #1 above).

If you already possess a github.com account, you can skip this step. Otherwise, you can establish your free account by following these steps:

1. Visit https://github.com/ and click on Sign up.
2. Select the free account option.
3. You will receive a verification email at the provided address.
4. Click on the link to complete the verification process.

Step 2: Install Git

Git is an open-source version control application essential for this class. To check if you already have Git installed, follow these steps:

- If you're using a Mac, open Terminal and type:

```
$ git --version
```

- If you're on a Windows machine, open PowerShell and enter the same command:

```
$ git --version
```

You should see output similar to this:

~ git version 2.30.1

Any version above 1.9.5 will suffice for this class!

Downloading and Installing Git

If you do not already have Git installed, you have two options:

1. **Download and install Git** from www.git-scm.com.
2. **Install it as part of the GitHub Desktop** package available at desktop.github.com.

Should you require further assistance with the installation of Git, you can find more information in the ProGit chapter on installing Git: ProGit Chapter on Installing Git.

Where is Your Shell?

Now is a good time to create a shortcut to the command line application you will want to use with Git:

- If you are working on Windows, you can use *PowerShell* or *Git Shell*, which is installed with the Git package.
- If you are working on a Mac or other Unix-based system, you can use the Terminal application.

Go ahead and open your command line application now!

Step 3: Set Up Your Text Editor

For this class, we'll use a basic text editor to work with our code. Let's ensure you have one installed and ready to use from the command line.

Choose Your Editor

You can use almost any text editor, but we recommend the following:

- GitPad
- Atom
- Vi or Vim
- Sublime
- Notepad or Notepad++

If you don't already have a text editor installed, please download and install one of the options listed above now!

Your Editor on the Command Line

Once you have installed an editor, ensure you can open it from the command line. If you are using a Mac, you need to install Shell Commands from the Atom menu.

For Windows, this is done during the installation process. To check if it's installed correctly, try the following command to open the Atom text editor:

```
$ atom .
```

Exploring

Congratulations! You should now have a working version of Git and a text editor on your system. If you have some time before class starts, here are some interesting resources to explore:

- **GitHub Explore:** Discover interesting projects on GitHub. If you find something you like, star the repository to easily find it later.
- **GitHub Training Kit:** GitHub's open-source training materials. This kit contains extra resources that may be helpful for reviewing class material. You can also contribute to the materials or suggest improvements if you need more detailed explanations.

<Getting Started with Collaboration>

We will start by introducing you to Git, GitHub, and the collaboration features we will use throughout the class. Even if you have used GitHub before, we hope this information provides a solid understanding of how to use it to build better software.

What is GitHub?

GitHub is a collaboration platform built on top of a distributed version control system called Git.

Figure 1. GitHub's beloved Octocat logo.

In addition to hosting and sharing your Git projects, GitHub offers several features to help you and your team collaborate more effectively, including:

- Issues
- Pull Requests
- Organisations and Teams

Figure 2. Key GitHub Features.

GitHub does not force you into a "one size fits all" ecosystem. Instead, it brings together all your favourite tools and may introduce you to new, indispensable ones like continuous integration and continuous deployment, helping you and your team build better software together.

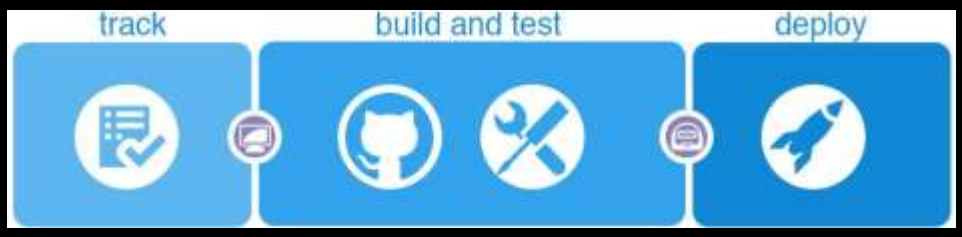

Figure 3. The GitHub Ecosystem.

What is Git?

Git is:

- A distributed version control system (DVCS)
- Free and open source
- Designed to handle everything from small to very large projects with speed and efficiency
- Easy to learn, with a small footprint and lightning-fast performance

Git outclasses many other source control management (SCM) tools with features like cheap local branching, convenient staging areas, and multiple workflows.

As we begin to discuss Git and what makes it special, try to forget everything you know about other version control systems (VCSs) for a moment. Git stores and thinks about information differently from other VCSs.

We will learn more about how Git stores your code throughout this class, but the first thing to understand is how Git works with your content.

Snapshots, not Deltas

One of the first concepts to grasp is that Git does not store your information as a series of changes. Instead, Git takes a snapshot of your repository at a given point in time. This snapshot is called a commit.

Optimized for Local Operations

Git is optimised for local operations. When you clone a repository to your local machine, you get a copy of the entire repository and its history. This means you can work on a plane, on a train, or anywhere else your adventures take you!

Branches are Lightweight and Cheap

Branches are a key concept in Git.

When you create a new branch in Git, you are simply creating a pointer to the most recent snapshot in a line of work. Git keeps the snapshots for each branch separate until you explicitly tell it to merge those snapshots into the main line of work.

Git is Explicit

Git is very straightforward. It only performs actions when you tell it to. There are no auto-saves or automatic syncing with the remote repository; Git waits for you to tell it when to take a snapshot and when to send that snapshot to the remote.

Exploring a GitHub Repository

A repository, or "repo," is the basic element of GitHub. Think of it as a project folder. However, unlike a regular folder on your computer, a GitHub repository offers powerful tools for collaboration.
It contains all the project files, including documentation, and tracks the revision history of each file. Whether you're just exploring or a major contributor, knowing your way around a repository is essential!

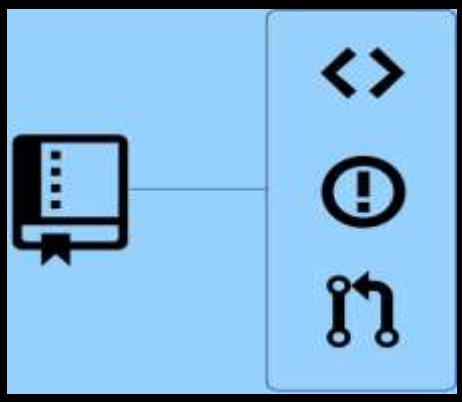

Figure 4. GitHub Repositories.

Repository Navigation

Code:

The code view shows all the files in the repository, including the project code, documentation, and other important files. This is also called the root of the project. Any changes to these files are tracked using Git version control.

Issues:

Issues are used to track bugs and feature requests. They can be assigned to specific team members and are designed to encourage discussion and collaboration.

Pull-Requests:

A Pull Request (PR) represents a proposed change to the repository, such as adding, modifying, or deleting files. PRs are often used to resolve issues.

Wiki:

Wikis on GitHub can be used to communicate project details, display user documentation, or anything else you need. GitHub helps you keep track of wiki edits.

Pulse:

Pulse is your project's dashboard. It shows information on completed work and work in progress.

Graphs:

Graphs give a detailed view of repository activity, including who has contributed, when work is being done, and who has forked the repository.

README.md:

The README.md file is recommended for all repositories. GitHub displays it below the repository. The README should explain the project and point readers to useful information.

CONTRIBUTING.md:

The CONTRIBUTING.md file describes the process for collaborating on the repository. A link to this file appears when a user attempts to create a new issue or pull request.

Using GitHub Issues

GitHub issues are used to record and discuss ideas, enhancements, tasks, and bugs.

They make collaboration easier by:

- Replacing email for project discussions, ensuring everyone has the full story.
- Allowing cross-linking to other issues and pull requests.
- Creating a single, comprehensive record of decisions.
- Making it easy to pull the right people into a conversation.

Activity: Creating a GitHub Issue

Follow these steps to create an issue in the class repository:

Activity Instructions:

1. Click the Issues tab.
2. Click **New Issue.**
3. Type a subject line for the issue.
4. Typing `- [ ]` will create a checklist in your issue or pull request.
5. When you @mention someone in an issue, they will receive a notification, even if they are not subscribed to the issue or watching the repository.
6. Typing `#` followed by an issue or pull request number (without a space) in the same repository will create a cross-link.
7. Tone can be easily lost in written communication. To help, GitHub allows you to add emoji to your comments. Simply surround the emoji id with `:` (e.g., :smiley:).

Using Markdown

GitHub uses Markdown to help you add basic text formatting to issues. Here is some

Commonly Used Markdown Syntax

```
# Header
```

- The `#` indicates a Header. # = Header 1, ## = Header 2, etc.

```
* List item
```

- A single `*` followed by a space will create a bulleted list. You canalso use a `-`.

```
Bold item
```

- Two Asterix `**` on either side of a string will make that text bold.

```
- [ ] Checklist
```

- A `-` followed by a space and `[]` will create a handy checklist in yourissue or PR.

`@mention`

- When you @mention someone in an issue, they will receive a notification- even If they are not currently subscribed to the issue or watching the repository.

`#975`

- A #followed by the number of an issue or pull request (without aspace) in the Same repository will create a cross-link.

`:smiley:`

- tone is easily lost in written communication. To help, GitHub allows you to drop emoji into your comments. Simply surround the emoji id with ::

<Understanding the GitHub Flow>

In this section, we will discuss the collaborative workflow enabled by GitHub.

The Essential GitHub Workflow

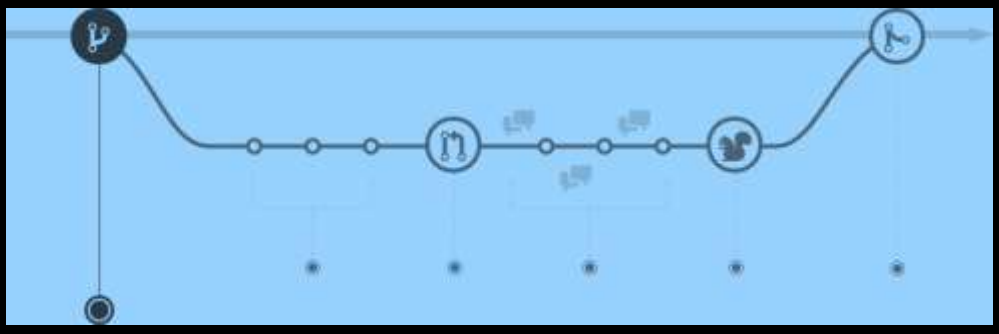

Figure 5. GitHub Workflow.

The GitHub flow is a simple workflow that lets you experiment with new ideas safely, without worrying about affecting the main project. Branching is a key concept. Everything in GitHub lives on a branch. By convention, the main version of your project lives on a branch called `master`.

When you want to try out a new feature or fix an issue, you create a new branch of the project. This new branch will initially be identical to `master`, but any changes you make will only affect your branch. This new branch is often called a "feature" branch.

As you make changes to the project files, you will commit these changes to the feature branch. When you are ready to discuss your changes, you open a pull request. A pull request doesn't need to be perfect—it's a starting point for discussion and refinement by the project team. Once the changes in the pull request are approved, the feature branch is merged into the master branch. In the next section, you will learn how to put this GitHub workflow into practice.

<Branching with Git>

The first step in the GitHub workflow is to create a branch. This allows you to keep your work separate from the master branch.

Branching Defined

When you create a branch, you are making an identical copy of the project at that moment in time. This copy is completely separate from the master branch, keeping the code on master safe while you experiment and fix issues.

Activity: Creating a Branch with GitHub

Earlier, you created an issue about a change you want to make to the project. Now, let's create a branch to add your file.

Activity Instructions:

1. Navigate to the `class repository`.
2. Click the branch dropdown.
3. Enter the branch name 'firstname-lastname-hometown'.
4. Press `Enter`.

When you create a new branch on GitHub, you are automatically switched to your branch. Any changes you make to the files in the repository will be applied to this new branch.

A word of caution. When you return to the repository or click the top-level repository link, notice that GitHub automatically assumes you want to see the items on themaster branch. If you want to continue working on yourbranch, you will need to reselect it using the branch dropdown.

<Local Git Configuration>

In this section, we'll set up your local environment to work smoothly with Git.

Checking Your Git Version

First, let's make sure Git is properly installed on your system. Open your command line application (Terminal or PowerShell) and type:

```
$ git --version
~ git version 2.7.0
```

If you don't see a version number or encounter an error, you might need to reinstall Git. Visit www.git-scm.com to find the latest version.

To see what the latest version of Git is, visit www.git-scm.com.

Git Configuration Levels

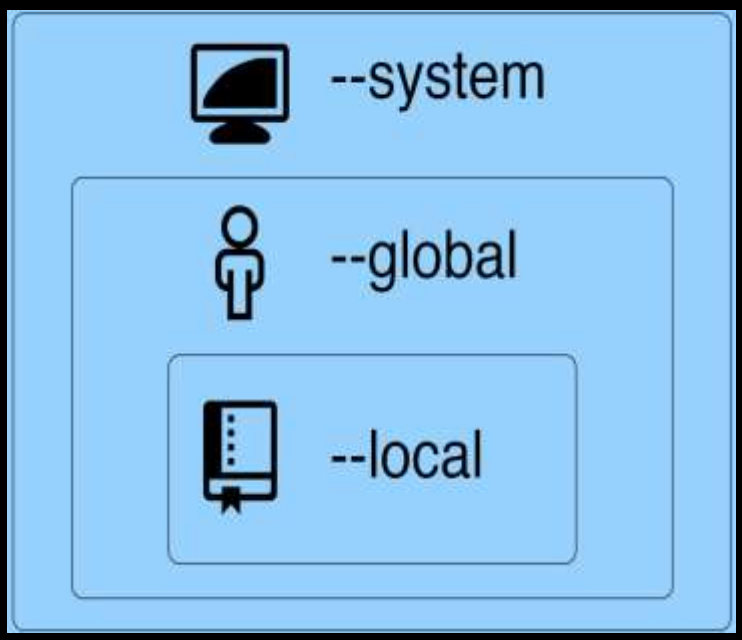

Figure 7. Git Configuration Levels.

Git allows you to configure settings at three different levels:

`--system`

These are system-wide configurations. They apply to all users onthis computer.

`--global`

These are the user level configurations. They only apply to your useraccount.

`--local`

These are the repository level configurations. They only apply to thespecific repository where they are set.

The default value for git config is `--local`.

Viewing Your Configurations

```
$ git config --list
```

If you would like to see which config settings have been added automatically, you can type `git config --list`. This will automaticallyread from each of the storage containers for config settings and list them.

```
$ git config --global --list
```

Configuring Your User Name and Email

```
$ git config --global user.name "First Last"
$ git config --global user.email "you@email.com"
```

Git uses the config settings for your user name and email address to generate a unique fingerprint for each of the commits you create.

Configuring a Default Text Editor

```
$ git config --global core.editor "atom --wait"
```

Next, we will add the default text editor git will use when you need to editthings like commit messages. If you have downloaded and installed the open-source text editor atom, the command shown above will configure itproperly.

If you are using a Mac, you will need to install the shell commands from the Atom menu before the command above will work.

If you would like to use a different editor you can find additional instructions at https://help.github.com/articles/associating-text-editors-with-git/

Configuring autocrlf

```
$ //for Windows users
$ git config --global core.autocrlf true
$ //for Mac or Linux users
```

Different systems handle line endings and line breaks differently. If you open a file created on another system and do not have this config optionset, git will think you made changes to the file based on the way your system handles this type of file.

Memory Tip: `autocrlf` stands for auto carriage return line feed.

Configuring Default Push Behavior

```
$ git config --global push.default simple
```

One final configuration option we will want to set is our default value for push. When you push changes from your local computer to the remote you

<Working Locally with Git>

If you prefer to work on the command line, you can easily integrate Git intoyour current workflow.

Creating a Local Copy of the repo

Figure 8. Cloning a repository.

Before we can work locally, we will need to create a clone of therepository.

When you clone a repository, you are creating a copy of everything in thatrepository, including its history. This is one of the benefits of a DVCS likegit - rather than being required to query a slow centralized server to reviewthe commit history, queries are run locally and are lightning fast.

Let's go ahead and clone the class repository to your local desktop.

Activity Instructions

- Navigate to the **class repository** on GitHub.
- Copy the **clone URL**.
- Open the CLI.
- Type `git clone <URL>`.
- Type `cd <repo-name>`.

Our Favourite Git command: `git status`

```
$ git status
On branch master
nothing to commit, working directory clean
```

`git status` is a command you will use often to verify the current state of your repository and the files it contains. Right now, we can see that weare on branch master, everything is up to date with origin/master and ourworking directory is clean.

Using Branches locally

```
$ git branch
```

If you type `git branch` you will see a list of local branches.

```
$ git branch --all
$ git branch -a
```

If you want to see all of the branches, including the read-only copies ofyour remote branches, you can add the `--all` option or just `-a`.

The `--all` and `-a` are actually synonyms in Git. Git often provides a verbose and a short option.

Switching Branches

```
$ git checkout <branch-name>
```

To checkout the branch you created online, type git checkout and the name of your branch. You do not need to type `remotes/origin` in front of the branch - only the branch name. You will notice a message that saysyour branch was set up to track the same remote branch from origin.

Activity: Creating a New File

Activity Instructions

- Clone the class repository to your local desktop.
- Check out your branch.
- Create a file named after your home town with the `.md` extension.
- Save the file.
- Close your text editor.

The Two Stage Commit

After you have finished making your changes, it is time to commit them. When working from the command line, you will need to be familiar with theidea of the two stage commit.

Figure 9. The Two Stage Commit - Part 1.

When you work locally, your files exist in one of four states. They are eitheruntracked, modified, staged, or committed.

An untracked file is one that is not currently part of the version-controlleddirectory.

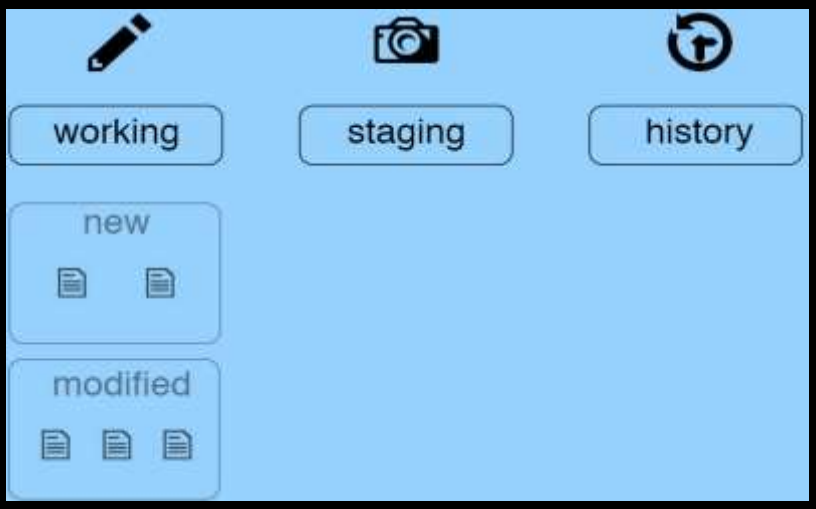

Figure 10. The Two Stage Commit - Part 2.

To add these files to version control, you will create a collection of filesthat represent a discrete unit of work. We build this unit in the stagingarea.

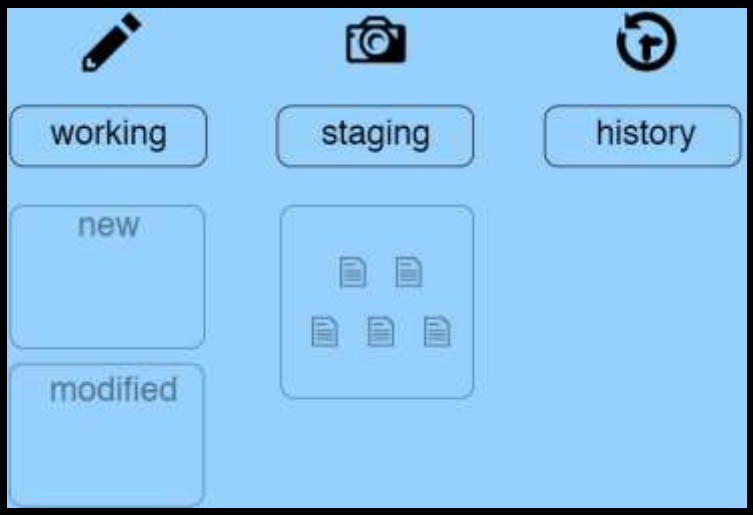

Figure 11. The Two Stage Commit - Part 3.

When we are satisfied with the unit of work we have assembled, we will commit everything in the staging area.

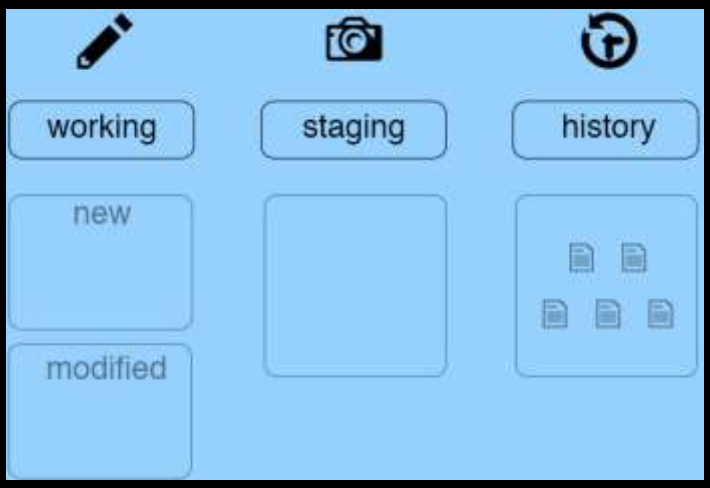

Figure 12. The Two Stage Commit - Part 4.

In order to make a file part of the version-controlled directory we will firstdo a git add and then we will do a git commit. Let's do it now.

```
$ git status
$ git add my-file.md
$ git status
$ git commit
```

When you type the commit command without any options, Git will open your default text editor to request a commit message. Simply type your message on the top line of the file. Any line without a # will be included inthe commit message.

Tips for Good Commit Messages

Good commit messages should:

- Be short. ~50 characters are ideal.
- Describe the change introduced by the commit.
- Tell the story of how your project has evolved.

To Be Continue...