Introduction to Git and GitHub

Version Control Workshop

- **Goal**: Learn the basics of version control with Git and GitHub and how it's used in software development everyday.

What is Version Control?

Why Version Control?

- Keeps track of code changes over time
- Allows collaboration on projects
- Protects against data loss and accidental overwrites

Key Terms

- **Repository**: A project's storage location
- **Commit**: A saved change in the project
- **Branch**: An independent line of development

The alternative

> Why can't I just directly edit my own files and share them to whoever?
> *Says the non-git user.*

Many issues:

- No way to track individual changes to each file
- Can't identify who made each change
- No way to review code changes

Git and GitHub

What is Git?

- **Git** is a distributed version control system
- Tracks changes and allows for reverting to previous versions

What is GitHub?

- **GitHub** is a hosting platform for Git repositories
- Facilitates collaboration with features like pull requests and issues

Setup

1. Download Git: https://git-scm.com/downloads

2. Download GitHub Desktop: https://desktop.github.com/download/

3. Create a GitHub Account: https://github.com/signup

Git Setup

Initialize a Git Repository

1. **Navigate to your project folder** in the terminal

2. Run `git init` to create a new Git repository

```
$ git init
```

- Creates a `.git` folder to track changes

Tracking Changes

Key Commands

- `git status` – See the current status of the repo
- `git add <file>` – Stage changes for a commit
- `git commit -m "message"` – Save staged changes with a message

Committing Changes

Making a Commit

1. **Stage** your changes with `git add <file>`

2. **Commit** with a message describing the changes

```
$ git add .
$ git commit -m "Initial commit"
```

- A **commit** saves your work at that point in time

Visualizing Commit History
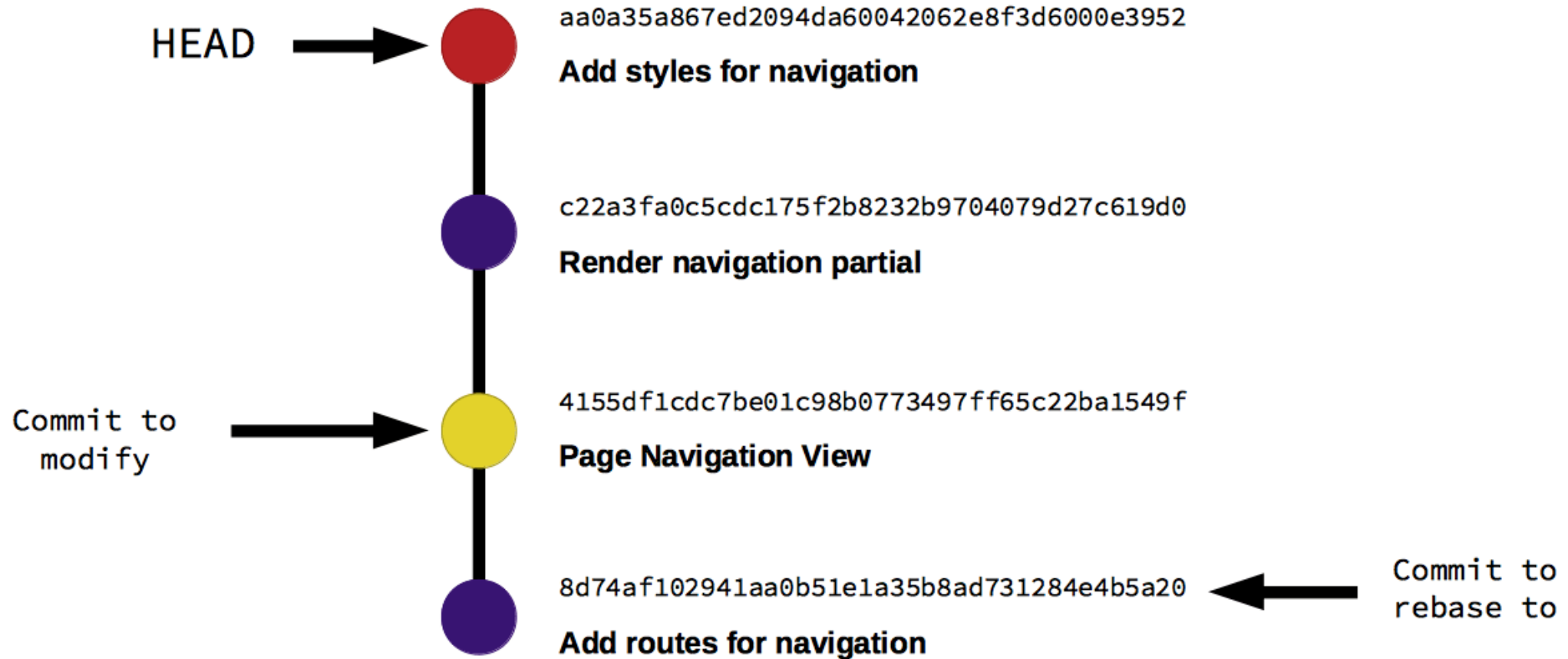
Branching

Why Branch?

- Branching allows for experimentation without affecting the main code
- Common to create branches for new features or bug fixes

Create a Branch

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

- Creates an isolated working environment

Visualizing Branching
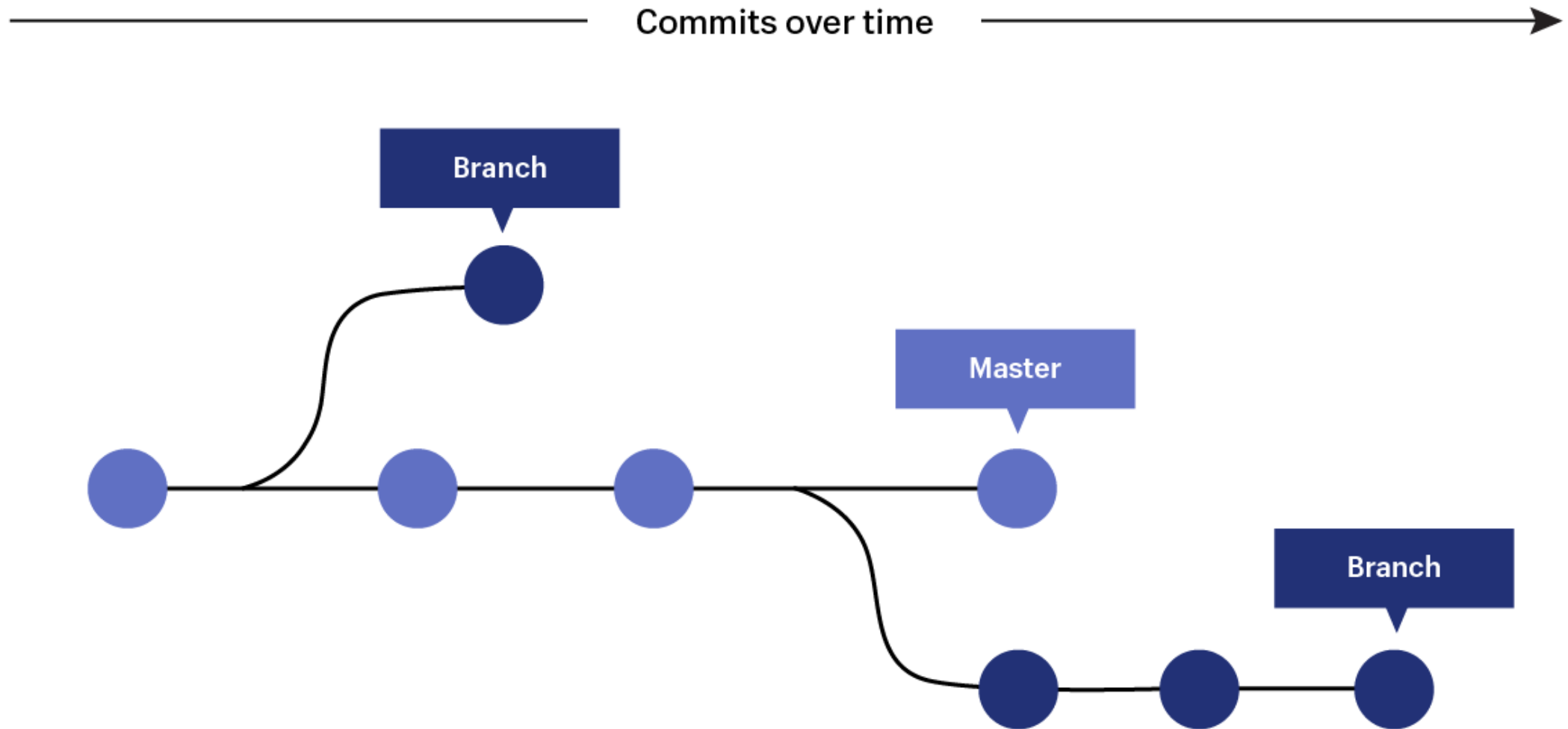
Merging

Merging Branches

- Use `git merge` to integrate changes from one branch into another

- **Example**: Merge a feature branch into the main branch

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

- **Resolve conflicts** if Git cannot automatically combine changes

An Example

Connecting to GitHub

Adding a Remote Repository

1. Create a repository on GitHub
2. Link it to your local repo

```
$ git remote add origin <url>
```

- `origin` is the default name for the remote repo

Push Changes to GitHub

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

Collaboration on GitHub

Forking and Pull Requests

- **Fork**: Copy a repository to your GitHub account
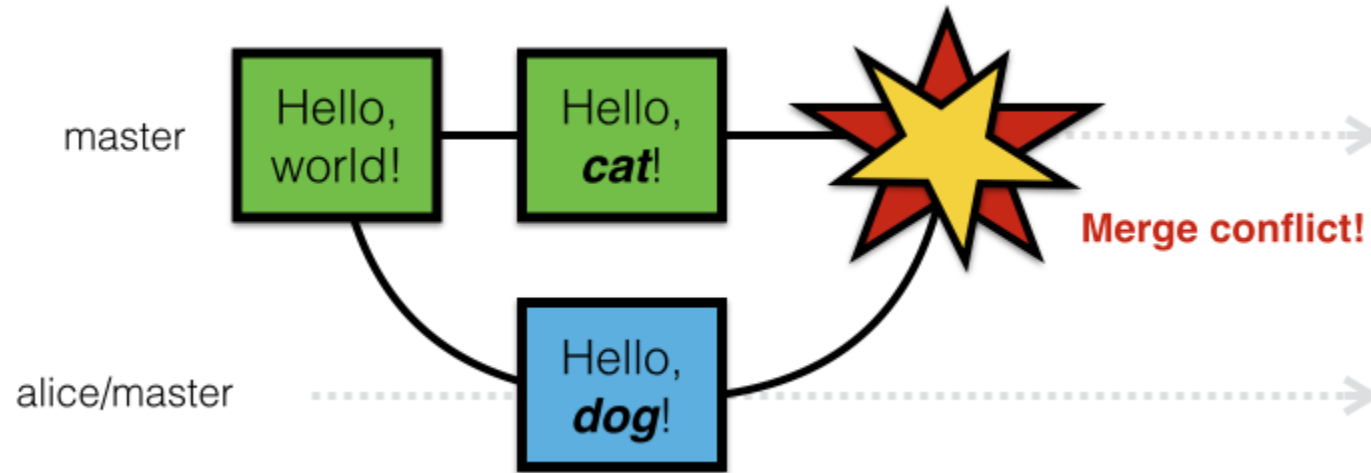- **Pull Request**: Request to merge your changes into the main repository

Steps to Contribute

1. Fork the repository
2. Make changes in your fork
3. Open a pull request to suggest changes

Resolving Merge Conflicts
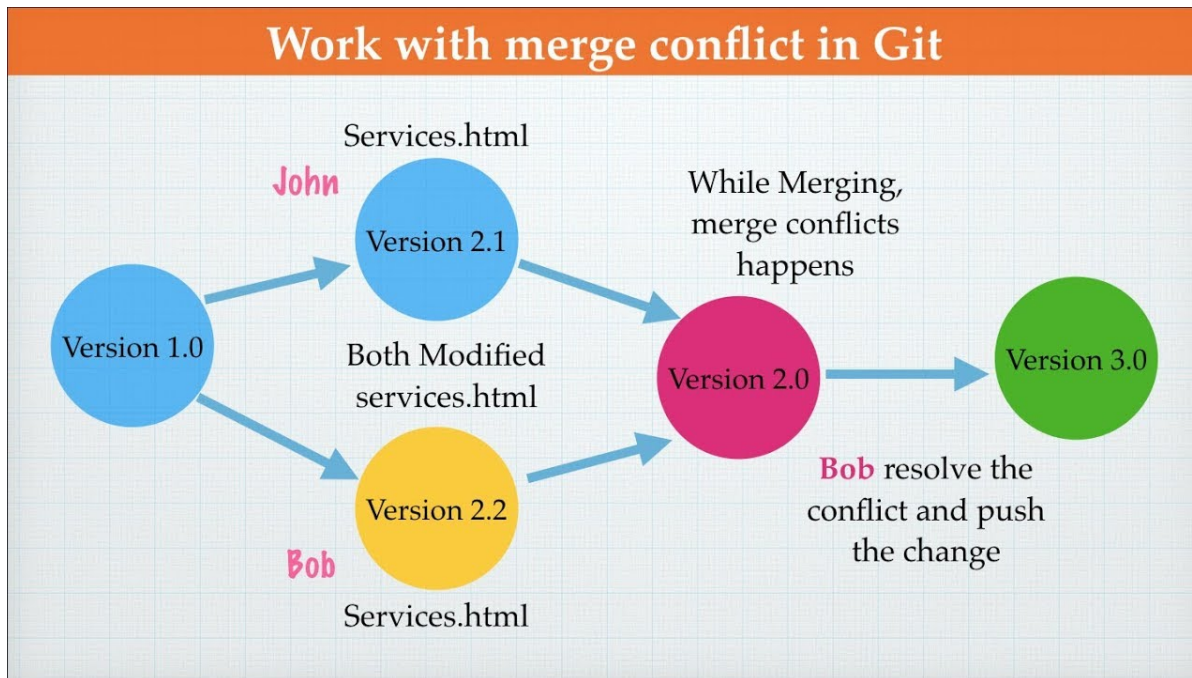
What are Merge Conflicts?

- Occur when two branches have conflicting changes

- Git will highlight conflicting files

Resolving Conflicts

1. Open the conflicted files

2. Edit to combine or reject the changes

3. Commit the resolved changes

Summary

Key Takeaways

- **Git** helps track changes in your code

- **GitHub** enables collaboration and version control

- **Commit often**, and use branches for safe experimentation

Q&A

Questions?

- Thank you for attending!
- Additional resources:
  - Git Documentation: https://git-scm.com/doc
  - GitHub Docs: https://docs.github.com/en
  - Connect with me on LinkedIn