



# Welcome to this **Co**Grammar Tutorial: Version Control Practice

The session will start shortly...

Questions? Drop them in the chat.  
We'll have dedicated moderators  
answering questions.



# Software Engineering Session Housekeeping

---

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly.

## **(Fundamental British Values: Mutual Respect and Tolerance)**

- No question is daft or silly - **ask them!**
- There are **Q&A sessions** throughout this session, should you wish to ask any follow-up questions.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: [Questions](#)

## Software Engineering Session Housekeeping cont.

---

- For all **non-academic questions**, please submit a query: [www.hyperiondev.com/support](https://www.hyperiondev.com/support)
- Report a **safeguarding** incident: [www.hyperiondev.com/safeguardreporting](https://www.hyperiondev.com/safeguardreporting)
- We would love your **feedback** on lectures: [Feedback on Lectures](#)

# Enhancing Accessibility: Activate Browser Captions

---

## Why Enable Browser Captions?

- Captions provide **real-time text for spoken content**, ensuring inclusivity.
- Ideal for individuals in noisy or quiet environments or for those with **hearing impairments**.

## How to Activate Captions:

### 1. YouTube or Video Players:

- Look for the CC (Closed Captions) icon and click to enable.

### 2. Browser Settings:

- Google Chrome: Go to *Settings > Accessibility > Live Captions* and toggle ON.
- Edge: Enable captions in *Settings > Accessibility*.

# Safeguarding & Welfare

We are committed to all our students and staff feeling safe and happy; we want to make sure there is always someone you can turn to if you are worried about anything.

If you are feeling upset or unsafe, are worried about a friend, student or family member, or you feel like something isn't right, speak to our safeguarding team:



Ian Wyles  
Designated Safeguarding  
Lead



Simone Botes



Nurhaan Snyman



Rafiq Manan



Ronald Munodawafa



Tevin Pitts

Scan to report a  
safeguarding concern



or email the Designated  
Safeguarding Lead:  
Ian Wyles

[safeguarding@hyperiondev.com](mailto:safeguarding@hyperiondev.com)

# Skills Bootcamp Progression Overview

## ✓ Criterion 1 - Initial Requirements

Specific achievements **within the first two weeks** of the program.

To meet this criterion, students need to, by no later than **01 December 2024**:

- **Guided Learning Hours (GLH):** Attend a **minimum of 7-8 GLH per week** (lectures, workshops, or mentor calls) for a total minimum of **15 GLH**.
- **Task Completion:** Successfully complete the **first 4 of the assigned tasks**.

## ✓ Criterion 2 - Mid-Course Progress

Progress through the successful completion of tasks **within the first half** of the program.

To meet this criterion, students should, by no later than **12 January 2025**:

- **Guided Learning Hours (GLH):** Complete at least **60 GLH**.
- **Task Completion :** Successfully complete the **first 13 of the assigned tasks**.

# Skills Bootcamp Progression Overview

## ✓ Criterion 3 – End-Course Progress

Showcasing students' progress nearing the completion of the course.

To meet this criterion, students should:

- **Guided Learning Hours (GLH):** Complete the **total minimum required GLH**, by the **support end date**.
- **Task Completion :** **Complete all mandatory tasks**, including any necessary resubmissions, by the end of the bootcamp, **09 March 2025**.

## ✓ Criterion 4 - Employability

Demonstrating progress to find employment.

To meet this criterion, students should:

- **Record an Interview Invite:** Students are required to record proof of invitation to an interview by **30 March 2025**.
  - **South Holland Students** are required to proof and interview by **17 March 2025**.
- **Record a Final Job Outcome :** Within 12 weeks post-graduation, students are required to record a job outcome.



**SKILLS  
FOR LIFE**

**SKILLS BOOTCAMPS**



Department  
for Education

# CoGrammar Version Control Practice



# Learning Objectives & Outcomes

- **Explain the purpose and benefits of version control systems** in the context of individual and team-based software development.
- **Perform basic Git operations**, including repository initialization, staging, committing, and viewing commit histories.
- **Implement branching workflows**, such as feature branching, and merge changes back into the main branch while handling conflicts.
- **Collaborate effectively using Git and remote repositories**, including cloning, pushing, pulling, and submitting/merging pull requests.
- **Apply best practices for maintaining a clean project history**, such as writing meaningful commit messages, and using Git commands to resolve issues.

## Poll

1. You've just added new files to your working directory and want to include them in the next commit. Which command stages all new and modified files at once?

1. `git stage all`
2. `git commit -a`
3. `git add .` ✓

## Poll

2. During a pull request review, you notice a teammate pushed changes that are unrelated to the feature being developed. What is the recommended course of action?
  1. Merge the pull request and handle the unrelated changes in a future commit
  2. Delete the teammate's branch and restart the feature
  3. Ask the teammate to remove the unrelated changes and commit them to a separate branch

# Git in High-Scale Software Development



# Git in High-Scale Software Development

Companies like **Netflix** and **Uber** handle **thousands of code updates daily** across vast, distributed teams. With **hundreds of developers** contributing to their systems, pull requests are reviewed and merged continuously, sometimes reaching **hundreds per day**. Teams rely on Git to manage frequent updates, resolve merge conflicts efficiently, and ensure code stability in real-time. Netflix, for example, deploys new code to production **hundreds of times per day**, leveraging Git and CI/CD pipelines for smooth rollouts. Similarly, Uber's microservices architecture involves **thousands of repositories**, where Git ensures synchronisation and conflict resolution across all services.

# Git in High-Scale Software Development

github.com/uber

uber

Type ↵ to search

+ ▾


Overview

Repositories 173

Projects

Packages

People 76




## Uber Open Source

Open Source Software at Uber

2.9k followers 70+ countries and counting. <http://uber.github.io/>


Follow

Pinned

 cadence Public


Cadence is a distributed, scalable, durable, and highly available orchestration engine to execute asynchronous long-running business logic in a scalable and resilient way.

Go 8.3k 800

 h3 Public


Hexagonal hierarchical geospatial indexing system

C 4.9k 468

 RIBs Public


Uber's cross-platform mobile architecture framework - Android Repository

Kotlin 7.8k 906

 kraken Public

P2P Docker registry capable of distributing TBs of data in seconds


Go 6.1k 419

 baseweb Public

A React Component library implementing the Base design language

TypeScript 8.7k 827

People



[View all](#)

Top languages

Go JavaScript Python Java Swift

Most used topics

uber android geospatial h3 machine-learning



# Introduction to Git and Version Control



# Track, Manage, and Collaborate Effectively on Codebases

The screenshot illustrates a merge conflict in a code editor. The interface is divided into three main panels:

- Incoming:** Shows the code from the 'origin/main' branch. It contains an HTML document with a title 'Resolving Github Conflicts'. The body contains two paragraphs: 'This is some code, I am changing the code' and 'This is some more code'.
- Current:** Shows the code from the 'Hubspot, origin/Hubspot' branch. It contains the same HTML document, but the body only contains the text 'Test Test'.
- Result:** Shows the merged code. It contains the HTML document with the title 'Resolving Github Conflicts'. The body contains the text 'Test Test'. A 'Complete Merge' button is visible.

At the bottom, a terminal window displays the following output:

```
hint: You can replace "git config" with "git config --global" to set a default
hint: preference for all repositories. You can also pass --rebase, --no-rebase,
hint: or --ff-only on the command line to override the configured default per
hint: invocation.
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

# What is Version Control?

- **Track Changes**
  - Like a time machine for your code
  - See who changed what and when
- **Collaborate**
  - Multiple developers, one codebase
  - No more "*final\_final\_v2.py*"

# Why is Git the Most Popular?

- Distributed version control: Work locally and sync with remote repositories.
- Widely adopted by platforms like GitHub, GitLab, and Bitbucket.
- Robust branching and merging capabilities.

# Key Commands and Concepts

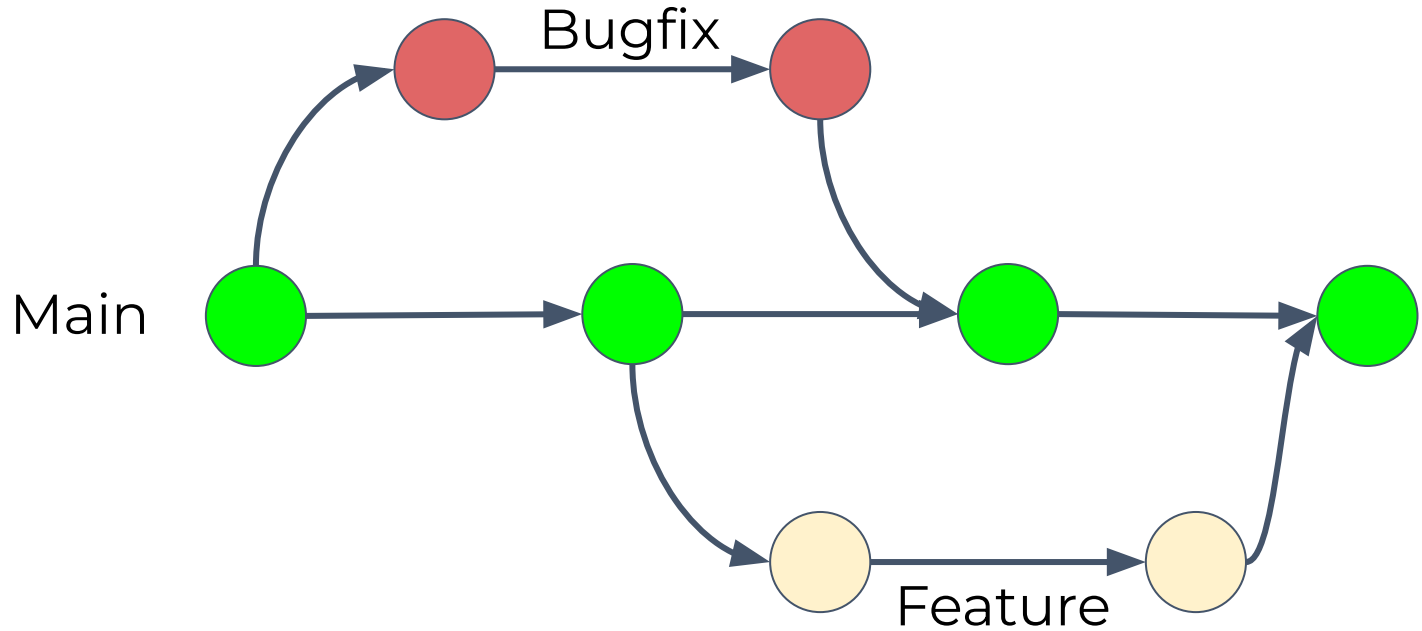


# Key Git Concepts

- **Repository:** A storage for your project.
- **Commit:** A snapshot of changes.
- **Branch:** An independent line of development.
- **Merge:** Combining branches.



# Branching Strategies: A Visual Guide



# Essential Git Commands

```
# Initialize a repository
git init

# Stage changes
git add filename
git add . # all files

# Commit changes
git commit -m "Your message"

# View history
git log --oneline
```

# Branching: Your Development Pipeline

```
# Create & switch to new branch
git checkout -b feature-name

# List branches
git branch

# Switch branches
git checkout branch-name
```

# Essential Git Commands

- Initialize a Repository: `git init`
- Track Changes: `git add`, `git commit`
- View History: `git log`
- Branching and Merging: `git branch`, `git merge`
- Remote Collaboration: `git push`, `git pull`, `git clone`

# The Role of .gitignore

- Exclude unnecessary or sensitive files from version control.
- Examples: `node_modules/`, `*.log`, `.env`.

# GitHub Overview

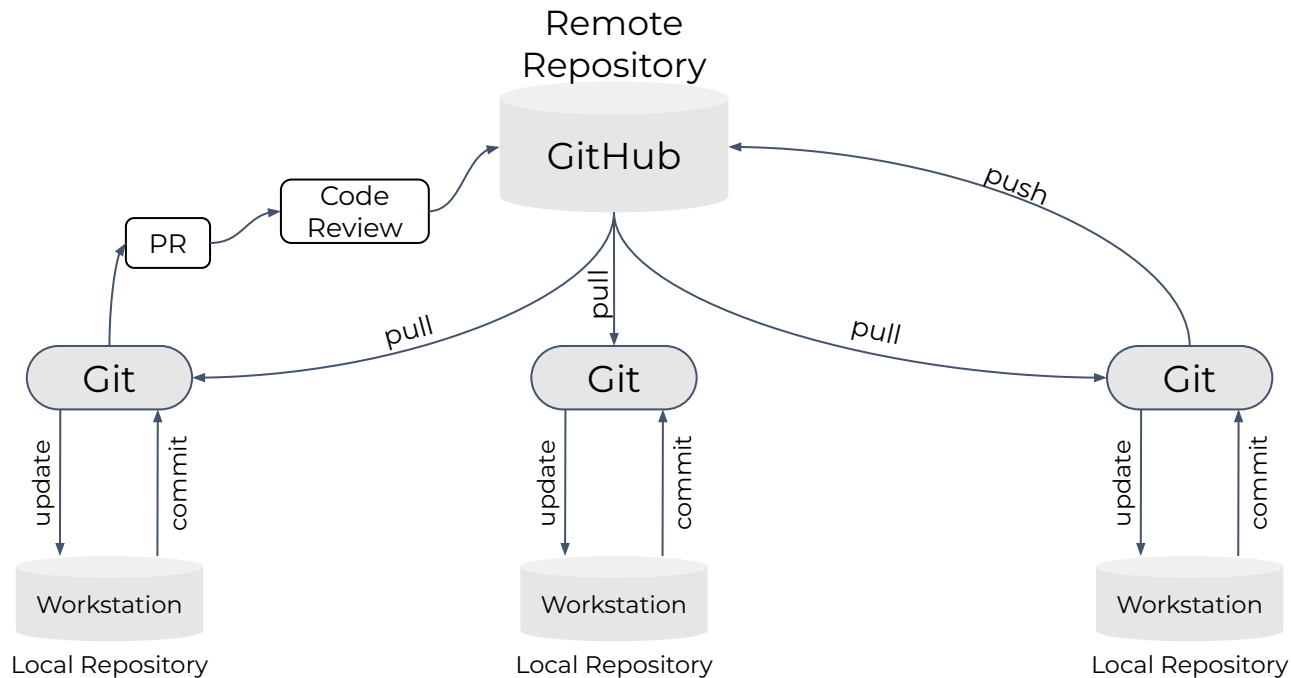




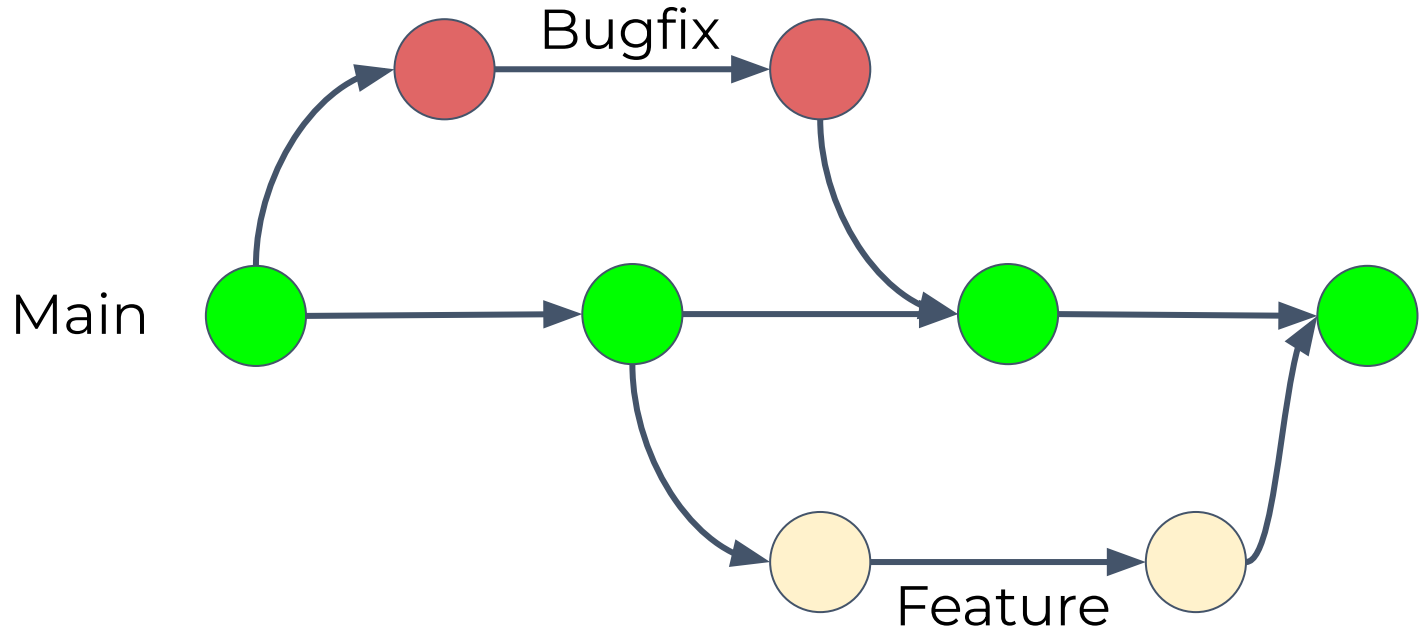
# GitHub Workflow

1. Fork repository
2. Clone locally
3. Create branch
4. Make changes & commit
5. Push to your fork
6. Create Pull Request

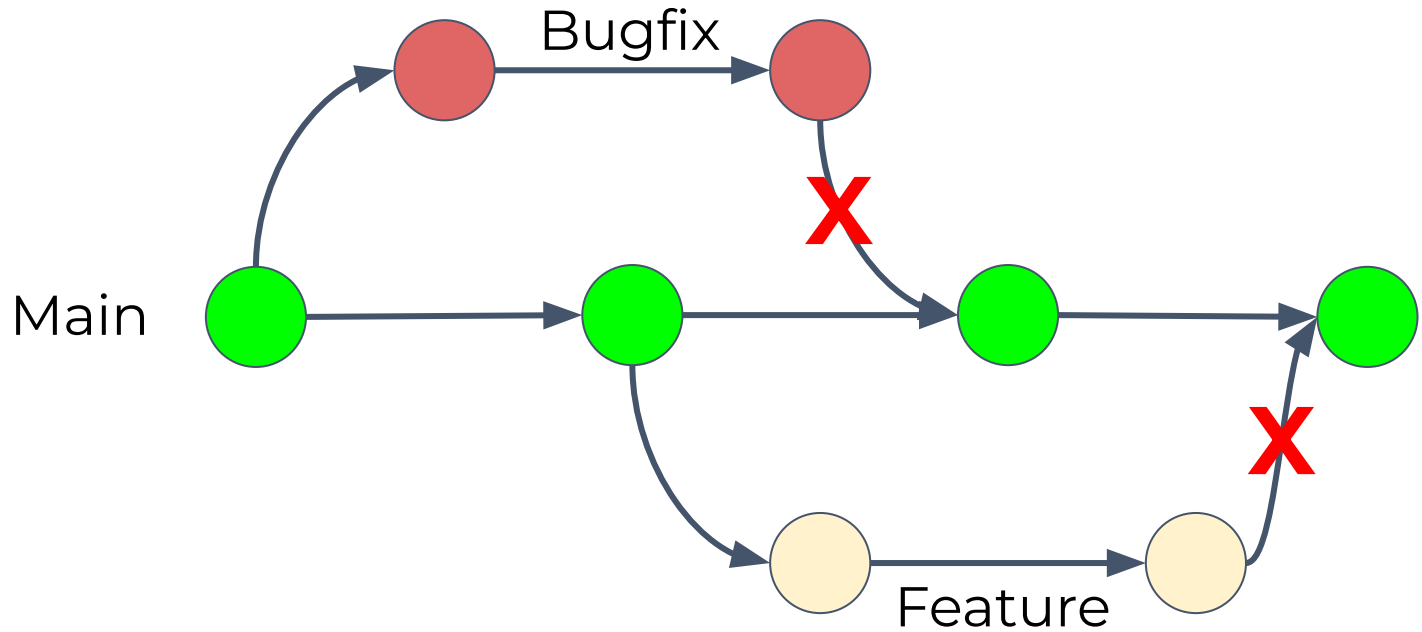
# Pull Requests and Code Reviews



# What Are Branches in Git?



# Handling Merge Conflicts



# Handling Merge Conflicts

- **Why Do Merge Conflicts Happen?**
  - Two branches modify the same line in a file.
  - Changes are made to the same file in ways Git cannot automatically combine.
- **How to Resolve Conflicts:**
  1. Identify the conflict (Git will mark files with conflicts).
  2. Edit the file to keep the desired changes.
  3. Mark the conflict as resolved (`git add`).
  4. Complete the merge (`git commit`).
- **Tools to Simplify Conflict Resolution:**
  - VS Code Git integration.
  - GitHub's conflict resolution editor.

# Best Practices & Tools





# Managing What Git Tracks with .gitignore

- **What is .gitignore?**
  - A file to tell Git which files or directories to ignore (not track).
  - Keeps sensitive or irrelevant files out of the repository.
- **Common Use Cases:**
  - Ignoring files like logs, temporary files, and environment variables.
  - Excluding OS or editor-specific files (e.g., `.DS_Store`, `*.swp`).
- **How to Use .gitignore:**
  - Add a `.gitignore` file at the root of your project.
  - Use patterns to specify ignored files (e.g., `*.log`, `/node_modules/`).
- **Best Practice:**
  - Always add `.gitignore` when initializing a project to avoid tracking unnecessary files.

# Guidelines for Effective Code Reviews

- **What is Code Review?**
  - A systematic examination of code by peers to improve quality and ensure adherence to team standards.
- **Review Guidelines:**
  - Focus on the code, not the person.
  - Check for functionality, readability, and adherence to standards.
  - Ensure the code is well-tested.
- **Providing Constructive Feedback:**
  - Be specific: "Consider renaming this variable to make it clearer."
  - Be polite: "What if we refactor this function for better readability?"
  - Avoid negative or personal comments.

# Using GitHub's Review Features Effectively

- **GitHub Review Features:**
  - Leave inline comments on specific lines of code.
  - Approve or request changes on pull requests.
  - Use suggestions for quick fixes.
- **Best Practices for Reviewers:**
  - Understand the feature's purpose before reviewing.
  - Test locally if necessary.
  - Avoid nitpicking minor issues unless they impact functionality.
- **Best Practices for Submitters:**
  - Write clear commit messages.
  - Use meaningful PR descriptions (what/why/how).
  - Address feedback promptly and update your PR.

## Poll

What is a common reason merge conflicts occur in Git?

1. Two developers created new files with the same name in different branches.
2. The same line in a file was modified by different branches.
3. Git cannot track changes made to binary files.

## Poll

You've accidentally committed sensitive API keys to your repository. What's the most effective long-term solution?

1. Delete the commit and force push to remove the history
2. Add the file to .gitignore, rotate the API keys, and store them in environment variables
3. Simply remove the keys from the current version and commit the change

# Practical: Version Control for Collaborative Web Development





# Practical: Version Control for Collaborative Web Development

- **Objective:** Manage a simple HTML/CSS project using Git and GitHub, focusing on version control and collaboration. Set up a repository, work with branches, resolve conflicts, and collaborate with a partner.
- **Steps to Implement:**
  - **Set Up the Project:**
    - i. Initialise a Git repository locally and add an `.gitignore` file.
    - ii. Create a GitHub repository and push the local project to it.
  - **Work on Features with Branching:**
    - i. Create a feature branch (e.g., `feature-header`).
    - ii. Add and commit changes to the project (e.g., modify `index.html` to include a header).
  - **Merge and Resolve Conflicts:**
    - i. Create another branch with conflicting changes.
    - ii. Attempt to merge and resolve conflicts manually.
  - **Collaborate Using GitHub:**
    - i. Clone the repository, create a new feature branch, and push changes.
    - ii. Open a pull request (PR), review a partner's PR, and merge it into the main branch.
  - **Use `.gitignore`:**
    - i. Add file patterns to `.gitignore` (e.g., `*.log` or `.DS_Store`).
    - ii. Confirm ignored files are not tracked in the repository.

# Lesson Conclusion and Recap

Recap the key concepts and techniques covered during the lesson.

- **Branching Strategies:** Key strategies like feature branches and main branch usage help organise work and streamline collaboration.
- **Merging and Conflict Resolution:** Merging branches and handling merge conflicts ensure smooth integration of changes from different contributors.
- **Using .gitignore:** The .gitignore file helps manage which files to track or ignore, keeping the repository clean and focused.
- **Pull Requests (PRs) and Code Reviews:** PRs and code reviews support collaborative development, allowing team members to review, discuss, and improve code before merging.
- **Remote Repositories and GitHub Collaboration:** Leveraging platforms like GitHub enhances teamwork, making it easier to share, collaborate, and track project progress.

# Resources

## Resources

- **Software:**

- [Git - Downloading Package](#)
- [Download GitHub Desktop](#)

- **Additional Resources**

- [Hello World - GitHub Docs](#)
- [Get started with GitHub documentation](#)

- **Books:**

- [Pro Git book](#)

# Questions and Answers



# Thank you for attending



Department  
for Education

CoGrammar

