

Code Review and Practices

Week 6

Code is read much more often than it is written.

— Guido van Rossum

Objectives

- Understand the importance of code review
- Try tools & techniques for effective reviews
- Engage in hands-on review practice

Contents

- Code review
- Practice
 - Git PR template setting
 - Linter set-up

Code Review

Code Review

- It is the systematic examination of software source code
- It is performed by developers other than the code author
- It aims to find and fix mistakes overlooked during development

Benefits of Code Review (1/2)

- Improved code quality
 - Code reviews catch mistakes, leading to more robust code
 - Encourages adherence to coding standards and best practices
- Reduced errors and vulnerabilities
 - Many eyes on the code mean a higher chance of catching errors
 - Reduces the risk of security vulnerabilities slipping through

Benefits of Code Review (2/2)

- Knowledge sharing
 - New team members can learn from experienced developers
 - Facilitates a shared understanding of the codebase
- Collaborative team culture
 - Code reviews promote a culture of collaboration and openness
 - Developers learn to give and receive feedback constructively
- Cost efficiency
 - Finding and fixing issues during review is far cheaper than after software goes live
 - Reduces technical debt by addressing issues in their infancy

Code Review Best Practices (1/3)

- Be constructive, not critical
 - Focus on the code and its quality, not on the individual developer
 - Provide feedback that is actionable and positive
- Review regularly & keep it manageable
 - Review code frequently to prevent the buildup of changes
 - Limit the amount of code to be reviewed in one session
- Automate
 - Utilize tools (like linters) to catch trivial issues automatically
 - This allows to focus on deeper, architectural, or logical issues

Code Review Best Practices (2/3)

- Start with a clear understanding
 - Before diving deep, understand the purpose of the changes
 - Use PR descriptions or initial comments to get context
- Prioritize important issues over coding style
 - Prioritize significant issues like potential bugs, security vulnerabilities, or performance concerns over coding style

Code Review Best Practices (3/3)

- Ensure good communication
 - Clarify doubts rather than making assumptions
 - Encourage a two-way conversation
 - Remember, it is a review discussion, not a monologue
- Close the loop
 - Ensure all raised issues are addressed
 - Use a system to track and manage feedback
 - Ensure nothing slips through

Code Review Checklist

1. **Design.** Is the code well-designed for the system?
2. **Functionality.** Does the code behave as intended?
3. **Complexity.** Could the code be made simpler?
4. **Tests.** Does the code have automated tests?
5. **Coding Styles**
 - a. Naming: Did the developer choose clear names?
 - b. Comments: Are the comments clear and useful?
 - c. Style: Does the code follow our style guides?
6. **Documentation.** Was relevant documentation updated?

Types of Code Review

- Formal code review
 - Rigorous process involving multiple roles (e.g., author, reviewer, inspector, moderator, scribe)
- Lightweight code review
 - Over-the-shoulder: Look over the shoulder to check the code
 - Walkthroughs: The author explains the code gathering feedback
 - **Pull requests**: Review code before merging into the main branch
- **Pair programming**
 - Two developers work together at one workstation, continuously reviewing the code on the fly

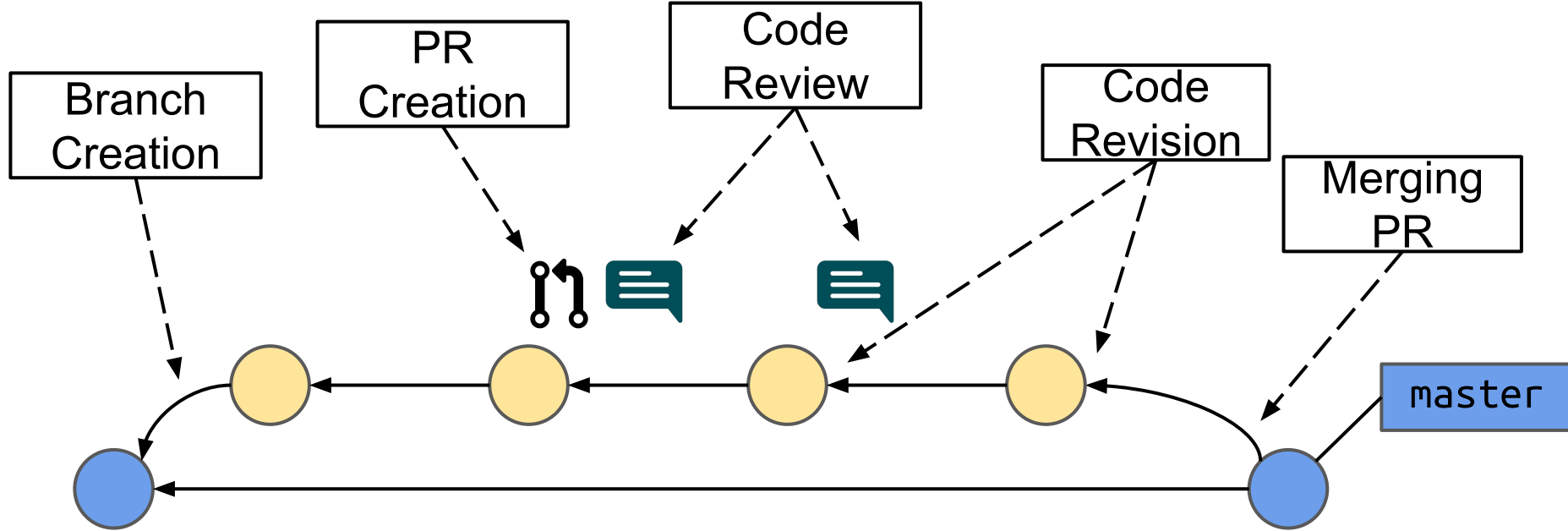
Pair Programming

- Collaboration in real-time
 - Pair programming is an instantaneous code review process
- Shared knowledge
 - Both partners bring their unique skills and knowledge
- Continuous feedback
 - The navigator provides immediate feedback to the driver
- Two sets of eyes
 - Reduces the likelihood of mistakes

Pair Programming Benefits

- Higher code quality
 - The continuous feedback can result in more maintainable code
- Shared responsibility
 - Spreads accountability, so no single person is responsible for mistakes
- Faster problem solving
 - Two heads often think better than one
- Reduced code review time
 - Code often needs fewer revisions after pair programming sessions
- Knowledge transfer
 - Share insights and learn from each other's experiences and perspectives

Pull Request (PR) Review Workflow



Pull Request (PR) Review Workflow

1. **Branch Creation.** Add features, fix bugs, refactor, etc.
2. **PR Creation.** Create a PR against the main branch
3. **Code Review.** Other members review the code
4. **Code Revision.** Revise and repeat from step 3 if needed
5. **Merging.** If reviewers approve the PR, and integration tests have passed, merge it and close the branch

Best Practices for Pull Request (1/3)

- Clear title
 - A concise and descriptive title that gives a snapshot of changes
 - "Add user authentication for the admin portal" instead of "Updates"
- Detailed description
 - A thorough explanation of what the PR does, why the change is necessary, and any relevant context
 - Include any related issues (e.g., "Related to issue #123")

Best Practices for Pull Request (2/3)

- Logical commits
 - Break down changes into smaller, logical commits
 - Each commit should represent a single "idea" or "change" and have a descriptive commit message
- Include relevant documentation
 - Prepare for a how-to-use documentation if the PR introduces a new feature
 - Update existing documentation if necessary

Best Practices for Pull Request (3/3)

- Visuals when relevant
 - Include screenshots or even demo videos for frontend changes
 - Provides a clear understanding of visual changes
- Testing
 - Describe what testing has been done and the results
 - Highlight any areas that need particular attention
- Request reviewers wisely
 - Choose reviewers who have expertise in the modified code
 - Include both someone deeply familiar with the project and someone less familiar to get diverse feedback

Git PR Template Exercise

Git PR Template

- A default markdown file(`.github/pull_request_template.md`) automatically included when creating a Pull Request
- Guides the author to provide all required information
- Ensures consistency and quality across all PRs

Git PR Template Example

PR Title: [Descriptive title summarizing the change]

Related Issue(s):

Link or reference any related issues or tickets.

PR Description:

[Provide a brief summary of the changes made.]

Changes Included:

- [] Added new feature(s)
- [] Fixed identified bug(s)
- [] Updated relevant documentation

Screenshots (if UI changes were made):

Attach screenshots or GIFs of any visual changes. (Only for frontend-related changes)

Notes for Reviewer:

Any specific instructions or points to be considered by the reviewer.

Reviewer Checklist:

- [] Code is written in clean, maintainable, and idiomatic form.
- [] Automated test coverage is adequate.
- [] All existing tests pass.
- [] Manual testing has been performed to ensure the PR works as expected.
- [] Code review comments have been addressed or clarified.

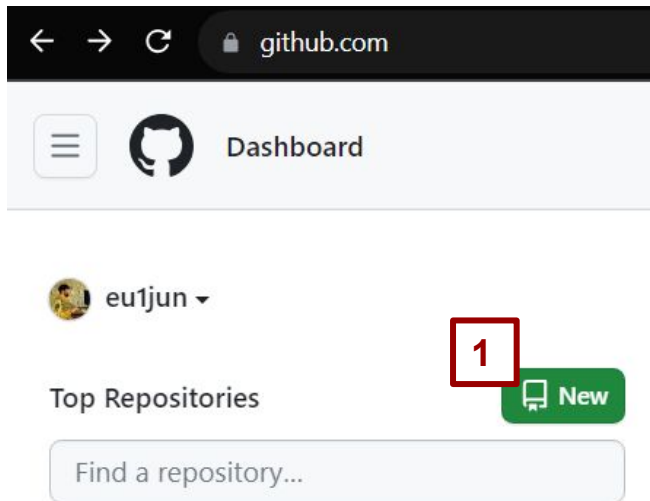
Additional Comments:

Add any other comments or information that might be useful for the review process.

Add PR Template to GitHub Repo (1/4)

1. Make a new repository

a. Create one with README



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk ().*

Repository template

No template ▾

Start your repository with a template repository's contents.

Owner *

eu1jun ▾

Repository name *

template-exercise

✔ template-exercise is available.

2

Initialize this repository with:

☒ Add a README file

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: None ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: None ▾

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set `main` as the default branch. Change the default name in your settings.

☐ You are creating a private repository in your personal account.

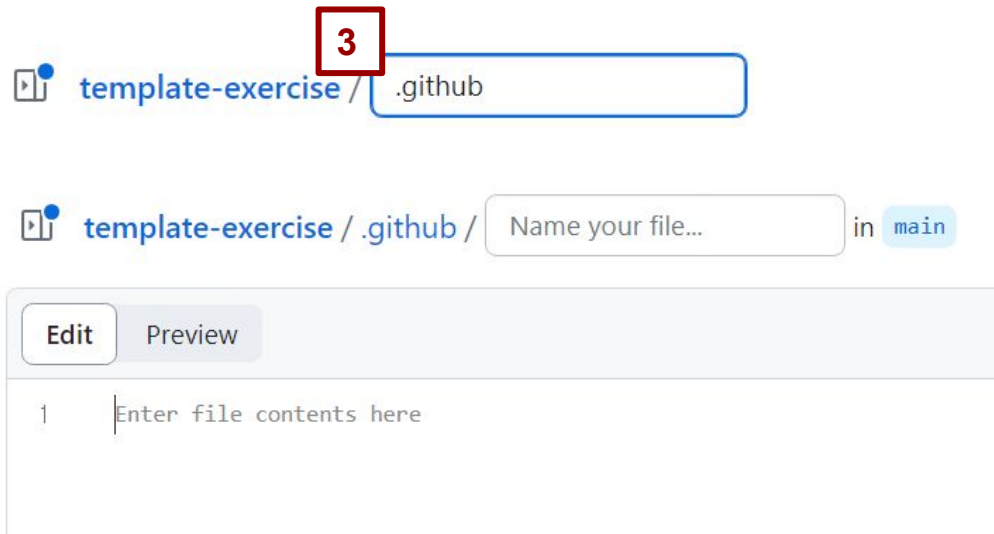
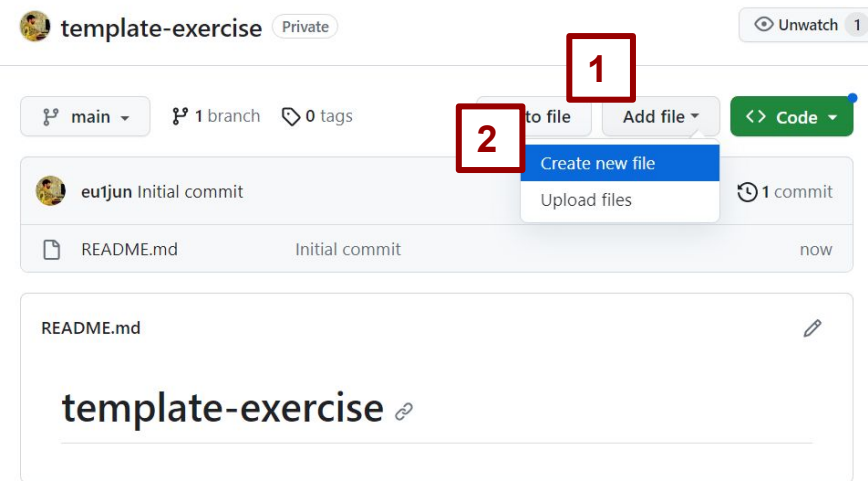
3

Create repository

Add PR Template to GitHub Repo (2/4)

2. Create a .github/ directory

- Type `.github/` in text box, don't forget '/'



Add PR Template to GitHub Repo (3/4)

3. Name a file `pull_request_template.md`

- Paste the previous template
- Click Commit Changes...



Add PR Template to GitHub Repo (4/4)

4. Commit (and push)

- In GitHub, commit directly to the main branch
- If you've working locally, commit and push them to repository

Commit changes

Commit message

Create pull_request_template.md

Extended description

Add an optional extended description..

Commit Email

onsaga1@snu.ac.kr

☒ Commit directly to the main branch

☐ Create a new branch for this commit and start a pull request

[Learn more about pull requests](#)

Cancel Commit changes

```
git add .github/pull_request_template.md
git commit -m "Add PR template"
git push origin main
```

main template-exercise / .github /

Go to file

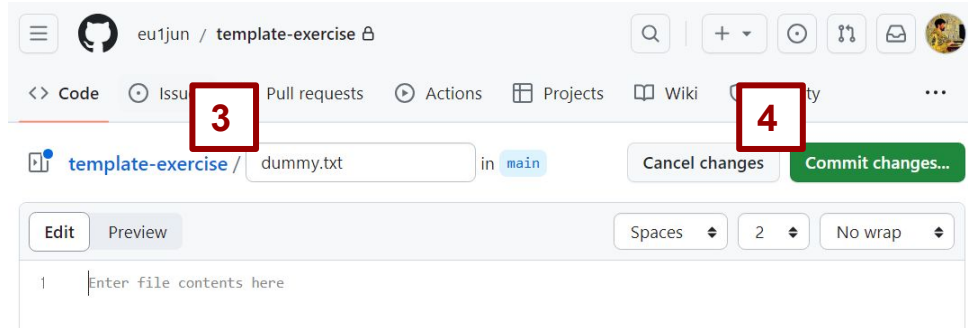
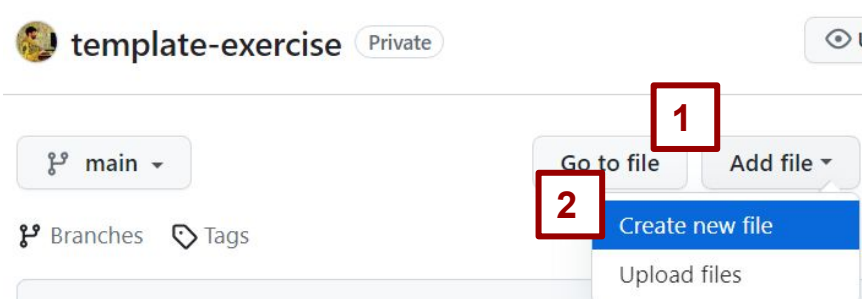
Add file

eurjun Create pull_request_template.md 63f0344 · now History

Name	Last commit message	Last commit date
..		
<u>pull_request_template.md</u>	Create pull_request_template.md	now

Test PR Template (1/3)

1. Create a new file to test the template
 - a. Proceed to the next slide before the commit



Test PR Template (2/3)

2. Commit and make PR

- Check the option for a new branch
- Set the branch name
- Propose changes

The screenshot shows a 'Propose changes' dialog box with the following fields and options:

- Commit message:** A text input field containing 'Create dummy.txt'.
- Extended description:** A larger text input field with the placeholder text 'Add an optional extended description..'. A small icon is visible at the bottom right of the field.
- Commit Email:** A dropdown menu showing 'onsaga1@snu.ac.kr'.
- Commit options:** Two radio buttons are present. The first is 'Commit directly to the main branch'. The second is 'Create a new branch for this commit and start a pull request', which is selected. Below this second option is a link that says 'Learn more about pull requests'.
- Branch name:** A text input field containing 'test/pr_template'.
- Buttons:** At the bottom right, there are two buttons: a grey 'Cancel' button and a green 'Propose changes' button.

Three red boxes with white numbers are overlaid on the image to highlight specific steps:

- 1** is placed over the 'Create a new branch for this commit and start a pull request' radio button.
- 2** is placed over the 'test/pr_template' text in the branch name field.
- 3** is placed over the green 'Propose changes' button.

Test PR Template (3/3)

3. Now, you applied the template to your PR

a. Check with the preview page, too

base: main ← compare: test/pr_template ✓ Able to merge. These branches can be merged.

Create dummy.txt

Write Preview

H B I

PR Title: [Descriptive title summarizing the change]

Related Issue(s):
Link or reference any related issues or tickets.

PR Description:
[Provide a brief summary of the changes made.]

Changes Included:

- [] Added new feature(s)
- [] Fixed identified bug(s)
- [] Updated relevant documentation

Screenshots (if UI changes were made):
Attach screenshots or GIFs of any visual changes. (Only for frontend-related changes)

Notes for Reviewer:
Any specific instructions or points to be considered by the reviewer.

Create dummy.txt

Write Preview

PR Title: [Descriptive title summarizing the change]

Related Issue(s):
Link or reference any related issues or tickets.

PR Description:
[Provide a brief summary of the changes made.]

Changes Included:

- ☐ Added new feature(s)
- ☐ Fixed identified bug(s)
- ☐ Updated relevant documentation

Screenshots (if UI changes were made):
Attach screenshots or GIFs of any visual changes. (Only for frontend-related changes)

Notes for Reviewer:
Any specific instructions or points to be considered by the reviewer.

Reviewer Checklist:

- ☐ Code is written in clean, maintainable, and idiomatic form.
- ☐ Automated test coverage is adequate.

Linter

Lint

- Tools that analyze source code to detect programming errors, bugs, stylistic errors, and suspicious constructs
- Popular linters
 - JavaScript: ESLint
 - Python: Pylint
 - Kotlin: Ktlint
 - Java: Checkstyle
 - Ruby: RuboCop

Benefits of Linters

- Ensure coding standards are met
- Catch potential bugs or vulnerabilities early
- Reduce the number of issues during code reviews, allowing reviewers to focus on bigger-picture concerns

Real-time Linting in Android Studio

- Android Studio automatically runs lint checks while you write code and highlights potential issues
- You can see the lint warning or error message by hovering over the highlighted code

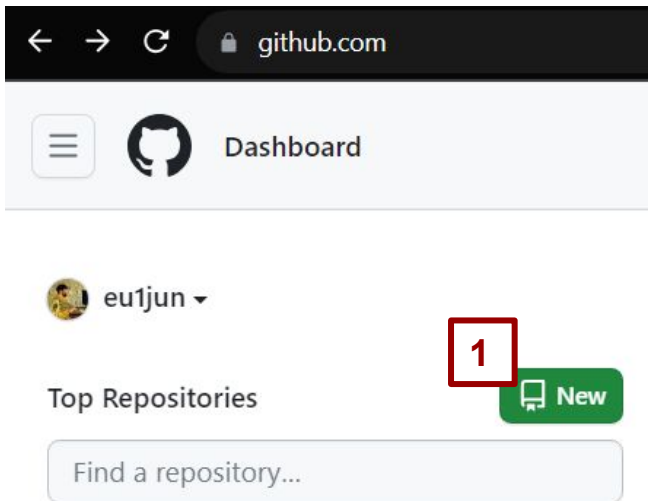
Manual Linting in Android Studio

- Navigate to Analyze > Inspect Code... from top menu
- Choose to inspect your entire project or a specific module
- Upon completion of inspection, a report is presented in the Inspection Results window, categorizing identified issues

Set Linting with GitHub Actions (1/4)

1. Make a new repository

a. Create one with README



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (*).

Repository template

No template ▾

Start your repository with a template repository's contents.

Owner *

eu1jun ▾

Repository name *

lint-exercise

✔ lint-exercise is available.

2

Initialize this repository with:

☒ Add a README file

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: None ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: None ▾

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set `main` as the default branch. Change the default name in your [settings](#).

 You are creating a private repository in your personal account.

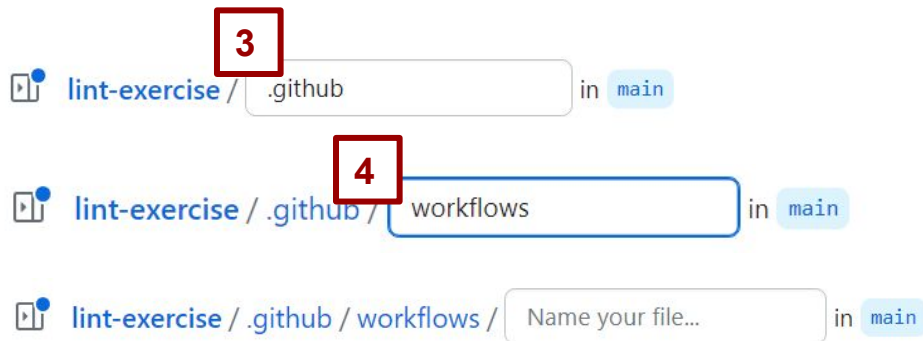
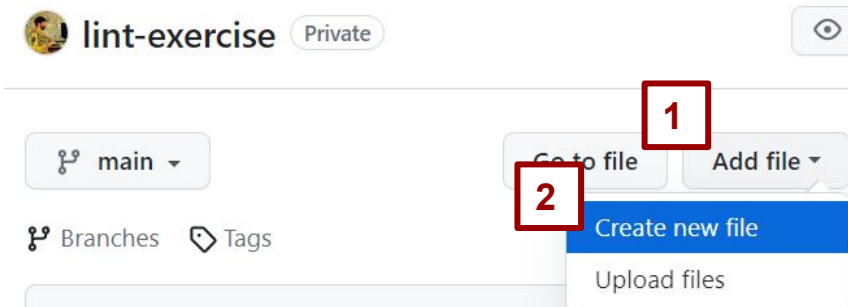
3

Create repository

Set Linting with GitHub Actions (2/4)

2. Create a `.github/workflows/` directory

- Type `.github/` in text box, don't forget `/`
- Type `workflows/` in text box, don't forget `/`



Set Linting with GitHub Actions (3/4)

3. Name a file android-lint.yml

- Paste the right workflow
- Click Commit Changes...



```
name: Android Lint + ktlint
on: pull_request
jobs:
  lint_and_ktlint:
    name: Run Android Lint and ktlint
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v4
      - name: Set up JDK 17
        uses: actions/setup-java@v4
        with:
          distribution: 'temurin'
          java-version: '17'
      - name: Grant execute permission for gradlew
        run: chmod +x ./gradlew
      - name: Android Lint (debug)
        run: ./gradlew lintDebug
      - name: ktlint check
        run: ./gradlew :app:ktlintMainSourceSetCheck
```

Set Linting with GitHub Actions (4/4)

4. Commit changes

a. Directly to the main branch

Commit changes

Commit message

Create android-lint.yml

Extended description

Add an optional extended description..

Commit Email

onsaga1@snu.ac.kr

☒ Commit directly to the main branch




☐ Create a new branch for this commit and start a pull request


Learn more about pull requests


Cancel



Commit changes

1

  main 

[lint-exercise](#) / [.github](#) / [workflows](#) / 

 **eu1jun** Create android-lint.yml

Name	Last commit message
 ..	
 <u>android-lint.yml</u>	Create android-lint.yml

Test Linting with GitHub Actions (1/4)

1. Download LintExercise zip from github [\[link\]](#)

a. It has wrong style

```
class MainActivity : ComponentActivity() {  
    // ✗ Lint: Class body should not start with a blank line  
    // (→ remove the empty line below this line)  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContent {  
  
            // ✗ Lint issues:  
            // - Missing newline after "("  
            // - Each argument should be on a separate line  
            // - Unexpected indentation (should be 16 spaces)  
            // - Missing newline before ")"  
            // - Missing trailing comma before ")"  
            // - Trailing spaces detected  
            Text(text = "Hello", // ← Arguments should not start on the same line as the function call  
                style = MaterialTheme.typography.bodyLarge, // ← Incorrect indentation  
                maxLines = 1) // ← Missing newline and trailing comma before ")"  
  
            ..  
        }  
    }  
}
```

Test Linting with GitHub Actions (2/4)

2. Extract and upload files

- Extract zipped folder
- Upload files (not LintExercise) by drag and drop into the lint-exercise repository root

1

LintExercise

Name

- app
- build.gradle.kts
- gradle
- gradle.properties
- gradlew
- gradlew.bat
- local.properties
- settings.gradle.kts

2

lint-exercise Private

main

Go to file Add file

Create new file

Upload files

3

Upload to root

eu1jun Create android-lint.yml

.github/workflows Create android-lint.yml

README.md Initial commit

4

Drop to upload your files

LintExercise

Name	Date Modified
app	Today at 11:51 AM
build.gradle.kts	Today at 11:50 AM
gradle	Today at 11:44 AM
gradle.properties	Today at 11:43 AM
gradlew	Today at 11:43 AM
gradlew.bat	Today at 11:43 AM
local.properties	Today at 11:43 AM
settings.gradle.kts	Today at 11:43 AM

Test Linting with GitHub Actions (3/4)

3. Create a branch and open PR


a. Create PR, too

/gradle/wrapper/gradle-wrapper.jar

x

/gradle/wrapper/gradle-wrapper.properties

x



Commit changes

Test linting for the sample app

Add an optional extended description...

☐ Commit directly to the `main` branch.

☒ Create a **new branch** for this commit and start a pull request. [Learn more about pull requests.](#)

test/lint

Propose changes

Cancel


Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you

base: main

compare: test/lint

✓ Able to merge. These branches can b



Test linting for the sample app

Write

Preview

H B I ≡ <> 🔗 ≡ ≡ ≡ @ 📎 ↶ 📧

Leave a comment

Attach files by dragging & dropping, se **3** pasting them. 📎

Create pull request

Test Linting with GitHub Actions (4/4)

4. Linting begins automatically

a. Click Details



Some checks haven't completed yet

1 in progress check



Android Lint + ktlint / Run Android Lint and ktlint (pull_request) Started 1 minute ago — This check h...



No conflicts with base branch

Merging can be performed automatically.

→ View details

1

Merge pull request

You can also merge this with the command line. [View command line instructions.](#)

```
/home/runner/work/lint-exercise/lint-exercise/app/src/main/java/com/example/lintexercise/MainActivity.kt:16:1 First line
in a method block should not be empty
/home/runner/work/lint-exercise/lint-exercise/app/src/main/java/com/example/lintexercise/MainActivity.kt:24:18 Missing
newline after "("
/home/runner/work/lint-exercise/lint-exercise/app/src/main/java/com/example/lintexercise/MainActivity.kt:24:18 Argument
should be on a separate line (unless all arguments can fit a single line)
/home/runner/work/lint-exercise/lint-exercise/app/src/main/java/com/example/lintexercise/MainActivity.kt:24:34 A comment
in a 'value_argument_list' is only allowed when placed on a separate line (cannot be auto-corrected)
/home/runner/work/lint-exercise/lint-exercise/app/src/main/java/com/example/lintexercise/MainActivity.kt:25:61 A comment
in a 'value_argument_list' is only allowed when placed on a separate line (cannot be auto-corrected)
/home/runner/work/lint-exercise/lint-exercise/app/src/main/java/com/example/lintexercise/MainActivity.kt:26:28 Missing
newline before ")"
/home/runner/work/lint-exercise/lint-exercise/app/src/main/java/com/example/lintexercise/MainActivity.kt:26:29 Missing
newline before ")"
/home/runner/work/lint-exercise/lint-exercise/app/src/main/java/com/example/lintexercise/MainActivity.kt:26:29 Missing
trailing comma before ")"
/home/runner/work/lint-exercise/lint-exercise/app/src/main/java/com/example/lintexercise/ui/theme/Color.kt:1:1 File must
```

Additional Resources

- "Software Engineering at Google" by Titus Winters, Tom Manshreck

Thank You.

Any Questions?