

# Version Control (GIT)

Week 5

*A little less conversation, a little more action.*

— Elvis Presley

# Objectives

- Understand how to work with Git
- Learn a typical workflow of using GitHub
- Specify how to apply GitHub flow into the term project

# Contents

- Recap of Git
- Typical workflow
- Merge conflict exercise
- GitHub flow in the term project

# Install Git

- Check whether Git is downloaded in your machine
  - `git --version`
- If you don't have it, install from the [official site](#)
- Now, you're ready!

# Recap: Basic Commands (1/4)

- `git init`
  - Start Git in the current working directory
- `git log`
  - Show the history of commits
- `git add <blobs>`
  - Include blobs into the staging area
- `git status`
  - Show modified and staged blobs
- `git commit`
  - Create a commit from the staging area

# Recap: Branch Commands (2/4)

- `git checkout <branch name or commit id>`
  - Move to certain branch or commit
  - Create branch if -b option given
- `git stash`
  - Temporarily save the modified changes which are not committed
- `git merge <branch name>`
  - Merge the changes of current branch and mentioned branches
- `git rebase <branch name>`
  - Delete commit and create new commit on the mentioned branch

# Recap: Undo Commands (3/4)

- `git reset <commit id>`
  - Delete all commits after the determined commit
  - `git reset --hard`
  - `git reset --mixed`
  - `git reset --soft`
- `git revert <commit id>`
  - Create a new commit with reverse patch of the determined one
  - No deletion

# Recap: Remote Commands (4/4)

- `git remote add <remote name> <url>`
  - Delete all commits after the determined commit
- `git clone <url> <directory name>`
  - Copy the entire history from remote repository
- `git push <remote name> <branch>`
  - Reflect changes on the branch to the remote repository
- `git fetch <remote name> <branch>`
  - Download changes from the remote repository
- `git pull <remote name> <branch>`
  - fetch + merge

# GitHub

- The largest host of Git
  - In Git's perception, GitHub is just host for remote repositories
  - Just as GitLab, Bitbucket, etc.
- Many useful tools
  - Actions
  - Issues
  - Copilots



**GitHub**

# Access GitHub with Your Machine

- Sign in to the GitHub
- Register your SSH key
  - Generate ssh key in your local machine
    - `ssh-keygen -t ed25519 -C "your_email@example.com"`
  - Copy cat `~/.ssh/id_ed25519.pub`
  - Click your profile → Settings → SSH and GPC keys
  - Press 'New SSH key' 
  - Paste your public key
- Check details [here](#)
- You can also use tokens

# How to Work with Git? (1/3)

- Write commit messages well
  - Use imperative form
  - Let others know changes without looking the source code
  - Explain “why”, “for what”, and “how”
  - We recommend the [Conventional Commits](#) specification

<type>[optional scope]: <description>

[optional body]

[optional footer(s)]

# How to Work with Git? (2/3)

- Conventional Commits examples

```
fix!: add missing parameter to service call
```

The error occurred because ...

```
BREAKING CHANGES: foo function requires bar argument
```

```
style: remove empty line
```

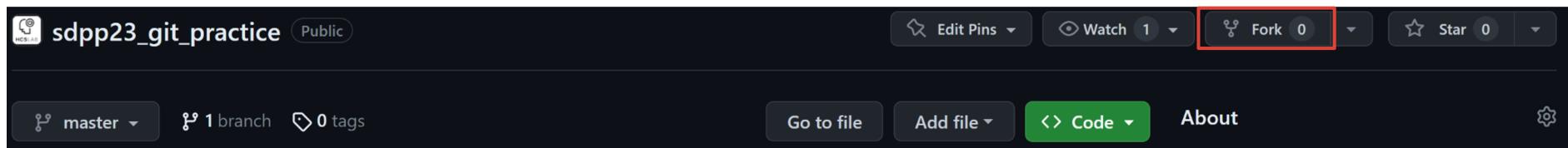
# How to Work with Git? (3/3)

- Keep commits and branches concise
  - Single commit is a “logically separate changeset”
  - Single branch must handle single feature
  - Merge after each feature is done (frequently!)
  - Double-check before merging into master branch

# Typical Workflow 1 (1/8)

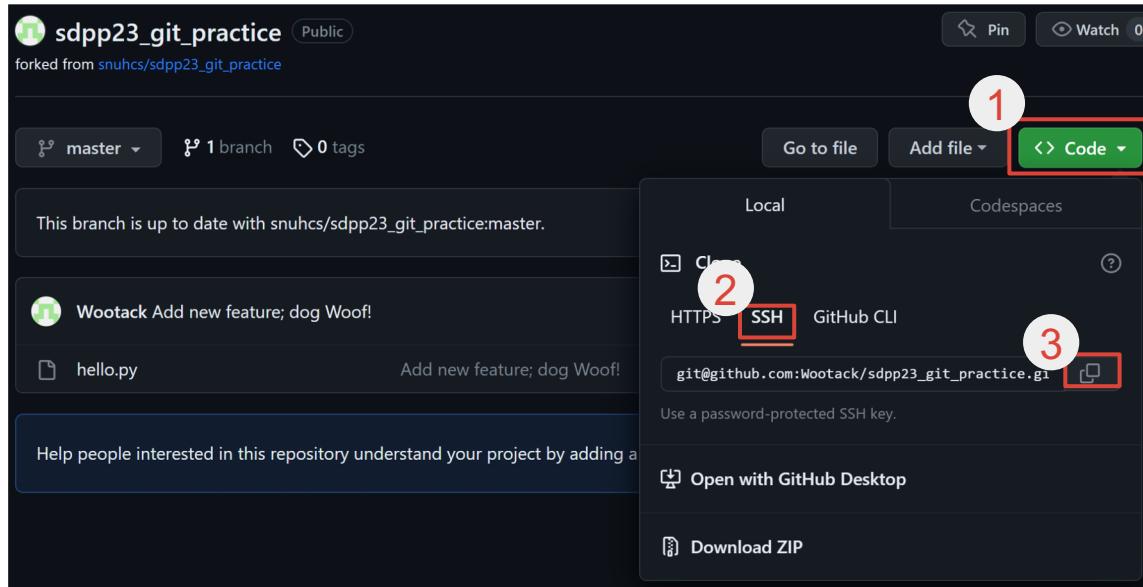
- You are not a contributor of a repository
  - You cannot push your edits

1. Fork the repository into your account



# Typical Workflow 1 (2/8)

## 2. Clone the repository into your local machine



```
$ git clone <Paste url>
```

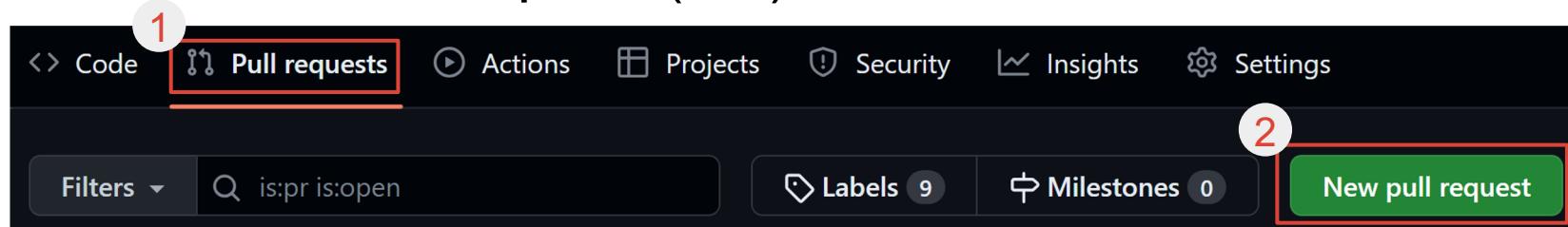
# Typical Workflow 1 (3/8)

3. Edit file
4. Add modified blobs into the staging area
5. Create commits
6. Push to the forked repository

```
wootackkim@casablanca:~/sdpp/04_git/sdpp23_git_practice$ vim hello.py
wootackkim@casablanca:~/sdpp/04_git/sdpp23_git_practice$ git add hello.py
wootackkim@casablanca:~/sdpp/04_git/sdpp23_git_practice$ git commit
[master f96b0aa] Add new feature; cow Moo!
  1 file changed, 2 insertions(+)
wootackkim@casablanca:~/sdpp/04_git/sdpp23_git_practice$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 96 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 304 bytes | 304.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:Wootack/sdpp23_git_practice.git
  aab81df..f96b0aa  master -> master
```

# Typical Workflow 1 (4/8)

## 7. Create a Pull Request (PR)



### Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#).

base repository: snuhcs/sdpp23\_git\_practice ▾ base: master ▾ ← head repository: Wootack/sdpp23\_git\_practice ▾ compare: master ▾

✓ Able to merge. These branches can be automatically merged.

Discuss and review the changes in this comparison with others. [Learn about pull requests](#)

Create pull request

-o 1 commit

1 file changed

1 contributor

# Typical Workflow 1 (5/8)

The screenshot shows a GitHub pull request creation interface with a dark theme. At the top, there's a placeholder text "Add new feature; cow Moo!" and tabs for "Write" and "Preview". Below the text area, there are several sections of template text:

- ## PR Type**  
New feature
- ## Motivation**  
Users want the cow Mooo feature according to ...
- ## Key Changes**  
'main` function could distinguish cow and `print` Mooo!
- ## To Reviewers**  
Among other sounds such as Moow, Mooo is selected due to ...

At the bottom, there's a note to "Attach files by dragging & dropping, selecting or pasting them." and a checkbox for "Allow edits by maintainers" with a help icon. A prominent green button on the right is labeled "Create pull request" with a dropdown arrow.

# Typical Workflow 1 (6/8)

## 8. Owners review your PR

The screenshot shows a GitHub interface with a dark theme. At the top, there is a navigation bar with links: Code, Issues, Pull requests (which has a red box around it and a red number '1' above it), Actions, Projects, Wiki, Security, Insights, and Settings.

In the center, there is a large, empty rectangular area with placeholder text: "Label issues and pull requests for n". Below this, a message says: "Now, GitHub will help potential first-time contributors discover".

At the bottom, there is a search bar with the query "is:pr is:open" and a "Filters" dropdown. To the right of the search bar, there is a search icon and a text input field.

The main content area displays a list of pull requests:

- 1 Open (checkbox) 0 Closed (checkbox) Author ▾
- 2 Add new feature; cow Moo! (checkbox) #1 opened 1 minute ago by Wootack • Review required

# Typical Workflow 1 (7/8)

Add new feature; cow Moo! #1

**Open** Wootack wants to merge 1 commit into `snuhcs:master` from `Wootack:master`

Conversation 0 Commits 1 Checks 0 Files changed 1 +2 -0

Changes from all commits File filter Conversations Jump to Review changes

Finish your review

**Write** Preview

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

**Comment**  
Submit general feedback without explicit approval.

**Approve**  
Submit feedback and approve merging these changes.

**Request changes**  
Submit feedback that must be addressed before merging.

Submit review

The screenshot shows a GitHub pull request interface for a merge from the branch 'Wootack:master' into 'snuhcs:master'. The title of the pull request is 'Add new feature; cow Moo! #1'. The top right corner has 'Edit' and 'Code' buttons. Below the title, there's a green 'Open' button and a message from 'Wootack' about wanting to merge. The main interface shows a conversation (0), one commit, no checks, and one file changed. A 'Review changes' button is highlighted with a red box. A modal window titled 'Finish your review' is open, with a 'Write' tab selected. It contains a 'Leave a comment' input field and an 'Attach files...' button. At the bottom of the modal, there are three radio buttons: 'Comment' (selected), 'Approve', and 'Request changes'. The 'Comment' option is described as submitting general feedback without explicit approval. The 'Approve' option is described as submitting feedback and approving merging these changes. The 'Request changes' option is described as submitting feedback that must be addressed before merging. A red box also highlights the 'Comment' radio button and the 'Submit review' button at the bottom of the modal.

# Typical Workflow 1 (8/8)

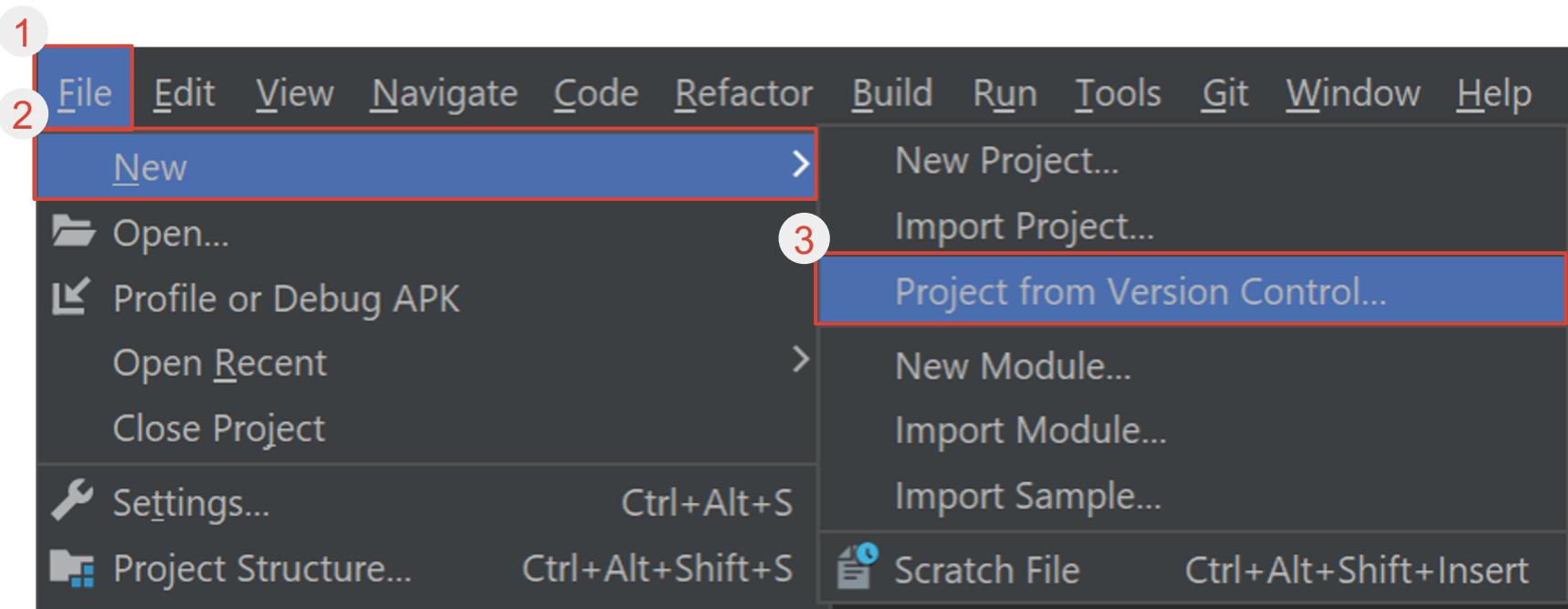
1. Fork the repository into your account
2. Clone the repository into your local machine
3. Edit file
4. Add modified blobs into the staging area
5. Create commits
6. Push to the forked repository
7. Create a Pull Request
8. Owners review your Pull Request

# Exercise

- Fork the [GitHub repository](#)
- Clone the forked repository
- Find a commit which maliciously adds typos
  - See commit message
- **Undo** that commit in proper way
- Push to the forked repository
- Send a **Pull Request!**
- Due: 25.10.12 23:59

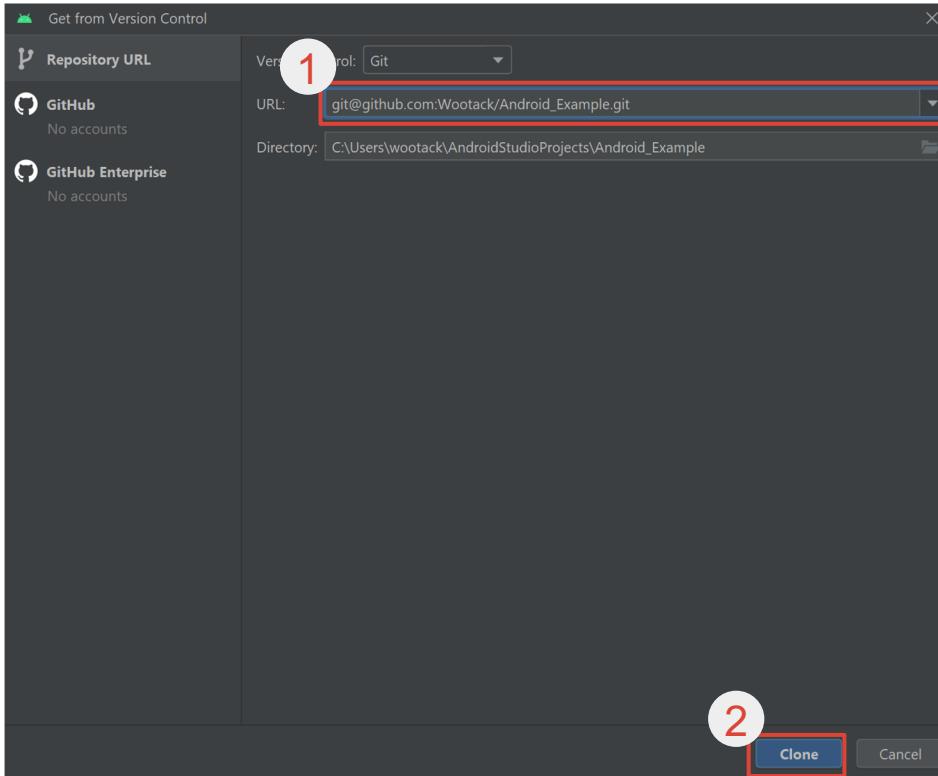
# Android Studio with Git/GitHub (1/2)

- “File” → “New” → “Project from Version Control...”



# Android Studio with Git/GitHub (2/2)

- Write URL of GitHub repository and Clone!



# Merge Conflicts

- If automatic merge is impossible, merge conflict occurs
  - User must **manually** solve it

```
<<<<< HEAD
```

```
print("Hi cat")
```

```
=====
```

```
print("Hi " + animalName)
```

```
>>>>> animal_name
```

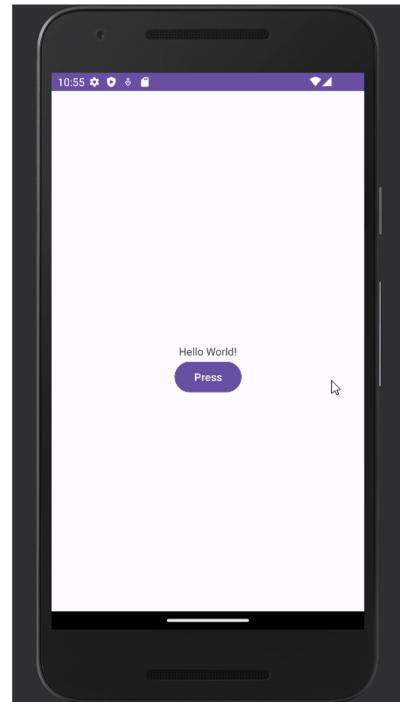
Contents from HEAD

Contents from incoming branch

Name of the incoming branch

# Group Exercise (1/4)

1. Join your team's repository by GitHub Classroom
2. Clone the repository



## Group Exercise (2/4)

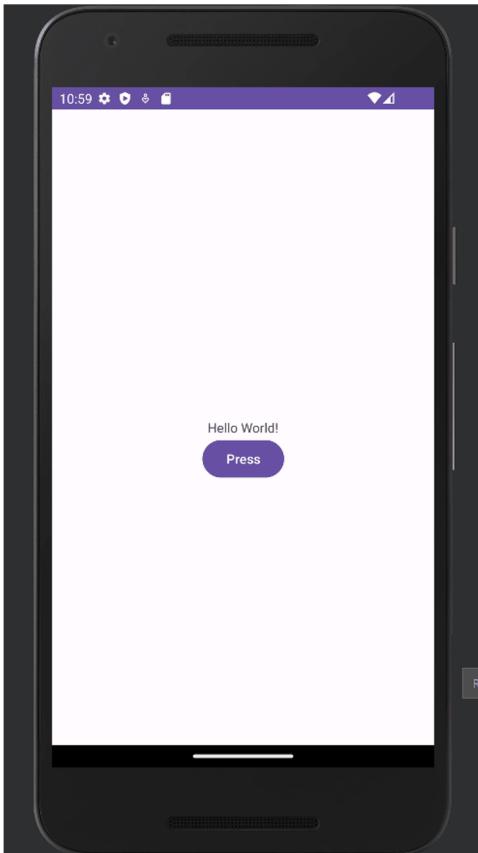
- Pair 1 add the *count* feature
  - Count the number of button press since the Activity create
  - Display in <content>\_<count> format
- Pair 2 add the *input content* feature
  - Add the EditText above the TextView
  - On the button click, read the input and display it

20 mins

# Group Exercise (3/4)

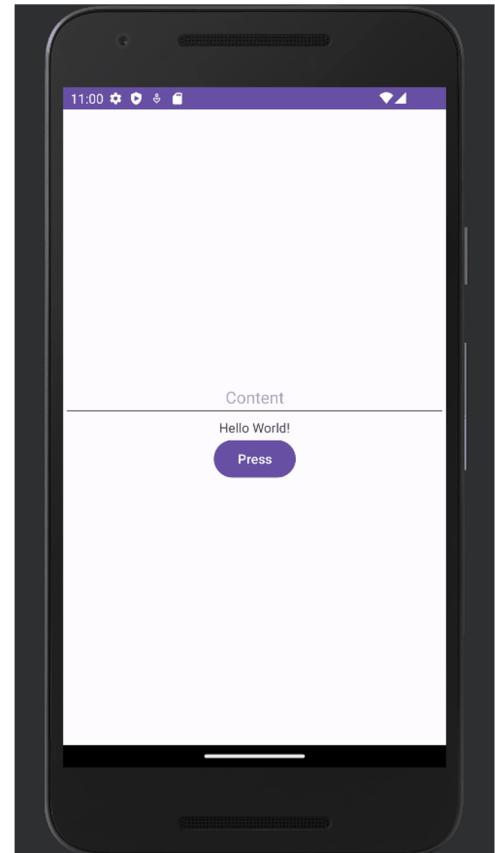
Pair 1

Count



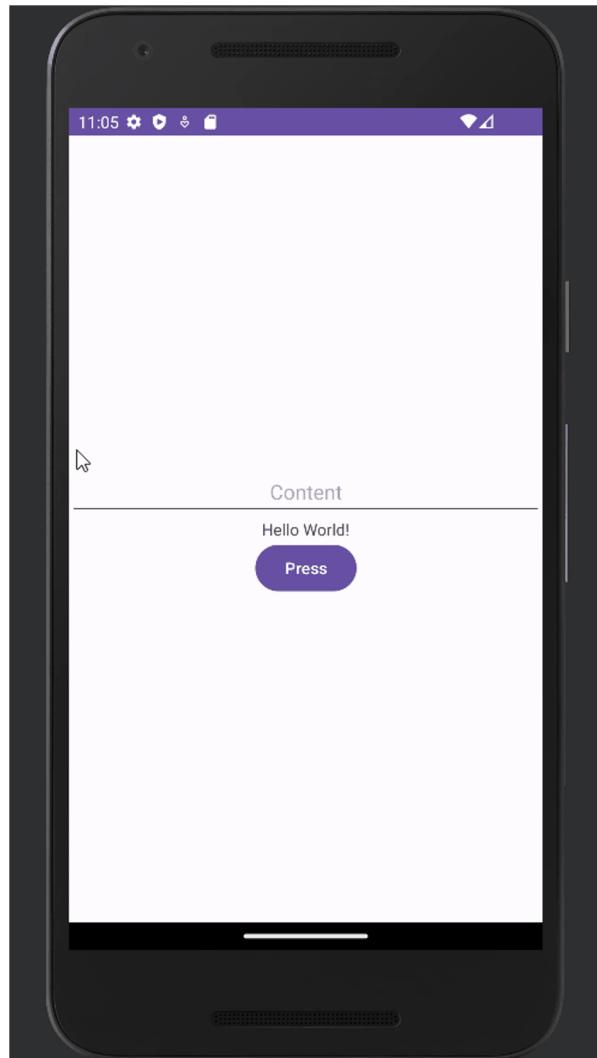
Pair 2

Content



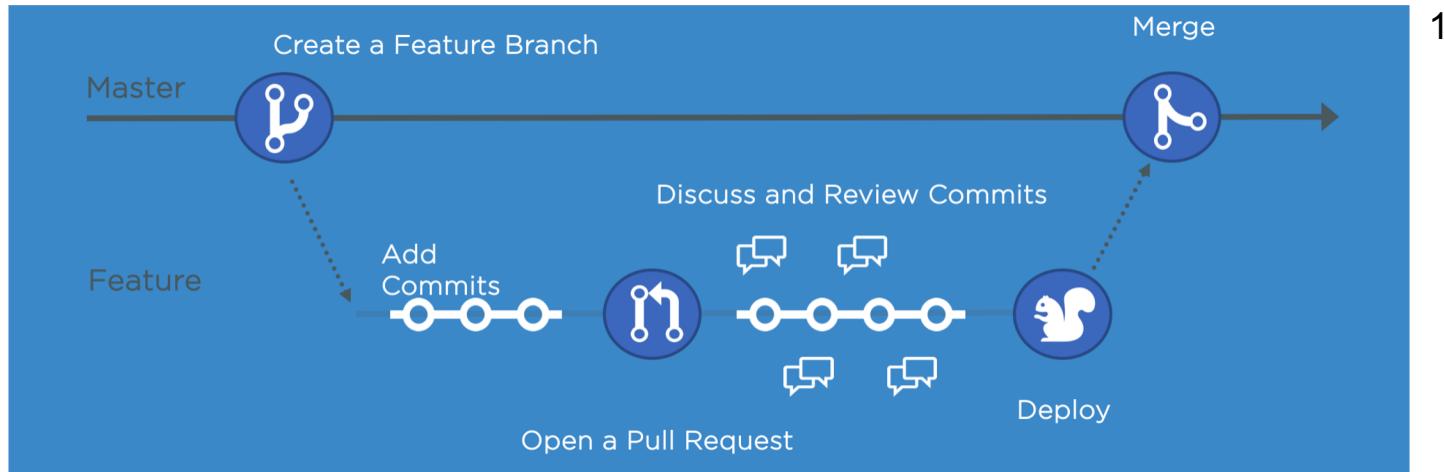
## Group Exercise (4/4)

4. Each pair send a pull request with proper content
    - What should you include?
  5. Process manager (PM) merge each pull request
    - Discuss with team members to solve the merge conflict
- Deadline: 25.10.12 23:59



# Git for the Term Project (1/7)

- You will use **GitHub flow**
- Only tested codes can be merged into the master
- Project manager (PM) must review PRs



1

# Git for the Term Project (2/7)

1. Each pair implements a feature
  - a. git switch master
  - b. git switch -c feat/<feature name>
  - c. git commit --author="NAME <EMAIL>"
    - i. E.g., "Wootack Kim <wootack.kim@hcs.snu.ac.kr>"
    - ii. Pair programming encourages frequent switch of driver
    - iii. The number of commits inside a pair should be balanced
  - Must include **unit test** from iteration 3

# Git for the Term Project (3/7)

```
commit 4f819ba8932c1203ec9a9363c6f8e14c2dcba8ef (HEAD -> feat/login)
Author: Wootack Kim <wootack.kim@hcs.snu.ac.kr>
Date:   Sun Sep 17 00:21:23 2023 +0900

  test: add unit test for login

commit c60aef64ea73575aabb504f5f5a8dc245204a432
Author: Hyunsoo Kim <hyunsoo.kim@hcs.snu.ac.kr>
Date:   Sun Sep 17 00:16:23 2023 +0900

  feat: connect frontend and backend

commit a70e37b6bb816a3ecacac3f6f3cd5f27d9e2d9e4
Author: Wootack Kim <wootack.kim@hcs.snu.ac.kr>
Date:   Sun Sep 17 00:15:10 2023 +0900

  feat: add backend operations for login

commit 8fc7d8aa6572da72dbe58c6e4aea12ea97a76239
Author: Hyunsoo Kim <hyunsoo.kim@hcs.snu.ac.kr>
Date:   Sun Sep 17 00:13:45 2023 +0900

  feat: add listeners for each ui component

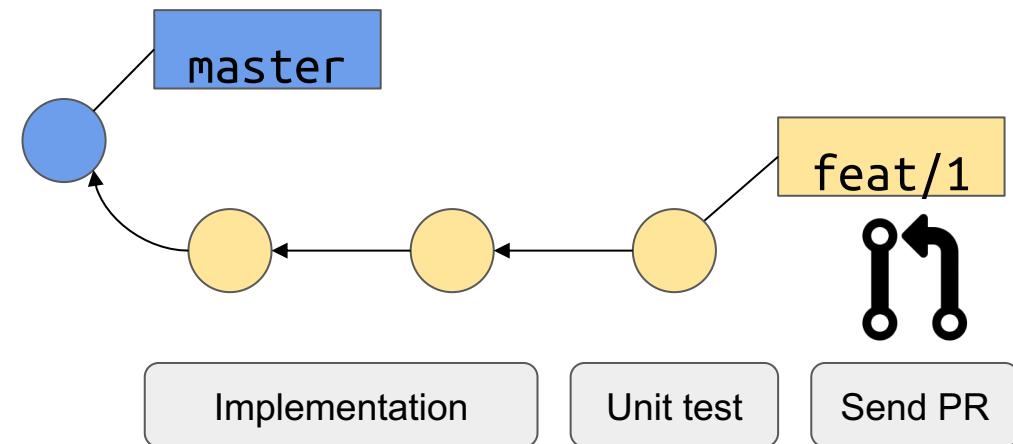
commit 318d4fdf5d7b6d458130b76e099a9b79d79c826f (master, develop)
Author: Wootack Kim <wootack.kim@hcs.snu.ac.kr>
Date:   Sun Sep 17 00:08:00 2023 +0900

  feat: add login UI
```

- Number of commits balanced
- Includes unit test

# Git for the Term Project (4/7)

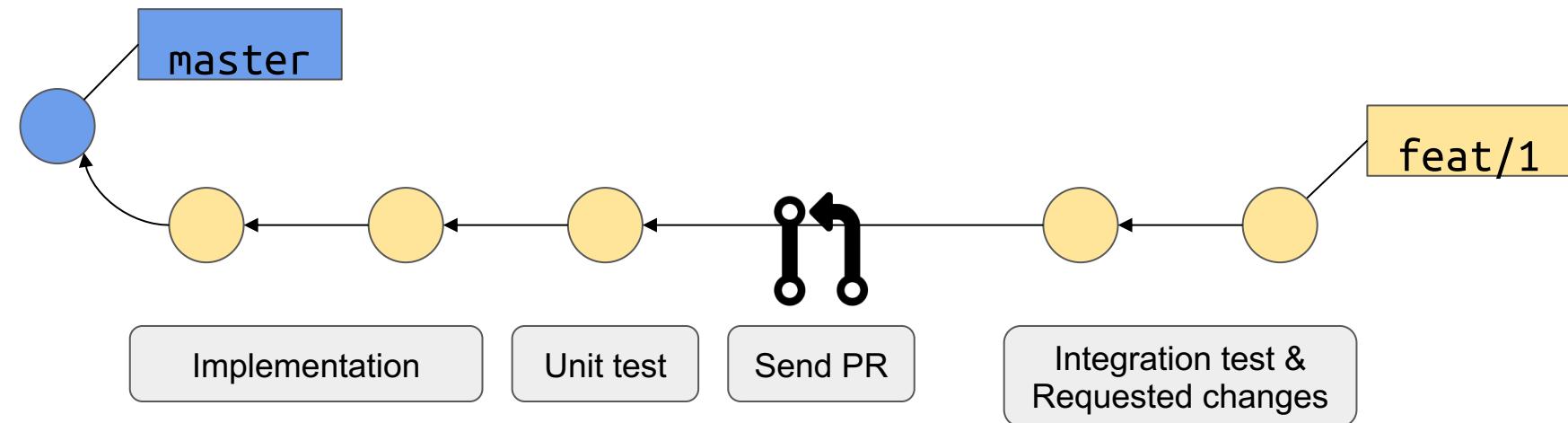
2. Each pair sends PR
  - a. PM reviews the PR
  - b. Requests changes to the pair



# Git for the Term Project (5/7)

## 3. Prepare merge

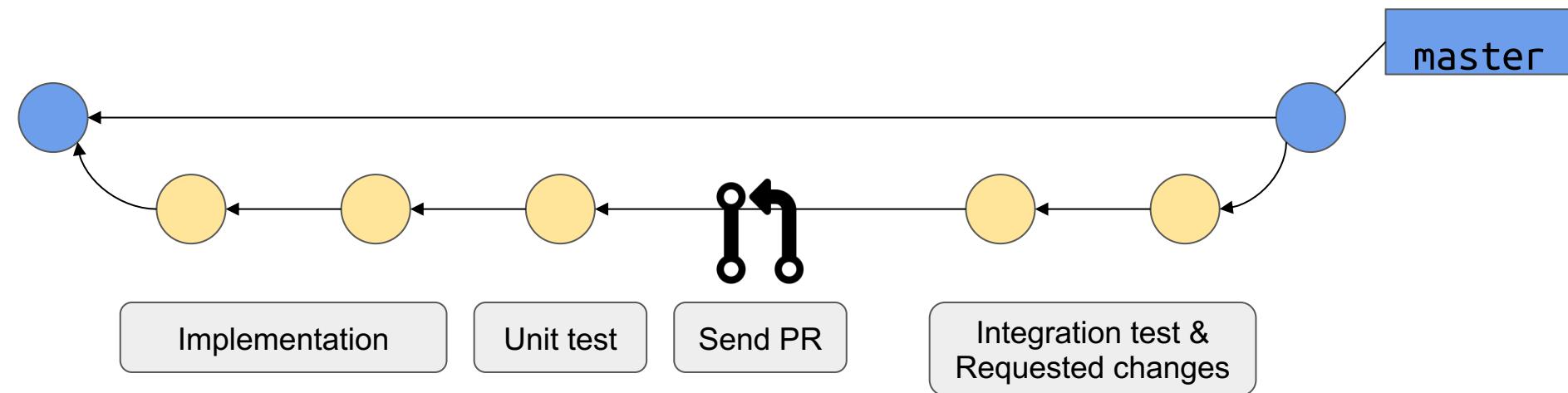
- a. Pair handles requested changes
- b. PM writes integration test code (from iteration 3)



# Git for the Term Project (6/7)

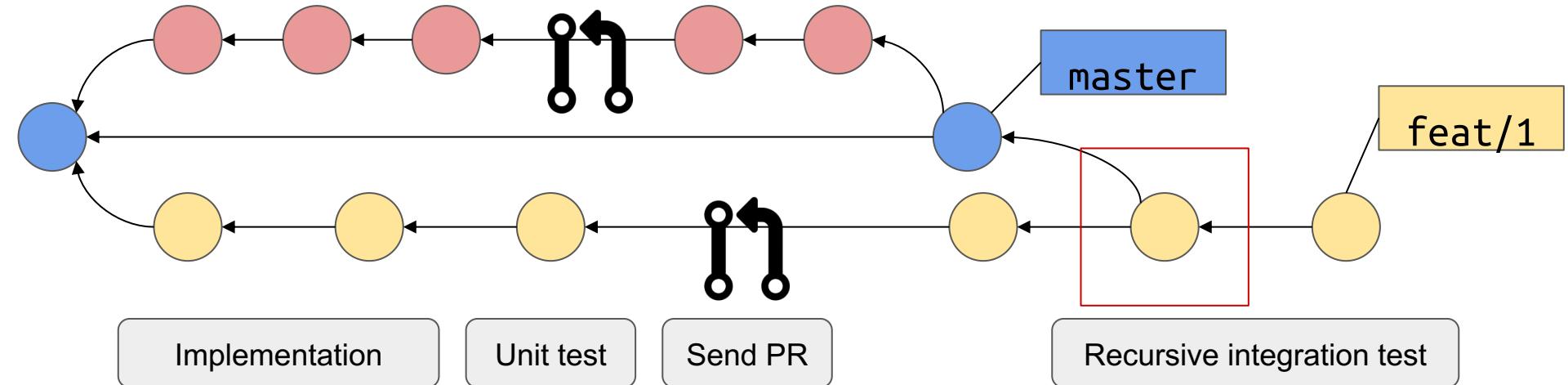
## 4. Integration test and merge

- a. If test fails, repeat step 3
- b. Remove merged branches



# Git for the Term Project (7/7)

- Recursive integration test
  - If the master is updated, integration test should reflect it



# Git for the Term Project - Tips (1/2)

- Please avoid using rebase
  - We will check commit metadata
- Delete merged branches
  - master is the one and only permanent branch
- Switch role while pair programming
  - Reflect it by setting the author of the commit
- Do not merge before passing the integration test
  - Do recursive integration test

# Git for the Term Project - Tips (2/2)

- Do not let Git to manage your private information
- Isolate private file and add it into the .gitignore
  - In Android Studio, [use properties file](#)

main.js

```
const connection = require('mysql').createConnection({  
  host: "Put your RDS endpoint here",  
  user: "RDS username",  
  database: "example",  
  password: "RDS user password",  
  port: 3306  
});
```

.gitignore

mysql\_connection.json

mysql\_connection.json

{

  "password":

  "1q2w~~"

}

# Summary

- Fork and send PR for projects which are not yours
- Resolving merge conflict
- GitHub flow in term project
  - Balanced commits
  - Unit test before PR
  - Recursive integration test before merge

# Supplementary Materials

- [MIT Missing Semester - Version Control \(Git\)](#)
- [Pro Git \(2nd\)](#)
- [Writing a proper GitHub issue](#)
- [More Practice on Using Git](#)

**Thank You.  
Any Questions?**