

Git & GitHub Support Guide

Everything you need for the weekly Pull Request routine

Read before Lab 1

Keep it open during every lab

Applies to Labs 1–9

1. Why Git, and why every week?

Git records **snapshots** of your project, so you can see what changed, undo mistakes, and work on a copy without breaking the working version. **GitHub** hosts your repository online. In this course you finish every lab by opening a **Pull Request (PR)**. That PR is what we mark. It also leaves you a portfolio of 9 small, reviewed steps that add up to a finished application.



Your laptop

GitHub

Branch names and PR titles are fixed for each lab (see the lab sheet). Never commit directly to main.

Commit message pattern: verb + what, e.g. "Add Book class with validated setters"

Tip Learn the seven steps above once. Every lab week is the same routine with a different branch name.

2. One-time setup (do this before Lab 1)

2.1 Install and check

```
git --version      # any recent version is fine
java -version      # should say 21 (or the LTS your lecturer announced)
```

2.2 Tell Git who you are

```
git config --global user.name "Your Name"
git config --global user.email "your-github-email@example.com"
git config --global init.defaultBranch main
```

2.3 Sign in to GitHub from your computer

GitHub no longer accepts your account password for `git push`. Pick **one** of these:

Option	Best for	How
GitHub Desktop / IDE login	Easiest	IntelliJ: <i>Settings</i> → <i>Version Control</i> → <i>GitHub</i> → <i>Add account</i> . VS Code: <i>Accounts menu</i> → <i>Sign in with GitHub</i> . The IDE stores the token for you.
Git Credential Manager	Windows/macOS (comes with Git)	The first <code>git push</code> opens a browser window. Sign in once.
Personal access token	Linux/terminal	GitHub → <i>Settings</i> → <i>Developer settings</i> → <i>Personal access tokens</i> → <i>Fine-grained</i> . Give it access to your one repository with <i>Contents: read/write</i> . Paste it when Git asks for a password.

Warning Never put a token or password in your code, in `README.md`, or in a commit message. If you do by accident, delete the token on GitHub immediately and make a new one.

3. Create your repository (once)

1. On GitHub choose **New repository**. Name it `lms-<your-index-number>`. Set it to **Private**. Tick *Add a README*.
2. *Settings* → *Collaborators* → add your lecturer and teaching assistant.
3. Clone it:

```
git clone https://github.com/<your-username>/lms-<index>.git
cd lms-<index>
```

4. Add a Java `.gitignore` (so compiled files never reach GitHub):

```
out/
build/
target/
*.class
lms.db
*.jar
.idea/
*.iml
.vscode/
```

5. Commit and push it straight to `main` (this is the **only** time you commit to `main` directly):

```
git add .gitignore README.md
git commit -m "Add .gitignore and project README"
git push origin main
```

6. On GitHub: *Settings* → *Rules* (or *Branches*) → protect `main` so changes can only arrive through a PR. If your plan does not offer this, simply promise yourself never to commit on `main`.

4. The weekly routine, command by command

Replace `<branch>` with the branch name printed on that week's lab sheet.

```
# 1. start from an up-to-date main
git checkout main
git pull origin main

# 2. make this week's branch
git checkout -b <branch>

# 3. work, then save small steps (repeat many times)
git status          # what changed?
git diff            # exactly which lines?
git add src/lms/model/Book.java # stage the files you mean to save
git commit -m "Add Book class with validated setters"

# 4. upload the branch
git push -u origin <branch> # -u only needed the first time

# 5-6. on GitHub: Compare & pull request -> write description -> request review
#       fix comments with more commits, then `git push` again

# 7. after the PR is merged
git checkout main
git pull origin main
```

Branch names and PR titles

Lab	Week	Branch	PR title
1	2	week2-encapsulation	Week 2: Classes, Objects & Encapsulation
2	3	week3-inheritance-polymorphism	Week 3: Inheritance & Polymorphism
3	4	week4-abstraction-interfaces	Week 4: Abstract Classes & Interfaces
4	5	week5-exceptions	Week 5: Exception Handling
5	8	week8-gui-layouts	Week 8: GUI I (Swing Layouts & Widgets)
6	9	week9-gui-events-mvc	Week 9: GUI II (Events, Dialogs & MVC)
7	10	week10-generics-streams	Week 10: Generics, Collections & Streams
8	11	week11-jdbc	Week 11: JDBC Database Connectivity
9	12	week12-threads-final	Week 12: Multithreading & Final Integration

5. What makes a good commit?

A commit is a **save point you could return to**. Make a commit each time one small thing works: a class compiles, a test passes, a bug is fixed.

Weak	Better
fixed bug	Fix crash when borrowing an already-borrowed book
updates	Add validation to Member.setName
work	Implement Book and Member with private fields
final version real	Add README with run instructions

Rules of thumb: 5 to 15 commits per lab · one idea per commit · message starts with a verb ("Add", "Fix", "Rename", "Remove") · never commit code that does not compile.

6. Writing the Pull Request

Use this template in the PR description box:

```
## What I built
- (two or three bullet points)

## How to run it
javac -d out $(find src -name "*.java") && java -cp out lms.Main

## What I learned / what was hard
(two sentences)

## Checklist
- [ ] Compiles with no warnings I understand
- [ ] I ran it and the output looks right
- [ ] UML diagram added in docs/ (uml-weekN.png)
- [ ] No .class files, no lms.db in the commit
```

Reviewing is part of the job. From Lab 2 onward you will also leave **two useful comments** on a classmate's PR (a question, a suggestion, or something you liked and why). Be kind and specific.

7. Merging

When the review is finished, press **Squash and merge**. Squashing turns your many small commits into one tidy commit on `main`, which keeps the history readable. Then delete the branch and update your computer:

```
git checkout main
git pull origin main
git branch -d <branch>
```

8. Using Git inside your IDE

You can do everything above with buttons. The commands are still worth knowing, because they work the same everywhere and error messages quote them.

Task	IntelliJ IDEA	VS Code
See changes	<i>Commit</i> panel (Alt+0)	<i>Source Control</i> panel (Ctrl+Shift+G)
New branch	<i>Git</i> menu → <i>New Branch</i>	Status bar → branch name → <i>Create new branch</i>
Commit	Tick files, write message, <i>Commit</i>	Type message, press ✓
Push	<i>Git</i> → <i>Push</i> (Ctrl+Shift+K)	<i>Sync Changes</i>

9. When things go wrong

"I forgot to make a branch and committed on `main`." Your work is not lost. Make the branch from where you are, then move `main` back:

```
git checkout -b week2-encapsulation # keeps your commits on the new branch
git checkout main
git reset --hard origin/main        # main matches GitHub again
git checkout week2-encapsulation
```

"**git push** says: rejected, non-fast-forward." Someone (or you, from another computer) changed that branch. Run `git pull --rebase origin <branch>` then push again.

"**I have a merge conflict.**" Git marks the clash inside the file:

```
<<<<<< HEAD
    private double price;
=====
    private int price;
>>>>>> week3-inheritance-polymorphism
```

Decide what the final line should be, delete the three marker lines, then:

```
git add <file>
git commit      # (or: git rebase --continue)
```

"**I committed something I should not have (a password, `lms.db`, a `.class` file).**" Remove it from Git but keep the file on disk, add it to `.gitignore`, and commit:

```
git rm --cached lms.db
echo "lms.db" >> .gitignore
git commit -am "Stop tracking lms.db"
```

(If it was a password or token, also **revoke it** on GitHub. Deleting the file does not delete history.)

"**I want to undo my last commit but keep my work.**"

```
git reset --soft HEAD~1
```

"**I changed a file and want it back the way it was at the last commit.**"

```
git restore src/lms/model/Book.java
```

"**Detached HEAD.**" You are looking at an old commit, not a branch. Keep any work with `git checkout -b rescue-branch`, or go back with `git checkout main`.

10. Cheat sheet

Command	Purpose
<code>git clone <url></code>	Download a repository
<code>git status</code>	What changed?
<code>git diff</code>	Show the exact changed lines
<code>git add <file></code>	Stage a file for the next commit
<code>git commit -m "msg"</code>	Save a snapshot
<code>git log --oneline --graph</code>	Compact history
<code>git checkout -b <name></code>	Create and switch to a branch
<code>git checkout <name></code>	Switch branch
<code>git push -u origin <branch></code>	Upload a new branch
<code>git pull origin main</code>	Download the latest <code>main</code>
<code>git restore <file></code>	Throw away uncommitted edits to a file
<code>git reset --soft HEAD~1</code>	Undo the last commit, keep the work
<code>git tag v1.0</code> and <code>git push origin v1.0</code>	Mark the final release (Week 12)

11. Marking: Git practice (10% of the course)

Criterion	4 (excellent)	3	2	1
Branch and PR naming	Always exact	Minor slips	Often wrong	Not used
Commit quality	Small, clear, verb-first	Mostly clear	Few, vague	One giant commit
PR description	Complete template	Mostly complete	Thin	Missing
Hygiene	Nothing unwanted committed	One slip	Several	<code>.class</code> /db/secrets committed
Reviewing	Two thoughtful comments each week	Comments, some shallow	Rare	None

12. Before Lab 1 checklist

- ☐ `java -version`, `javac -version` and `git --version` all work
- ☐ GitHub account created, signed in on your computer
- ☐ Private repository `lms-<index>` exists, lecturer and TA added
- ☐ `.gitignore` and `README.md` pushed to `main`
- ☐ You practised the weekly routine once on a throw-away branch `practice-branch` (create, commit, push, open a PR, close it without merging)