

Práctica 4.1 – GitHub Máster, Colaboración Controlada

En esta práctica vamos a aprender el flujo completo de trabajo con Git/GitHub: clonar, ramificar, hacer commits, manejar conflictos, crear Pull Requests y usar issues.

Se va a entregar un código que funciona pero está mal estructurado. La finalidad de la práctica es refactorizar el código realizando buenas prácticas de commits.

Esta práctica se va a realizar de forma individual.

Este documento debe recoger lo siguiente:

- **Índice**
- **Introducción:** Breve resumen de la factorización realizada
- **Configuración:** Uso y configuración de GitHub
- **Descripción de proceso:** Justificar tanto de forma argumentativa como de forma visual la realización de las distintas partes:
 - **Parte 1: Configuración inicial**
 - **Parte 2: Refactorización con Commits Atómicos**
 - **Parte 3: Colaboración y Conflictos**
 - **Parte 4: Code Review**
- **Conclusiones:** Realiza un pequeño análisis crítico de la práctica realizada.

Trabajos a realizar:

Fase 1: Configuración Inicial

1. **Fork** del repositorio del profesor
2. **Clonar** localmente
3. Crear **rama de desarrollo**: `git checkout -b feature/refactor-notifications`
4. Examinar el código y crear **issues** en GitHub para cada mejora identificada

Fase 2: Refactorización con Commits Atómicos

Cada cambio debe ser un **commit separado** con mensaje descriptivo:

Ejemplos de buenos mensajes:

```
- "feat: crear interfaz NotificationService"
- "refactor: extraer lógica de email a clase EmailService"
- "fix: corregir validación de número de teléfono"
- "test: añadir pruebas para SMSService"
- "docs: actualizar README con ejemplos de uso"
```

Commits obligatorios a demostrar:

1. Commit inicial (código base)
2. Crear interfaz NotificationService

3. Implementar EmailService
4. Implementar SMSService
5. Implementar PushService
6. Refactorizar NotificationManager para usar las nuevas clases
7. Añadir sistema de logs
8. Añadir validaciones
9. Crear fábrica de servicios (NotificationFactory)

Fase 3: Colaboración y Conflictos

1. **Simular conflicto:** El profeso subirá un cambio el 9 de marzo mientras se ha realizado ya algo de trabajo
2. **Resolver merge conflict** correctamente
3. Crear **Pull Request** con:
 - Título descriptivo
 - Descripción detallada
 - Issues relacionados
 - Reviewers asignados

Fase 4: Code Review

1. Revisar el PR de un compañero, indicando las revisiones que se realiza.
2. Dejar comentarios constructivos
3. Aprobar el PR
4. Hacer merge a main

Criterios de Evaluación

Habilidad	Puntos	Evidencia
Commits atómicos y descriptivos	3	Historial de commits claro
Uso correcto de ramas	2	Rama feature, no commit directo a main
Resolución de conflictos	2	Merge conflict resuelto apropiadamente
Pull Request bien documentado	2	Descripción clara, issues vinculados
Code Review útil	1	Comentarios constructivos a compañero

