Manual de los 15 Tipos de Conventional Commits versionado código en Spring Boot

1) **Conventional Commits** es una especificación para escribir *mensajes de commit estructurados y consistentes* en proyectos gestionados con Git. Esto mejora la legibilidad del historial, facilita la colaboración en equipo y permite automatizar tareas como la generación de changelogs o versiones. Estructura básica:

```
<tipo>(<ámbito>): <descripción corta>
[opcional cuerpo del mensaje]
[opcional pie de página]
```

- **Tipo**: Indica el propósito del commit (Ejem: `feat`, `fix`, `docs`).
- **Ámbito** (opcional): Especifica el módulo o componente afectado (e.g., `auth`, `api`, `config`).
- **Descripción corta**: Resumen breve (máximo 50 caracteres).
- **Cuerpo** (opcional): Detalles del cambio, explicando el **qué** y **por qué**.
- **Pie de página** (opcional): Referencias a issues o notas (e.g., `Closes #123`).

2) **Cómo funciona la estructura**

Cada commit sigue un formato estándar:

- **Primera línea**: `<tipo>(<ámbito>): <descripción corta>`. Ejemplo: `feat(tasks-api): add endpoint to create tasks` indica que añadimos un endpoint.
- **Cuerpo**: Explica detalles. Por ejemplo, en `fix(task-service)`, describimos cómo asociamos tareas al usuario autenticado.
- **Pie de página**: Conecta con issues (Ejem:, `Closes #15`) o agrega notas.

3) **Beneficios**

- Historial de Git claro y organizado, esencial para proyectos Spring Boot con múltiples capas.
- Facilita la colaboración en equipos, especialmente en proyectos con controladores, servicios y entidades.
- Automatiza tareas como generar changelogs o releases con herramientas como **Semantic Release**.
- Mejora la trazabilidad, útil para debugging y auditorías en aplicaciones empresariales.

4) **Los 15 Tipos Más Útiles** de Conventional Commits más relevantes para estudiantes trabajando en proyectos Spring Boot, basándome en su uso común en desarrollo backend:

- `feat`: Nueva funcionalidad para el usuario.
- `fix`: Corrección de errores.
- `docs`: Cambios en documentación.
- `style`: Cambios de formato o estilo (sin afectar la lógica).
- `refactor`: Reestructuración de código sin cambiar funcionalidad, reducir líneas de código, mejorar, etc.
- `test`: Agregar o modificar pruebas.
- `chore`: Tareas de mantenimiento (e.g., actualizar dependencias).
- `build`: Cambios en el sistema de compilación (e.g., Maven/Gradle).
- `ci`: Cambios en la configuración de integración continua.
- `perf`: Mejoras de rendimiento.
- `revert`: Revertir un commit previo.
- `config`: Cambios en archivos de configuración (e.g., `application.properties`).
- `security`: Cambios relacionados con seguridad (e.g., autenticación, autorización).
- `deps`: Gestión explícita de dependencias (similar a `chore`, pero enfocado en dependencias).
- `i18n`: Cambios relacionados con internacionalización o localización.

5) Ejemplo con un Código Fuente Real

Usaremos un proyecto Spring Boot: una **aplicación de gestión de tareas** (*Task Manager*) que permite crear, listar, actualizar y eliminar tareas, con una base de datos H2, Spring Security para autenticación y soporte para internacionalización. La estructura es:

Estructura del Proyecto

```
task-manager/
├── src/
│   ├── main/
│   │   ├── java/com/example/taskmanager/
│   │   │   ├── controller/
│   │   │   │   ├── TaskController.java
│   │   │   ├── service/
│   │   │   │   ├── TaskService.java
│   │   │   ├── entity/
│   │   │   │   ├── Task.java
│   │   │   ├── config/
│   │   │   │   ├── SecurityConfig.java
│   │   │   │   ├── MessageConfig.java
│   │   │   ├── TaskManagerApplication.java
│   │   ├── resources/
│   │   │   ├── application.properties
│   │   │   ├── messages.properties
│   │   │   ├── messages_es.properties
│   ├── test/
│   │   ├── java/com/example/taskmanager/
│   │   │   ├── controller/
│   │   │   │   ├── TaskControllerTest.java
├── .github/
│   ├── workflows/
│   │   ├── ci.yml
├── pom.xml
├── README.md
├── .gitignore
├── .checkstyle.xml  # Para reglas de estilo
```

6) Ejemplos de los 15 Tipos de Commits

1. **`feat`** ***(nueva funcionalidad)***
   Escenario: Añadimos un endpoint REST para crear tareas.
   Cambio en el código:
   Archivo: src/main/java/com/example/taskmanager/controller/TaskController.java

```java
@RestController
@RequestMapping("/api/tasks")
public class TaskController {
    private final TaskService taskService;

    public TaskController(TaskService taskService) {
        this.taskService = taskService;
    }
```

```
+ @PostMapping
+ public ResponseEntity<Task> createTask(@Valid @RequestBody Task task, Authentication auth) {
+     Task createdTask = taskService.createTask(task, auth.getName());
+     return new ResponseEntity<>(createdTask, HttpStatus.CREATED);
+ }
}
```

Mensaje de Commit:

```
feat(tasks-api): add endpoint to create tasks
Implement POST /api/tasks to create tasks for authenticated users.
Includes validation and returns 201 status.
Closes #10
```

Este commit commit añade un endpoint REST (`POST /api/tasks`) para que los usuarios creen tareas. Es un cambio visible para el usuario final (o cliente de la API). El ámbito tasks-api indica que el cambio está en la capa de la API, y el cuerpo explica que se incluye validación y que se usa la autenticación del usuario. El pie de página cierra un issue (#10).

2. **`fix`** ***(corrección de errores)***

Escenario: Corregimos un error en el servicio que no asociaba tareas al usuario autenticado.
Cambio en el código:
Archivo: src/main/java/com/example/taskmanager/service/TaskService.java

```java
@Service
public class TaskService {
    private final TaskRepository taskRepository;

    public TaskService(TaskRepository taskRepository) {
        this.taskRepository = taskRepository;
    }

-   public Task createTask(Task task) {
+   public Task createTask(Task task, String userId) {
+       task.setUserId(userId);
        return taskRepository.save(task);
    }
}
```

Mensaje de Commit:

```
fix(task-service): set userId in task creation
Ensure tasks are associated with the authenticated user.
Closes #15
```

Este commit corrige un error en el servicio donde no se validaba la creación de una tarea asociada a un usuario autorizado. El commit `fix` solucionan problemas funcionales. El ámbito task-service apunta al servicio y el cuerpo detalla que la tarea se ha asociado con el usuario autorizado. El pie de página cierra el issue (#15).

3. `docs` *(Documentación)*
Escenario: Actualizamos el README.md con instrucciones de instalación.
Cambio en el código:
Archivo: README.md

```markdown
# Task Manager

## Installation
1. Clone the repository: `git clone https://github.com/user/task-manager.git`
2. Install dependencies: `mvn install`
3. Configure database in `src/main/resources/application.properties`
4. Run the application: `mvn spring-boot:run`
```

Mensaje de Commit:

```
docs(readme): add installation and setup instructions
Provide steps to clone, install, and run the Spring Boot app.
Verifies the version of the document.
```

Este commit actualiza el `README.md` para incluir instrucciones claras de instalación. Los commits `docs` son para cambios en documentación que no afectan al código. El ámbito readme especifica el archivo, y el mensaje es breve porque el cambio es simple. Se verifica que la versión del documento es la correcta.

4. `style` *(Cambios de Estilo)*
Escenario: Mejoramos el formato del código en TaskController.java para seguir convenciones de estilo.
Cambio en el código:
Archivo: src/main/java/com/example/taskmanager/controller/TaskController.java

```java
-@RestController @RequestMapping("/api/tasks")
-public class TaskController{
-    private final TaskService taskService;public TaskController(TaskService taskService){this.taskService=taskService;}
+@RestController
+@RequestMapping("/api/tasks")
+public class TaskController {
+    private final TaskService taskService;
+
+    public TaskController(TaskService taskService) {
+        this.taskService = taskService;
+    }
```

Mensaje de Commit:

```
style(task-controller): format code to follow style guidelines
Add consistent spacing and line breaks per Java conventions.
Verifies the new style in the frontend.
```

Este commit mejora el formato del controlador para seguir las convenciones de estilo de Java (espacios, saltos de línea). Los commits `style` no cambian la lógica, solo la presentación del código. El ámbito task-controller indica el archivo afectado Se verifica el nuevo estilo se aprecia en el frontend.

5. `refactor` *(Refactorización)*
Escenario: Reorganizamos el servicio para separar la validación del título.
Cambio en el código:
Archivo: src/main/java/com/example/taskmanager/service/TaskService.java

```java
@Service
public class TaskService {
    private final TaskRepository taskRepository;

    public TaskService(TaskRepository taskRepository) {
        this.taskRepository = taskRepository;
    }
-   public Task createTask(Task task, String userId) {
-       if (task.getTitle() == null || task.getTitle().trim().isEmpty()) {
-           throw new IllegalArgumentException("Title is required");
-       }
-       task.setUserId(userId);
-       return taskRepository.save(task);
-   }
+   private void validateTask(Task task) {
+       if (task.getTitle() == null || task.getTitle().trim().isEmpty()) {
+           throw new IllegalArgumentException("Title is required");
+       }
+   }
+   public Task createTask(Task task, String userId) {
+       validateTask(task);
+       task.setUserId(userId);
+       return taskRepository.save(task);
+   }
}
```

Mensaje de Commit:

```
refactor(task-service): extract title validation logic
Move validation to a separate method for better readability.
Closes task #105
```

Este commit reorganiza el servicio para separar la validación en un método independiente, mejorando la legibilidad y mantenibilidad. Los commits `refactor` mejoran el código sin cambiar su comportamiento. El ámbito task-service apunta al servicio, y el cuerpo explica el propósito. El pie de página cierra el issue (#105).

6. `test` *(Pruebas)*
Escenario: Añadimos pruebas unitarias para el servicio de tareas.
Cambio en el código:
Archivo: src/test/java/com/example/taskmanager/service/TaskServiceTest.java

```java
@SpringBootTest
public class TaskServiceTest {
    @MockBean
    private TaskRepository taskRepository;
    @Autowired
    private TaskService taskService;

    @Test
    void createTask_withValidTask_savesTask() {
        Task task = new Task("Test task");
        when(taskRepository.save(any(Task.class))).thenReturn(task);

        Task result = taskService.createTask(task, "user1");

        assertEquals("Test task", result.getTitle());
        assertEquals("user1", result.getUserId());
        verify(taskRepository).save(task);
    }
}
```

Mensaje de Commit:

```
test(task-service): add unit tests for task creation
Test task creation with userId and repository interaction.
Verifies jUnit configuration.
```

Este commit añade pruebas unitarias para verificar que el servicio crea tareas correctamente, esto asegura que el código sea confiable y es clave en Spring Boot. Los commits `test` son para pruebas nuevas o modificadas. El ámbito task-service indica que las pruebas están en el servicio, y el cuerpo detalla qué se prueba. Se debe verificar que la herramienta jUnit está bien configurada.

7. `chore` *(Mantenimiento)*
Escenario: Actualizamos la versión de Spring Boot.
Cambio en el código:
Archivo: pom.xml

```xml
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
-   <version>2.7.0</version>
+   <version>3.1.0</version>
    <relativePath/>
</parent>
```

Mensaje de Commit:

```
chore(deps): upgrade Spring Boot to version 3.1.0
Update parent POM to use the latest Spring Boot version.
Verifies Spring Boot version.
```

Este commit actualiza Spring Boot a una nueva versión. Los commits `chore` manejan tareas de mantenimiento rutinarias; el ámbito deps indica que es una tarea de dependencias. Se verifica que se tiene la versión de Spring Boot deseada.

8. `build` ***(Sistema de Compilación)***
Escenario: Añadimos un plugin de Maven para optimizar el build.
Cambio en el código:
Archivo: pom.xml

```xml
<build>
  <plugins>
+   <plugin>
+     <groupId>org.apache.maven.plugins</groupId>
+     <artifactId>maven-surefire-plugin</artifactId>
+     <version>3.0.0-M7</version>
+     <configuration>
+       <skipTests>false</skipTests>
+     </configuration>
+   </plugin>
  </plugins>
</build>
```

Mensaje de Commit:

```
build(maven): add surefire plugin for test execution
Configure Maven to run tests during build process.
Closes task #85.
```

*Este commit añade el plugin **Surefire** de Maven para ejecutar pruebas durante el build. Los commits `build` afectan el sistema de compilación (Maven en este caso). El ámbito maven especifica la herramienta, y el cuerpo explica el propósito. El pie de página cierra el issue #85.*

9. `ci` ***(Integración Continua)***
Escenario: Configuramos GitHub Actions para pruebas automáticas.
Cambio en el código:
Archivo: .github/workflows/ci.yml

```yaml
name: CI
on: [push]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Set up JDK 17
```

```
    uses: actions/setup-java@v3
    with:
      java-version: '17'
      distribution: 'temurin'
  - name: Build with Maven
    run: mvn –batch-mode –update-snapshots verify
```

Mensaje de Commit:

```
ci(github): add GitHub Actions workflow for Maven build
Run Maven build and tests on push events with JDK 17.
Closes pipeline.
```

*Este commit configura **GitHub Actions** para ejecutar pruebas automáticamente en cada push. Los commits `ci` son para cambios en integración continua. El ámbito `github` indica la plataforma, y el cuerpo detalla la configuración. Se verifica que el pipeline se cierre correctamente.*

10. `perf` ***(Mejoras de Rendimiento)***

Escenario: Añadimos un índice en la entidad para optimizar consultas.
Cambio en el código:
Archivo: src/main/java/com/example/taskmanager/entity/Task.java

```java
@Entity
+@Table(indexes = {@Index(name = "idx_user_id", columnList = "userId")})
public class Task {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String title;
    private String userId;
    // Getters y setters
}
```

Mensaje de Commit:

```
perf(tasks-db): add index on userId for faster queries
Improve performance of task retrieval by user.
Closes task #75.
```

*Este commit Añadimos un índice en la entidad Task para optimizar consultas a la base de datos por **userId**. Los commits `perf` mejoran el rendimiento en general. El ámbito `tasks-db` apunta a la capa de base de datos. El pie de página cierra el issue #75.*

11. `revert` ***(Revertir un Commit)***

Escenario: Revertimos un cambio que eliminó la validación en el controlador.
Cambio en el código:
Archivo: src/main/java/com/example/taskmanager/controller/TaskController.java

```java
@PostMapping
-public ResponseEntity<Task> createTask(@RequestBody Task task, Authentication auth) {
-    Task createdTask = taskService.createTask(task, auth.getName());
```

```
-   return new ResponseEntity<>(createdTask, HttpStatus.CREATED);
+public ResponseEntity<Task> createTask(@Valid @RequestBody Task task, Authentication auth) {
+   Task createdTask = taskService.createTask(task, auth.getName());
+   return new ResponseEntity<>(createdTask, HttpStatus.CREATED);
}
```

Mensaje de Commit:

```
revert(tasks-api): restore validation in task creation
Revert commit that removed @Valid annotation in POST endpoint.
Restores input validation for task creation.
Closes task #200.
```

Este commit revierte un cambio que eliminó la validación @Valid en el controlador, lo que causaba errores. Los commits `revert` deshacen cambios previos. El ámbito `tasks-api` indica la capa afectada, y el cuerpo explica qué se restauró. El pie de página cierra el issue #200.

12. `config` ***(Configuración)***

Escenario: Añadimos configuración de logging en application.properties.
Cambio en el código:
Archivo: src/main/resources/application.properties

```properties
+logging.level.com.example.taskmanager=DEBUG
+logging.file.name=logs/task-manager.log
```

Mensaje de Commit:

```
config(logging): add debug logging configuration
Enable DEBUG level logging and output to a file.
Closes task #206.
```

Este commit añade configuración de logging en `application.properties` para depuración. Los commits `config` manejan ajustes en archivos de configuración. El ámbito `logging` especifica el tipo de configuración. El pie de página cierra el issue #206.

13. `security` ***(Seguridad)***

Escenario: Añadimos autenticación JWT en la configuración de seguridad.
Cambio en el código:
Archivo: src/main/java/com/example/taskmanager/config/SecurityConfig.java

```java
@Configuration
@EnableWebSecurity
public class SecurityConfig {
+  @Bean
+  public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
+     http
+        .authorizeHttpRequests(auth -> auth
+           .requestMatchers("/api/tasks/**").authenticated()
+           .anyRequest().permitAll())
+        .oauth2ResourceServer(oauth2 -> oauth2.jwt());
+     return http.build();
```

```
+ }
}
```

Mensaje de Commit:

```
security(auth): add JWT authentication for task endpoints
Configure Spring Security to use JWT for /api/tasks.
Ensures only authenticated users can access task APIs.
Closes task #300.
```

Este commit Implementamos autenticación JWT para proteger los endpoints de tareas. Los commits `security` son para mejoras de seguridad. El ámbito `auth` indica que es un cambio de autenticación, y el cuerpo detalla la configuración. El pie de página cierra el issue #300.

14. **`deps` (*Dependencias*)**
Escenario: Añadimos una dependencia para soporte de JWT.
Cambio en el código:
Archivo: pom.xml

```xml
<dependencies>
+  <dependency>
+     <groupId>org.springframework.boot</groupId>
+     <artifactId>spring-boot-starter-oauth2-resource-server</artifactId>
+  </dependency>
</dependencies>
```

Mensaje de Commit:

```
deps(auth): add spring-boot-starter-oauth2-resource-server
Include dependency for JWT authentication support.
Closes task #500.
```

Este commit añade una dependencia para soporte de JWT en el `pom.xml`. Los commits `deps` son específicos para gestionar dependencias. El ámbito `auth` indica el propósito de la dependencia. El pie de página cierra el issue #300.

15. **`i18n` (*Internacionalización*)**
Escenario: Añadimos soporte para mensajes en español.
Cambio en el código:
Archivo: src/main/resources/messages_es.properties

```properties
+task.created.success=Tarea creada exitosamente
+task.title.required=El título es obligatorio
```

Archivo: src/main/java/com/example/taskmanager/config/MessageConfig.java

```java
@Configuration
public class MessageConfig {
+  @Bean
+  public MessageSource messageSource() {
```

```
+     ReloadableResourceBundleMessageSource messageSource = new ReloadableResourceBundleMessageSource();
+     messageSource.setBasename("classpath:messages");
+     messageSource.setDefaultEncoding("UTF-8");
+     return messageSource;
+ }
}
```

Mensaje de Commit:

```
i18n(messages): add Spanish translations for task messages
Include messages_es.properties and configure MessageSource.
Closes #20.
```

Este commit añade soporte para mensajes en español y configuramos el `MessageSource`. Los commits `i18n` son para internacionalización. El ámbito `messages` apunta a los archivos de traducción, y el pie cierra el issue #20.

7) Buenas Prácticas para Conventional Commits

- **Usa tipos específicos**: Cada tipo (`feat`, `fix`, etc.) tiene un propósito claro. Usa `security` para cambios de autenticación y `i18n` para internacionalización.
- **Define el ámbito**: Usa nombres como `tasks-api`, `task-service`, o `auth` para reflejar el contexto en Spring Boot.
- **Sé breve**: La primera línea debe ser menor a 50 caracteres.
- **Explica en el cuerpo**: Detalla el **qué** y **por qué**, no el **cómo**.
- **Referencia issues**: Usa `Closes #123` para conectar con GitHub Issues.
- **Usa herramientas**:
- **Commitizen**: Guía para escribir commits (`npm install -g commitizen` o usa `git-cz`).
  - **Husky**: Valida commits antes de ejecutarlos.
  - **Maven Release Plugin** o **Semantic Release**: Automatiza versiones.
- **Revisa el historial**: Usa `git log --oneline` para verificar la claridad.

8) ¿Por qué usar Conventional Commits en Spring Boot?

Como estudiante, estás aprendiendo a construir aplicaciones backend con Spring Boot, un framework muy usado en la industria para APIs REST. Los **Conventional Commits** te ayudan a:

- Crear un historial de Git claro, esencial en proyectos Spring Boot con capas como controladores, servicios y entidades.
- Colaborar en equipo, ya que los mensajes estructurados permiten entender los cambios rápidamente.
- Prepararte para herramientas de automatización, como generar changelogs o releases, comunes en proyectos profesionales.
- Practicar hábitos que usan proyectos open-source como Spring Boot o Spring Security.

9) Consejos para estudiantes

- **Empieza con proyectos simples**: Aplica Conventional Commits en tus proyectos de Spring Boot, como una API de tareas o usuarios. Usa `feat` para endpoints, `test` para pruebas, etc.
- **Usa herramientas**:
  - Instala **Commitizen** (`npm install -g commitizen`) o usa `git-cz` para escribir commits.
  - Configura **Husky** para validar commits automáticamente.
  - Usa el **Maven Release Plugin** para automatizar versiones.
- **Revisa el historial**: Usa `git log --oneline` para ver cómo los commits estructurados hacen el historial más claro.

- **Colabora con reglas**: En proyectos grupales, acuerda con tu equipo usar estos 15 tipos para mantener consistencia.
- **Explora proyectos reales**: Mira repositorios como `spring-projects/spring-boot` en GitHub para aprender cómo usan commits estructurados.

10) Cómo implementar Conventional Commits en Spring Boot

- **Configura tu proyecto**:
  - Crea un proyecto con Spring Initializr (https://start.spring.io/) o clona el ejemplo del manual.
  - Asegúrate de tener Git, Java (JDK 17), y Maven instalados.
- **Haz cambios específicos**:
  - Por ejemplo, añade un endpoint, corrige un error, o configura internacionalización.
- **Escribe el commit**:
  - Usa la estructura: `<tipo>(<ámbito>): <descripción>`.
  - Ejemplo: `i18n(messages): add French translations`.
- **Automatiza**:
  - Usa **Commitizen** o **commitlint** para validar el formato.
  - Configura **Husky** para verificar commits.
  - Explora **Semantic Release** o **Maven Release Plugin** para generar versiones.
- **Prueba en equipo**:
  - Comparte el proyecto con compañeros y revisen los commits juntos.

11) Recursos adicionales

- **Conventional Commits**: https://www.conventionalcommits.org/
- **Commitizen**: https://commitizen.github.io/cz-cli/
- **Husky**: https://typicode.github.io/husky/
- **Maven Release Plugin**: https://maven.apache.org/maven-release/maven-release-plugin/
- **Ejemplos reales**: Revisa repositorios como `spring-projects/spring-boot` o `spring-projects/spring-security`.