**sEstrategia de Control de Versiones**

A continuación se presenta el documento que describe la estrategia de control de versiones aplicada al proyecto, en el marco de la guía ISO/IEC 29110. Este texto puede incorporarse como parte del plan de proyecto, sin hacer referencia explícita punto por punto al estándar, pero garantizando el cumplimiento de sus requerimientos.

## **1. Objetivo**

Este documento tiene como finalidad describir el enfoque para la gestión y el control de versiones de los entregables, el código fuente y las pruebas asociadas al proyecto. Se definen las convenciones para el nombrado de archivos de entregables, la estructura de directorios, el flujo de trabajo en GitHub y las políticas de aprobación de cambios.

## **2. Estructura de Directorios**

Para facilitar la organización, la trazabilidad y el acceso a los distintos artefactos, la estructura de directorios del repositorio se define de la siguiente manera:

/entregables/

[NombreDelProducto]\_v[Version].pdf

[NombreDelProducto]\_v[Version].docx

...

/src/

(Código fuente de la aplicación)

/tests/

(Conjunto de pruebas automatizadas)

/docs/

(Documentación adicional que no forma parte de los entregables versionados)

/README.md

.gitignore

1. **Carpeta entregables/**
   * Aquí se almacenan todos los productos de salida de cada tarea (diagramas, planes, manuales, especificaciones, etc.).
   * Cada archivo llevará un sufijo en su nombre que indique claramente la versión, por ejemplo:  
     + Requerimientos\_v1.0.pdf
     + DiseñoArquitectura\_v1.2.docx
   * De esta manera, auditoría y revisiones posteriores pueden identificar rápidamente cuál es la versión entregada en cada hito.
2. **Carpeta src/**
   * Contiene exclusivamente el código fuente de la solución.
   * Se garantiza que ningún binario ni entregable final se guarde aquí, manteniendo este espacio limpio y dedicado únicamente a código en desarrollo.
3. **Carpeta tests/**
   * Aloja todos los tests automatizados (unitarios, de integración, etc.).
   * La existencia de esta carpeta separada refuerza la trazabilidad entre el código fuente y sus pruebas correspondientes, facilitando la validación continua de calidad.
4. **Carpeta docs/ (opcional)**
   * Reúne otra documentación de apoyo (instrucciones de despliegue, diagramas detallados, plantillas internas, etc.) que no necesariamente está sujeta a versionado de entregables, pero sí requiere un control básico de cambios.

## **3. Convenciones de Versionado de Entregables**

Para cumplir el requisito de que cada producto de salida esté claramente versionado:

1. **Formato del nombre de archivo**
   * [NombreProducto]\_[v][númeroMayor].[númeroMenor][.númeroPatch]
   * Ejemplos:  
     + PlanProyecto\_v1.0.pdf
     + CasosPrueba\_v2.1.3.xlsx
2. **Incremento de versiones**
   * Se seguirá un esquema semántico simplificado:  
     + **vX.0**: Primera versión entregable para revisión.
     + **vX.Y**: Ediciones menores que incluyen correcciones o ajustes de contenido.
     + **vX.Y.Z** (opcional): Versiones de parche o aclaraciones puntuales.
3. **Registro de cambios**
   * Cada vez que se actualice la versión de un entregable, se incluirá un breve apunte en el documento “Registro de entregables” (archivo dentro de /docs/) donde se anote:  
     + Nombre del archivo y versión nueva.
     + Fecha de creación de la versión.
     + Resumen de las modificaciones realizadas desde la versión previa.
   * Esto facilita la labor del auditor para verificar qué se entregó en cada hito y con qué cambios.

## **4. Control de Código Fuente y Pull Requests**

Para asegurar la integridad del repositorio de código y mantener un flujo de trabajo colaborativo que cumpla con estándares de calidad:

1. **Repositorio en GitHub**
   * El proyecto estará alojado en un repositorio privado en GitHub.
   * No se permite el uso de force push sobre la rama principal (master o main).
   * Todas las ramas de desarrollo deberán derivar de la rama principal y, posteriormente, fusionarse mediante Pull Request (PR).
2. **Flujo de trabajo con Pull Requests**
   * **Creación de ramas de trabajo**
     + Cada funcionalidad, corrección de bug o cambio significativo se trabajará en una rama específica nombrada según la convención:  
       1. feature/[descripción-corta]
       2. bugfix/[descripción-corta]
       3. hotfix/[descripción-corta]
   * **El Pull Request (PR)**
     + Todo cambio debe realizarse a través de un PR que será revisado por, al menos, otro miembro del equipo técnico.
     + El título del PR deberá indicar brevemente la tarea o ticket asociado (por ejemplo: Implementa autenticación JWT (TICKET-123)).
     + En la descripción del PR se incluirá:  
       1. Objetivo del cambio.
       2. Breve resumen de la implementación.
       3. Referencia al número de issue o requerimiento correspondiente.
   * **Revisiones automatizadas**
     + Cada PR dispara un conjunto de chequeos automáticos que incluyen:  
       1. Ejecución de pruebas unitarias y de integración ubicadas en /tests/.
       2. Validación de estándares de estilo y linters (por ejemplo, ESLint, Pylint).
       3. Verificación de que no se introduzcan errores de compilación o fallos sintácticos.
   * **Aprobación y fusión**
     + Una vez aprobados los chequeos automáticos y la revisión manual, el revisor aprueba el PR.
     + Se procede a hacer merge sobre la rama de integración (p. ej., develop).
     + En caso de surgir conflictos, el autor deberá resolverlos localmente antes de solicitar nueva revisión.
3. **Ramas de liberación y producción**
   * **Rama develop**
     + Integra todas las características completadas y probadas en entornos de desarrollo.
     + Se puede desplegar en entornos de prueba/preproducción.
   * **Rama production**
     + Solo recibe cambios que ya estén verificados en develop y que hayan pasado las pruebas en entornos de preproducción.
     + Antes de hacer merge a production, se ejecutan pruebas finales (smoke tests) para garantizar estabilidad.
     + Una vez aprobado el despliegue en production, se etiqueta el commit con el número de versión del software (por ejemplo: v1.0.0).

## **5. Gestión de Pruebas**

Para lograr la trazabilidad entre código, pruebas y entregables finales:

1. **Ejecución Continua de Tests**
   * Cada vez que se genera un PR, la integración continua ejecuta automáticamente todos los tests ubicados en /tests/.
   * Solo si los tests pasan correctamente, se permite la fusión del PR.
2. **Versionado de Pruebas**
   * Al igual que el código, los tests se versionan bajo Git.
   * Cuando un entregable (por ejemplo, un manual de usuario o un documento de diseño) requiere validación, se genera un nuevo test o se actualiza uno existente para reflejar los criterios definidos en dicho entregable.
3. **Informes de Cobertura y Resultados**
   * Cada ejecución de tests produce un reporte de cobertura (Coverage Report), el cual se almacena en un servidor de integración continua (por ejemplo, GitHub Actions o Jenkins).
   * Estos informes pueden consultarse para verificar qué parte del código está cubierta y dejar registro de la calidad de las pruebas.

## **6. Acceso y Mecanismos de Recuperación**

Para asegurar la disponibilidad y la integridad histórica de los artefactos:

1. **Control de Accesos**
   * El repositorio GitHub contará con permisos restringidos:  
     + **Administradores**: Pueden crear ramas protegidas, configurar reglas de fusión y administrar integrantes del proyecto.
     + **Colaboradores**: Pueden crear ramas y PR, pero no modificar la rama principal sin aprobación ni realizar force push.
   * El directorio de entregables (/entregables/) se sincroniza automáticamente con el repositorio. Cada commit que incluya cambios en esta carpeta mantiene el historial de versiones.
2. **Protección de Ramas**
   * La rama principal (master o main) y la rama de producción (production) estarán configuradas como ramas protegidas en GitHub.
   * Para poder fusionar en estas ramas es obligatorio:  
     + Tener al menos una aprobación de revisión de código.
     + Haber pasado todos los chequeos automáticos (pruebas unitarias, linting, etc.).
3. **Ciclo de Entrega**
   * Al cierre de cada hito o iteración, se etiquetan los commits relevantes (tanto de código como de entregables) con un identificador claro (por ejemplo, Milestone-1-v1.0).
   * De esta forma, auditoría o interesados pueden retroceder a cualquier punto del historial y acceder a los artefactos entregados en ese momento.
4. **Almacenamiento de Copias de Seguridad**
   * Además del repositorio GitHub, se mantiene un respaldo periódicamente (backup) en un servidor interno o servicio en la nube con retención mínima de 90 días.
   * Esto garantiza la recuperación de cualquier versión anterior, incluso en caso de fallos en la plataforma principal.

## **7. Roles y Responsabilidades**

Para garantizar que la estrategia se aplique de forma consistente, se asignan los siguientes roles:

1. **Propietario del Repositorio (Repository Owner)**
   * Se encarga de mantener las políticas de protección de ramas en GitHub.
   * Administra los permisos de acceso y la configuración de integración continua.
2. **Desarrollador / Equipo Técnico**
   * Crea nuevas ramas para cada cambio y abre Pull Requests.
   * Se responsabiliza de nombrar correctamente los archivos de entregables y actualizar el “Registro de entregables” en /docs/ cada vez que exista una nueva versión.
   * Ejecuta pruebas locales antes de enviar cambios para revisión.
3. **Revisor de Pull Request (Reviewer)**
   * Verifica que los cambios propuestos cumplan con los estándares de calidad, el estilo de código y la funcionalidad solicitada.
   * Comprueba que los tests asociados se ejecuten correctamente y los criterios de aceptación estén cubiertos.
   * Aprueba o solicita correcciones al autor del PR.
4. **Auditor / Lead de Proyecto**
   * Revisa periódicamente el contenido de /entregables/ y /docs/RegistroEntregables.md para asegurar que las versiones se correspondan con los hitos planificados.
   * Verifica el cumplimiento de los procesos de control de versiones establecidos y solicita ajustes si se detecta alguna desviación.

## **8. Consideraciones Adicionales**

1. **Evitar Archivos Binarios en Código**
   * Solo los entregables finales se almacenan en /entregables/. No se incluyen binarios ni artefactos compilados dentro de /src/.
   * Esto evita el sobrepeso del repositorio y facilita las operaciones de clonación y revisión de historial.
2. **Documentación de Estrategias y Procesos**
   * Cualquier ajuste futuro en la estrategia (nueva convención de versión, cambio en la política de branches, etc.) se documentará en /docs/ProcesoControlVersiones.md, detallando motivo, alcance y fecha de aplicación.
3. **Comunicación y Formación**
   * Antes de iniciar el proyecto, se impartirá una breve capacitación al equipo sobre:  
     + Convenciones de nombre y versionado de entregables.
     + Uso adecuado de ramas, Pull Requests y protección de master/production.
     + Procedimiento para mantener actualizado el “Registro de entregables”.
   * De esta forma, todos los miembros conocen el flujo y reducen errores al aplicar la estrategia.

## **10. Ubicación en GitHub y Seguridad de Recuperación**

Para garantizar la disponibilidad, la integridad y la recuperación de todos los artefactos del proyecto, se establecen las siguientes medidas relacionadas con el uso de GitHub y los mecanismos de respaldo:

1. **Repositorio Privado en GitHub**
   * El proyecto estará alojado en un repositorio privado dentro de la organización correspondiente en GitHub.
   * Solo tendrán acceso personas autorizadas (roles definidos en la sección 7), evitando exposiciones involuntarias de código o entregables.
   * GitHub ofrece alta disponibilidad y replicación interna, reduciendo el riesgo de pérdida por fallos en la plataforma principal.
2. **Protección de Ramas Críticas**
   * Las ramas main (o master) y production están configuradas como protegidas.
   * Prohibición de force push y requerimiento de, al menos, una aprobación de Pull Request antes de fusionar.
   * Esto asegura que ningún cambio malintencionado o accidental sobrescriba el historial, facilitando el rastreo y la recuperación de versiones anteriores.
3. **Etiquetado (Tags) de Versiones**
   * Cada entrega oficial (hito) recibe un “tag” semántico en GitHub (por ejemplo: v1.0.0, Milestone-2-v1.1).
   * Estos “tags” funcionan como puntos de restauración inmutables: ante cualquier incidencia, se puede clonar el repositorio y hacer “checkout” de ese tag para recuperar exactamente el estado de código y entregables en aquel momento.
4. **Política de Copias de Seguridad Externas**
   * Adicionalmente al servicio de GitHub, se configura un trabajo automatizado (por ejemplo, mediante GitHub Actions o un servidor interno) que, con frecuencia semanal (o diaria en hitos críticos), exporta y asegura copias completas del repositorio en un almacenamiento aparte.
   * Estas copias se mantienen en un servidor de respaldo en la nube o en un almacenamiento interno con retención mínima de 90 días.
   * En caso de que GitHub experimente interrupciones prolongadas o que se requiera restaurar datos históricos, se podrán recuperar los últimos respaldos externos sin pérdida significativa de información.
5. **Monitoreo de Integridad de Datos**
   * Se habilita el registro de “audit logs” en GitHub (disponible en cuentas empresariales) para tener seguimiento de quién realiza cambios en la configuración del repositorio, incluidos accesos y modificaciones de permisos.
   * Cada seis meses (o tras un hito mayor), el equipo de auditoría interna revisa dichos logs para verificar que no haya actividad sospechosa y garantizar que las políticas de acceso se han aplicado correctamente.
6. **Planes de Recuperación ante Desastre**
   * En caso de requerir recuperar el proyecto completo (código y entregables), el flujo de recuperación consiste en:  
     + Clonar el respaldo externo más reciente en un entorno aislado.
     + Validar que los archivos de /entregables/, las etiquetas (tags) y las ramas protegidas estén completos.
     + Restaurar el repositorio en una nueva instancia de GitHub (o en GitHub Enterprise) mediante importación de datos y configuración de permisos idénticos.
     + Verificar que las acciones de integración continua (CI) y los che­queos automáticos se hayan reconfigurado correctamente para reanudar el flujo de trabajo sin interrupciones.
   * Este procedimiento deberá documentarse en detalle en /docs/ProcedimientoRecuperacion.md, indicando los pasos técnicos y los responsables asignados.
7. **Almacenamiento de Entregables Adicionales**
   * Además de versionar los entregables directamente en /entregables/, en GitHub se habilita la funcionalidad de “Releases”:  
     + Cada vez que se fusiona un hito a la rama principal, se genera una nueva Release que agrupa los archivos principales de ese hito (por ejemplo, documentos clave y binarios necesarios).
     + Las Releases quedan disponibles para descarga independiente, lo que facilita su recuperación sin necesidad de clonar el repositorio completo.