UD1. Control de versiones y documentación colaborativa

Descargar esta sección en PDF

1. Introducción

En el desarrollo de aplicaciones web, la gestión eficaz del código fuente y la documentación es un factor clave para asegurar la calidad, la trazabilidad y la colaboración entre desarrolladores. A lo largo del ciclo de vida de un proyecto, el código sufre múltiples cambios, revisiones y mejoras. Poder registrar, revertir, comparar y colaborar sobre esos cambios es imprescindible para cualquier equipo profesional.

Para cubrir esta necesidad, existen herramientas conocidas como sistemas de control de versiones, que permiten gestionar el historial del código fuente de manera estructurada. Aunque existen diferentes opciones, el sistema de control de versiones más utilizado actualmente en la industria es Git. Además, las plataformas colaborativas como GitHub, GitLab o Bitbucket han revolucionado la manera de trabajar en equipo, añadiendo a Git capacidades como revisión de código, integración continua, despliegue automático, gestión de incidencias y documentación.

Esta unidad didáctica tiene como objetivo que el alumno aprenda a trabajar con un sistema de control de versiones moderno, a utilizar herramientas colaborativas en proyectos reales, y a documentar adecuadamente el software y su despliegue. Estas habilidades serán la base sobre la cual se apoyará el resto del módulo, ya que acompañarán todas las fases del despliegue de una aplicación web.

A lo largo de esta unidad, se combinará la teoría con la práctica, y se fomentará el uso de buenas prácticas que son estándar en el mundo profesional. Se espera que los alumnos lleguen al final de la unidad siendo capaces de trabajar en un proyecto compartido de forma autónoma, utilizando herramientas actuales y cumpliendo criterios de calidad documental y técnica.

2. El control de versiones: concepto, utilidad e importancia

2.1. ¿Qué es un sistema de control de versiones?

Un sistema de control de versiones (VCS, por sus siglas en inglés) es una herramienta que permite registrar todos los cambios realizados en el código fuente (y en otros archivos) de un proyecto, de manera que cada modificación queda guardada con información sobre quién la hizo, cuándo, y por qué. Estos sistemas permiten:

- Volver a versiones anteriores del código si algo falla.
- Comparar cambios entre diferentes versiones.
- Trabajar en paralelo mediante ramas (branches).
- Fusionar aportaciones de varios desarrolladores.
- Llevar un registro detallado del historial del proyecto.

2.2. ¿Por qué es importante?

Sin un sistema de control de versiones, un equipo de desarrollo tendría que gestionar los cambios manualmente, copiando y renombrando archivos, lo cual es propenso a errores, ineficiente y difícil de escalar. Además, sin un control adecuado del historial, resulta casi imposible entender cómo ha evolucionado el software o quién introdujo una determinada modificación.

En el entorno profesional, el control de versiones es imprescindible no solo por la organización del código, sino también porque forma parte de procesos más amplios como la integración continua (CI), la entrega continua (CD), la revisión de código, la automatización de pruebas y el despliegue.

2.3. Ventajas principales del control de versiones

1. **Historial completo:** Cada cambio queda registrado. Es posible volver atrás en el tiempo o investigar el origen de un error.
2. **Trabajo en equipo:** Permite que varios desarrolladores trabajen simultáneamente sin pisarse unos a otros.
3. **Seguridad:** El código está duplicado en varios repositorios (locales y remotos), reduciendo el riesgo de pérdida de información.
4. **Ramas (branches):** Facilita el desarrollo de nuevas funcionalidades o la corrección de errores de forma aislada.
5. **Automatización y despliegue:** Es la base para configurar pipelines de CI/CD y para el despliegue automático de aplicaciones.
6. **Auditoría:** En entornos profesionales, es vital saber quién hizo qué cambios y cuándo, para responsabilizar o auditar procesos.

2.4. Tipos de sistemas de control de versiones

Existen tres grandes tipos de VCS según su arquitectura:

- **Centralizados (CVCS):** Existe un único repositorio central y todos los desarrolladores deben conectarse a él para obtener o enviar cambios. Ejemplo: Subversion (SVN).
- **Distribuidos (DVCS):** Cada desarrollador tiene una copia completa del repositorio, con todo su historial. Esto permite trabajar sin conexión y realizar operaciones locales antes de sincronizar con otros. Ejemplo: Git, Mercurial.
- **Híbridos:** Algunos sistemas como Perforce incorporan elementos de ambas arquitecturas.

Actualmente, **Git** es el estándar de facto en la industria, tanto por sus capacidades técnicas como por el ecosistema de herramientas y plataformas que lo rodean. En esta unidad nos centraremos exclusivamente en Git y su uso con plataformas colaborativas como GitHub.

En los siguientes apartados aprenderás no solo a instalar y configurar Git, sino también a trabajar con él de forma profesional: crear repositorios, trabajar con ramas, resolver conflictos, gestionar la colaboración en remoto y aplicar buenas prácticas en los mensajes de commit y en la organización del proyecto.

UD1. Control de versiones y documentación colaborativa

3. Introducción

En el desarrollo de aplicaciones web, la gestión eficaz del código fuente y la documentación es un factor clave para asegurar la calidad, la trazabilidad y la colaboración entre desarrolladores. A lo largo del ciclo de vida de un proyecto, el código sufre múltiples cambios, revisiones y mejoras. Poder registrar, revertir, comparar y colaborar sobre esos cambios es imprescindible para cualquier equipo profesional.

Para cubrir esta necesidad, existen herramientas conocidas como sistemas de control de versiones, que permiten gestionar el historial del código fuente de manera estructurada. Aunque existen diferentes opciones, el sistema de control de versiones más utilizado actualmente en la industria es Git. Además, las plataformas colaborativas como GitHub, GitLab o Bitbucket han revolucionado la manera de trabajar en equipo, añadiendo a Git capacidades como revisión de código, integración continua, despliegue automático, gestión de incidencias y documentación.

Esta unidad didáctica tiene como objetivo que el alumno aprenda a trabajar con un sistema de control de versiones moderno, a utilizar herramientas colaborativas en proyectos reales, y a documentar adecuadamente el software y su despliegue. Estas habilidades serán la base sobre la cual se apoyará el resto del módulo, ya que acompañarán todas las fases del despliegue de una aplicación web.

A lo largo de esta unidad, se combinará la teoría con la práctica, y se fomentará el uso de buenas prácticas que son estándar en el mundo profesional. Se espera que los alumnos lleguen al final de la unidad siendo capaces de trabajar en un proyecto compartido de forma autónoma, utilizando herramientas actuales y cumpliendo criterios de calidad documental y técnica.

4. El control de versiones: concepto, utilidad e importancia

4.1. ¿Qué es un sistema de control de versiones?

Un sistema de control de versiones (VCS, por sus siglas en inglés) es una herramienta que permite registrar todos los cambios realizados en el código fuente (y en otros archivos) de un proyecto, de manera que cada modificación queda guardada con información sobre quién la hizo, cuándo, y por qué. Estos sistemas permiten:

- Volver a versiones anteriores del código si algo falla.
- Comparar cambios entre diferentes versiones.
- Trabajar en paralelo mediante ramas (branches).
- Fusionar aportaciones de varios desarrolladores.
- Llevar un registro detallado del historial del proyecto.

4.2. ¿Por qué es importante?

Sin un sistema de control de versiones, un equipo de desarrollo tendría que gestionar los cambios manualmente, copiando y renombrando archivos, lo cual es propenso a errores, ineficiente y difícil de escalar. Además, sin un control adecuado del historial, resulta casi imposible entender cómo ha evolucionado el software o quién introdujo una determinada modificación.

En el entorno profesional, el control de versiones es imprescindible no solo por la organización del código, sino también porque forma parte de procesos más amplios como la integración continua (CI), la entrega continua (CD), la revisión de código, la automatización de pruebas y el despliegue.

4.3. Ventajas principales del control de versiones

1. **Historial completo:** Cada cambio queda registrado. Es posible volver atrás en el tiempo o investigar el origen de un error.
2. **Trabajo en equipo:** Permite que varios desarrolladores trabajen simultáneamente sin pisarse unos a otros.
3. **Seguridad:** El código está duplicado en varios repositorios (locales y remotos), reduciendo el riesgo de pérdida de información.
4. **Ramas (branches):** Facilita el desarrollo de nuevas funcionalidades o la corrección de errores de forma aislada.
5. **Automatización y despliegue:** Es la base para configurar pipelines de CI/CD y para el despliegue automático de aplicaciones.
6. **Auditoría:** En entornos profesionales, es vital saber quién hizo qué cambios y cuándo, para responsabilizar o auditar procesos.

4.4. Tipos de sistemas de control de versiones

Existen tres grandes tipos de VCS según su arquitectura:

- **Centralizados (CVCS):** Existe un único repositorio central y todos los desarrolladores deben conectarse a él para obtener o enviar cambios. Ejemplo: Subversion (SVN).
- **Distribuidos (DVCS):** Cada desarrollador tiene una copia completa del repositorio, con todo su historial. Esto permite trabajar sin conexión y realizar operaciones locales antes de sincronizar con otros. Ejemplo: Git, Mercurial.
- **Híbridos:** Algunos sistemas como Perforce incorporan elementos de ambas arquitecturas.

Actualmente, **Git** es el estándar de facto en la industria, tanto por sus capacidades técnicas como por el ecosistema de herramientas y plataformas que lo rodean. En esta unidad nos centraremos exclusivamente en Git y su uso con plataformas colaborativas como GitHub.

En los siguientes apartados aprenderás no solo a instalar y configurar Git, sino también a trabajar con él de forma profesional: crear repositorios, trabajar con ramas, resolver conflictos, gestionar la colaboración en remoto y aplicar buenas prácticas en los mensajes de commit y en la organización del proyecto.