Herramientas de Desarrollo

Semana 07
Trabajo colaborativo con sistema de control de versiones en la nube

Inicio

¿Tienen alguna consulta o duda sobre la clase anterior?

Logro de la Unidad

Al finalizar la unidad, el estudiante gestiona los sistemas de control de versiones para el desarrollo de una solución de software.

Imagen obtenida de: https://www.euroschoolindia.com/wp-content/uploads/2023/08/impact-of-school-leadership.jpg

Utilidad

- ¿Cómo se establece el trabajo Colaborativo usando la nube?

Universidad Tecnológica del Perú

contenido

Transformación

Universidad Tecnológica del Perú

1 Introducción

Introducción

El **trabajo colaborativo con un sistema de control de versiones en la nube** es una práctica que facilita la cooperación entre varios miembros de un equipo.

Especialmente en proyectos de desarrollo de software, pero también en otros tipos de proyectos que impliquen la creación de contenido digital.

2 Sistemas de control de versiones en la nube

Sistemas de control de versiones en la nube

Los sistemas de control de versiones en la nube son plataformas que permiten almacenar y gestionar los repositorios de código o archivos en línea, ofreciendo funcionalidades de control de versiones combinadas con acceso remoto y herramientas de colaboración en tiempo real.

Algunos de los sistemas más populares:
GitHub: Basado en Git, permite almacenamiento de proyectos y colaboración en línea.
GitLab: Ofrece características similares a GitHub, con la ventaja de ser completamente open-source en su versión auto-hospedada.
Bitbucket: Utiliza también Git y ofrece integración con otros servicios de Atlassian.

Sistemas de control de versiones en la nube

Beneficios de usar un sistema de control de versiones en la nube

Colaboración eficiente: Varias personas pueden trabajar en el mismo proyecto al mismo tiempo, realizando cambios y fusionándolos de manera sencilla.

Historial de versiones: Puedes seguir la evolución del proyecto a lo largo del tiempo, ver quién realizó qué cambios, y si es necesario, revertir a versiones anteriores.

Acceso remoto: Puedes trabajar desde cualquier lugar, siempre y cuando tengas acceso a la nube, lo que facilita el trabajo a distancia.

Seguridad: Los sistemas de control de versiones en la nube suelen ofrecer backups automáticos, lo que garantiza que los archivos no se pierdan en caso de un fallo local en el equipo.

Sistemas de control de versiones en la nube

…Beneficios de usar un sistema de control de versiones en la nube

Integración con herramientas de desarrollo: Muchos de estos servicios ofrecen integraciones con herramientas de gestión de proyectos, automatización de pruebas, despliegue continuo y otras funcionalidades.

Resolución de conflictos: Los sistemas de control de versiones como Git permiten gestionar los conflictos de manera efectiva cuando varios colaboradores modifican el mismo archivo al mismo tiempo.

Sistemas de control de versiones en la nube

Cómo usar un sistema de control de versiones en la nube

Crear un repositorio: El primer paso es crear un repositorio en una plataforma como GitHub o GitLab. Este repositorio servirá como lugar centralizado donde todos los archivos del proyecto se almacenarán.

Clonar el repositorio: Cada miembro del equipo puede clonar el repositorio en su máquina local para trabajar de manera independiente.

Realizar cambios: Cada persona trabaja en su propia copia del proyecto, realizando cambios y mejoras.

Hacer commits: Cada vez que un colaborador realiza un cambio significativo, lo registra con un *commit*. Este commit incluye una descripción breve de lo que se ha hecho.

Sistemas de control de versiones en la nube

…Cómo usar un sistema de control de versiones en la nube

Push: Después de hacer los commits localmente, se realiza un *push* para enviar los cambios al repositorio en la nube.

Pull: Antes de comenzar a trabajar, es importante hacer un *pull* para asegurarse de tener la versión más reciente del proyecto. Esto es esencial para evitar conflictos con los cambios de otros colaboradores.

Fusionar cambios: Si dos personas modifican el mismo archivo, puede haber un conflicto. En ese caso, se debe resolver el conflicto manualmente y luego realizar un *commit* y un *push* para completar el proceso.

Sistemas de control de versiones en la nube

Herramientas adicionales para facilitar el trabajo colaborativo

Gestión de tareas: Integrar el control de versiones con herramientas de gestión de proyectos como **Trello**, **Jira**, o **Asana** permite organizar las tareas, asignar responsabilidades y hacer seguimiento de avances.

Revisiones de código: Los sistemas de control de versiones en la nube permiten realizar *pull requests* o *merge requests*, que son revisiones de código por parte de otros miembros del equipo antes de integrar cambios en la rama principal.

Automatización de pruebas: La integración con herramientas como **Travis CI**, **Jenkins** o **GitHub Actions** permite automatizar pruebas y procesos de despliegue, garantizando que el código está libre de errores antes de ser lanzado.

Documentación: Es crucial mantener una buena documentación dentro del proyecto. Plataformas como GitHub permiten añadir archivos de documentación, como el archivo README.md, para guiar a otros colaboradores.

Sistemas de control de versiones en la nube

Consejos para un trabajo colaborativo efectivo

Comunicación constante: Mantén una buena comunicación con el equipo, utilizando plataformas de mensajería o video llamadas cuando sea necesario para aclarar dudas o discutir los cambios.

Uso de ramas: Trabaja en **ramas (branches)** para evitar conflictos entre diferentes tareas. La rama principal debe ser lo más estable posible, mientras que las ramas secundarias pueden ser para nuevas funcionalidades o correcciones.

Reglas claras para los commits: Establece convenciones claras para los mensajes de commit, para que todos sepan qué cambios se han hecho y por qué.

Revisiones de código: Asegúrate de revisar el código de otros antes de integrarlo, esto ayuda a mantener la calidad del proyecto y detectar posibles errores

3 Flujo de Trabajo Colaborativo con Git y Repositorios Remotos

Flujo de Trabajo Colaborativo con Git y Repositorios Remotos

1. Crear y configurar un repositorio remoto: Un miembro del equipo (generalmente el administrador o líder del proyecto) crea un repositorio en una plataforma como **GitHub** o **GitLab**.
Ejemplo:

```
git init
git remote add origin https://github.com/usuario/repositorio.git
```

2. Clonar el repositorio remoto: Los colaboradores clonan el repositorio en sus máquinas locales para comenzar a trabajar en el proyecto.
Ejemplo:

```
git clone https://github.com/usuario/repositorio.git
```

Flujo de Trabajo Colaborativo con Git y Repositorios Remotos

3. Crear ramas para trabajar en funcionalidades específicas: Cada colaborador crea una rama para trabajar en una nueva característica o corrección de errores. Esto permite que varias personas trabajen en el mismo proyecto sin interferir directamente en la rama principal (usualmente llamada main o master).

Ejemplo:

```
git checkout -b feature/nueva-caracteristica
```

4. Realizar cambios y commits: Los colaboradores hacen cambios en su código y luego los registran en el repositorio local con un commit.

Ejemplo:

```
git add .
git commit -m "Añadida nueva funcionalidad"
```

Flujo de Trabajo Colaborativo con Git y Repositorios Remotos

5. Sincronización con el repositorio remoto: Antes de hacer un push para compartir los cambios, es recomendable realizar un git pull para asegurarse de que los cambios más recientes del repositorio remoto están integrados en la rama local. Esto ayuda a evitar conflictos.
Ejemplo:

```
git pull origin main
```

6. Subir los cambios al repositorio remoto: Una vez que se han hecho los cambios y se ha probado localmente, los colaboradores suben su trabajo a la rama correspondiente del repositorio remoto.
Ejemplo:

```
git push origin feature/nueva-caracteristica
```

Flujo de Trabajo Colaborativo con Git y Repositorios Remotos

7. Crear un Pull Request (PR): En plataformas como GitHub y GitLab, los colaboradores pueden crear un PR (Pull Request) para solicitar la fusión de su rama con la rama principal (usualmente main). Esto es una solicitud de revisión por parte de otros miembros del equipo.
Ejemplo:
En GitHub, seleccionas tu rama, haces clic en "Pull Request", agregas una descripción de los cambios y solicitas la revisión de otro colaborador.

8. Revisión de código: Los demás miembros del equipo revisan el código propuesto en el PR. Pueden hacer comentarios, sugerir cambios o aprobar la fusión.

Flujo de Trabajo Colaborativo con Git y Repositorios Remotos

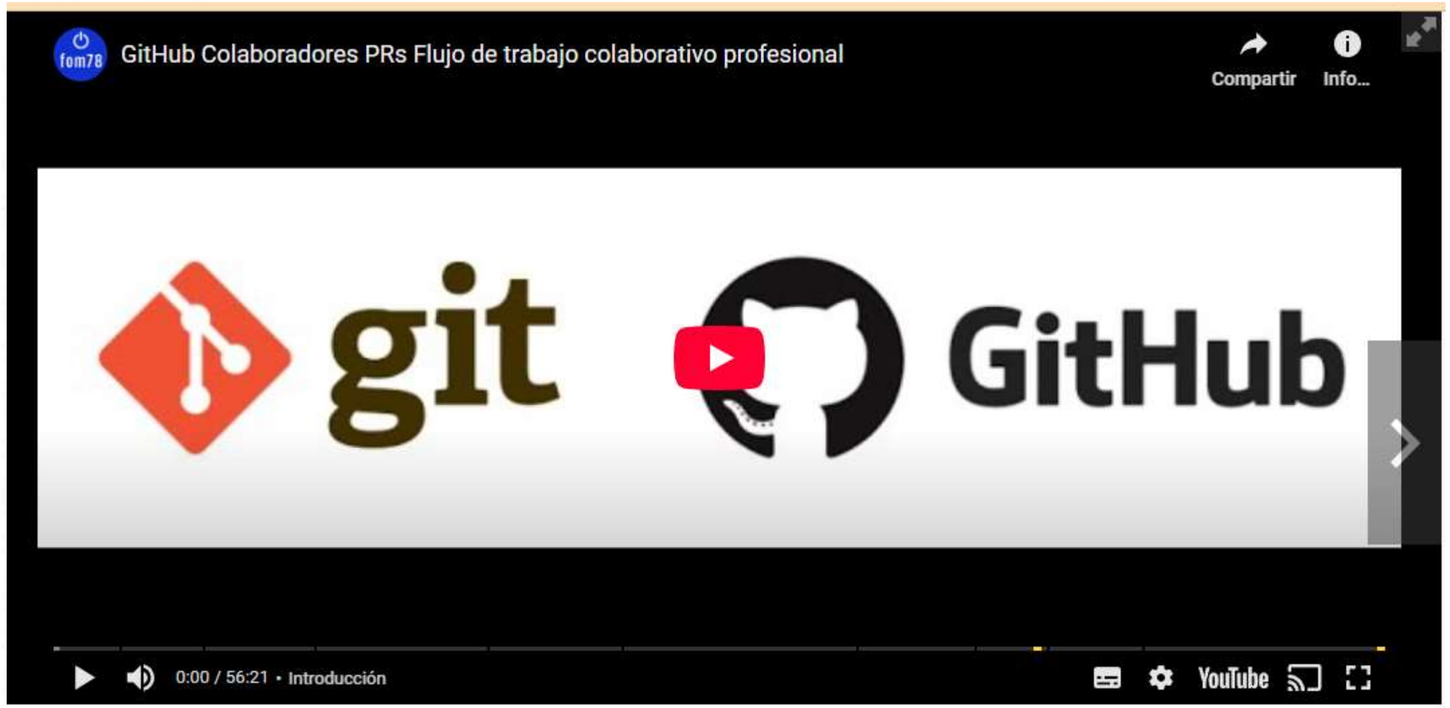
9. Fusión (Merge): Si el PR es aprobado, el responsable de la fusión (usualmente un líder de equipo o el mismo autor del PR) fusiona los cambios en la rama principal.
Ejemplo:
Usando la interfaz web de GitHub o GitLab, se hace clic en "Merge Pull Request".

10. Eliminar ramas locales y remotas: Una vez que se ha fusionado el PR, las ramas ya no son necesarias y se eliminan tanto localmente como en el repositorio remoto.
Ejemplo:

```
git branch -d feature/nueva-caracteristica
git push origin --delete feature/nueva-caracteristica
```

4 Enlace

GitHub Colaboradores PRs Flujo de trabajo colaborativo



GitHub Colaboradores PRs **Flujo de trabajo colaborativo** profesional

https://es.video.search.yahoo.com/video/play;_ylt=AwrNZG0L5uBnSXURuFi..Qt.;_ylu=c2VjA3NyBHNsawN2aWQEZ3BvcwMz?p=Flujo+de+Trabajo+Colaborativo+con+Git+y+Repositorios+Remotos&vid=e2e41d3ebdd9cda0ec74d57dc166dbd0&turl=https%3A%2F%2Ftse4.mm.bing.net%2Fth%3Fid%3DOVP.A6zT3eN8gxMmQk5yoyZmRQEsDh%26pid%3DApi%26h%3D225%26w%3D300%26c%3D7%26rs%3D1&rurl=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3D3DHMEGP-MHE&tit=GitHub+Colaboradores+PRs+%3Cb%3EFlujo%3C%2Fb%3E+%3Cb%3Ede%3C%2Fb%3E+%3Cb%3Etrabajo%3C%2Fb%3E+%3Cb%3Ecolaborativo%3C%2Fb%3E+profesional&c=2&sigr=UFFpnni14eXW&sigt=3lzq4qSr4Fq0&sigi=y3ralWHxtC83&fr2=p%3As%2Cv%3Av&h=225&w=300&l=3382&age=1687670269&fr=mcafee&type=E210ES714G0&tt=b

Práctica

Clonar un repositorio remoto para trabajar en él.

Crear una nueva rama llamada feature/correccion-error y cambiar a ella

Ver el estado de los archivos en tu repositorio local antes de hacer un commit

Hacer varios cambios en tu proyecto, ¿cómo los registrarías (commit)?

Subir tus cambios a la rama feature/correccion-error del repositorio remoto

Práctica

Hacer un pull de la rama principal (main) antes de hacer un push de tus cambios

Fusionar tu rama feature/correccion-error con la rama main utilizando un Pull Request (PR)

Eliminar la rama local después de fusionarla con la rama main

Eliminar la rama remota llamada feature/correccion-error después de fusionarla

Cierre

1. ¿Qué es un repositorio remoto en Git?
2. ¿Qué es un Pull Request (PR)?
3. ¿Cuál es la diferencia entre git merge y git rebase?
4. ¿Qué es una rama (branch) en Git y por qué es útil en el trabajo colaborativo?

Imagen obtenida de:
https://dpemfoco.com.br/wp-content/uploads/2019/08/Departamento-de-Pessoal-em-Foco-Checklist-Check-list-Checklists.png

Bibliografía

Hernández Bejarno, Miguel. *Ciclo de vida de desarrollo ágil de software seguro.* Fundación Universitaria Los Libertadores. https://tubiblioteca.utp.edu.pe/cgi-bin/koha/opac-detail.pl? biblionumber=36016

Guillamón Morales, Alicia. (). *Manual desarrollo de elementos software para gestión de sistemas.*
Editorial CEP, S.L. https://tubiblioteca.utp.edu.pe/cgi-bin/koha/opac-detail.pl?biblionumber=34982

Chacon, S., & Straub, B. (2014). *Pro Git* (2nd ed.). Apress.
Enlace: https://git-scm.com/book/es/v2

Poulton, N. (2017). *Docker Deep Dive*. Independently published.
Enlace: https://www.nigelpoulton.com/dvd/

Universidad
Tecnológica
del Perú