INGENIERÍA WEB - PRÁCTICA 01
Introducción a GIT para gestión de la configuración

Gestión de la Configuración

La gestión de la Configuración del Software es un conjunto de actividades diseñadas para identificar y definir los elementos en el sistema que probablemente cambien, controlando el cambio de estos elementos a lo largo de su ciclo de vida, estableciendo relaciones entre ellos, definiendo mecanismos para gestionar distintas versiones de estos elementos, y auditando e informando de los cambios realizados.

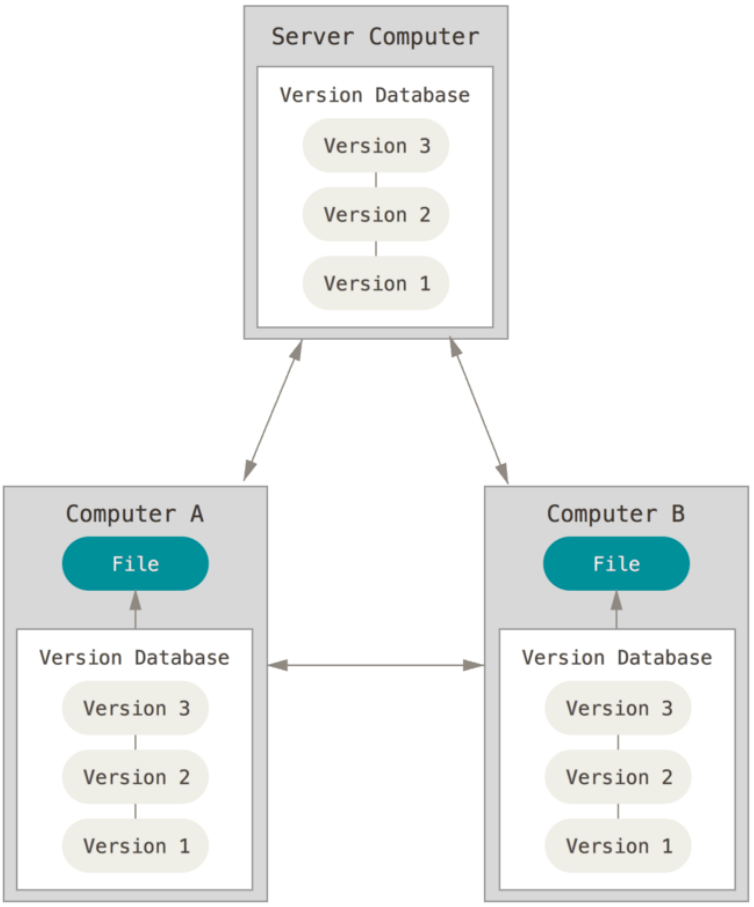
El propósito de la gestión de la configuración es establecer y mantener la integridad de los productos de software a través del ciclo de vida del proceso de software.

Sistema de control de Versiones

Un correcto control de versiones garantiza el registro de todos los cambios realizados sobre un fichero o conjunto de ficheros a lo largo del tiempo, de manera que podamos acceder a versiones antiguas posteriormente. Mediante un sistema control de versiones completo podríamos verificar en todo momento qué cambios han sido realizados y por quién, retroceder a estados anteriores del proyecto y realizar un seguimiento detallado de su evolución.

El sistema de control de versiones más empleado, y a la vez uno de los más rústicos y desaconsejables es el “copy&paste” y renombrado de ficheros. Con mayor o menor complejidad no es difícil encontrar en nuestro escritorio ficheros como “TrabajoFinal”, “TrabajoFinalV2”, “TrabajoFinalDefinitivoDeVerdad2.0_entregar_si”. Esto no deja de ser un sistema de control de versiones... aunque con muchas carencias.

La estructura de sistema de control de versiones más común en los entornos profesionales se conoce como “*Distributed Version Control System*”.



Los clientes crean *réplicas* completas del repositorio y sus historiales.

Con Git cada vez que "commiteas" o guardas el estado de tu proyecto, se toma una "foto" de todos sus ficheros en ese momento y se almacena una referencia a esa foto. Si los ficheros no han sido modificados, no se realiza una nueva "foto".

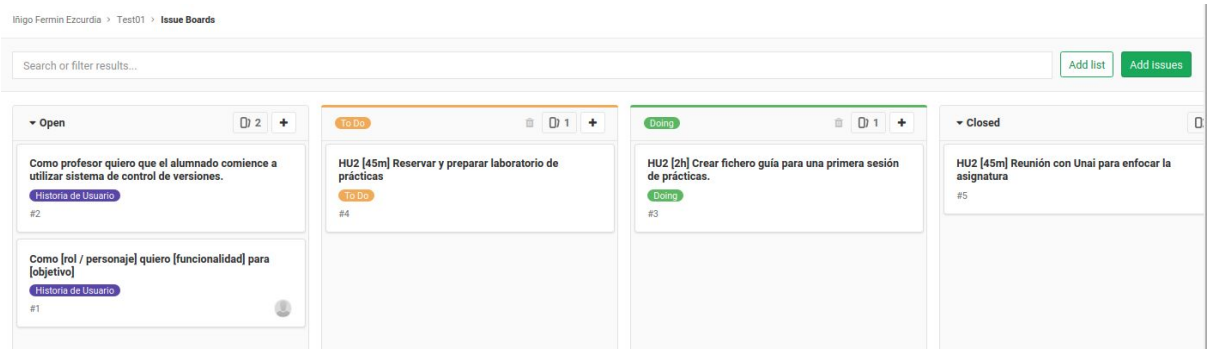
Una correcta gestión de código y versiones es indispensable en cualquier proyecto, sea este software o web. En esta asignatura trabajaremos por equipos de desarrollo y será importante el carácter evolutivo de nuestro proyecto, por lo que vamos a emplear gitlab como repositorio de código y git como gestor de configuración.

En esta primera práctica vamos a comenzar a familiarizarnos con este entorno, y mantendremos su uso durante toda la asignatura. Su correcto uso tendrá un peso importante en la evaluación del proyecto final.

Identificar, desglosar y documentar taréas en gitlab.

Para seguir una metodología ágil emplearemos en Gitlab el “panel de tareas” (Issues->boards) del apartado “tareas” (issues). Este panel está disponible en cada repositorio personal y también lo estará en los repositorios grupales. Lo utilizaremos para identificar, desglosar y documentar todas las tareas que componen una práctica o proyecto. También nos servirá para asignar responsables y prioridades y realizar un seguimiento del avance del proyecto.

Ejemplo de un board:



Antes de continuar, repasemos algunos conceptos:

Repositorio:
“*Almacen o lugar donde se guardan ciertas cosas*”. En nuestro caso, emplearemos repositorios en gitlab, allí se almacenarán y organizarán todos los archivos de nuestro proyectos.

Historia de Usuario:
Término empleado en desarrollo ágil. Descripciones, siempre muy cortas y esquemáticas, que resumen la necesidad concreta y objetivo de un usuario concreto. Su función principal es identificar problemas percibidos, proponer soluciones y estimar el esfuerzo que requieren implementar las ideas propuestas. Se componen de multitud de tareas o issues. Ej:

- *“Como estudiante quiero completar la practica1 para aprender los fundamentos sobre control de versiones.”*

Tarea / Issue:
Cada una de las responsabilidades o labores que componen una historia de usuario. Habitualmente atribuidas a una única persona como responsable. Siempre ligadas a una historia de usuario concreta. Se debe indicar a qué historia de usuario pertenecen. Ej:

- HU #2 Descargar y leer el primer guión de prácticas
- HU #2 Inicializar mi repositorio personal
- HU #2 Crear un listado de comandos útiles de git empleando estilado markdown

Columna Open: En esta columna del board/panel se ubican aquellas historias de usuario que todavía tienen tareas/issues pendientes de completar.

Columna To Do: En esta columna del board/panel se ubican aquellas tareas que todavía no han comenzado a trabajarse.

Columna Doing: En esta columna del board/panel se ubican aquellas tareas en las que se ha comenzado a trabajar.

Columna Closed: En esta columna del board/panel se ubican aquellas historias de usuario y tareas que han sido finalizadas.

---

1. Accede a gitlab:
   https://eim-laboratoriovirtual.unavarra.es/gitlab/
   Usuario y contraseña del correo de la universidad sin el "@e.unavarra.es"

2. Debemos tener un proyecto/repositorio con nuestro nombre asignado, si no lo tenemos debemos contactar con el profesor.

3. En dicho proyecto accedemos al apartado issues/board y en medio del board clickamos el botón "Add default list". Esto nos creará las listas por defecto. (Open, ToDo, Doing y Closed)

4. Dándole al botón "+" de la parte superior de la lista "Open" añadimos la primera historia de usuario que nos servirá en el aprendizaje para tener delante una plantilla de cómo debemos pensar las historias de usuario, en el título pondremos el siguiente texto :
   Como [rol / personaje] quiero [funcionalidad] para [objetivo]

5. En adelante cuando pensemos nuevas historias de usuario que añadir al backlog de nuestro proyecto siempre las redactaremos como hemos visto en el template anterior sustituyendo el rol/personaje , la funcionalidad y el objetivo.

6. Vamos a crear nuestra primera historia de usuario real, añadir una issue nueva, la #2 que diga:
   "*Como estudiante quiero completar la practica1 para aprender los fundamentos sobre control de versiones.*"

7. En la lista "ToDo" tendremos que ir añadiendo todas las tareas que pide esta práctica como parte de la historia de usuario #2, para ello en cada tarea escribimos al comienzo la historia de usuario en la que se está realizando la tarea ("HU #2" en este caso). Además en cada tarea creada podemos (y debemos) configurar:
   a. Responsable de la tarea: (Assignees) Establecer quién o quiénes va a realizar esta labor.

b. Tiempo estimado: En la descripción de la tarea, mediante un comentario con el comando /estimate establecemos una estimación de cuánto tardaremos en realizarla.
c. Tiempo trabajado: En la descripción de la tarea, mediante un comentario con el comando /spend registraré cuánto tiempo he trabajado hasta ahora en esta tarea.
d. Fecha objetivo: (Due Date) Fecha deadline para la cual esta tarea o historia de usuario debería estar completada.

8. Repasa el resto de este guión de prácticas e identifica las distintas tareas que propone, creando una nueva tarea con su correspondiente estimación temporal para cada una de ellas. Recuerda asignar responsables, tiempos y fechas objetivo.

Comenzando a utilizar Git

1) Instalar Git

Para instalar git en un sistema Linux basta con utilizar el gestor de paquetes del propio sistema. Abrimos una terminal e introducimos el siguiente comando y aceptamos cuando se nos requiera:
`sudo apt install git-all`

Si la instalación concluye con éxito podremos ejecutar el siguiente comando para verificar qué versión de git acabamos de instalar.
`git --version`

```
inigofermin_ezcurdia@inf-lab-59667:~$ git --version
git version 2.7.4
```

2) Añadir los ficheros en producción a nuestro repositorio git

Identificarnos, para que nuestras modificaciones y commits tengan un "autor":

```
inigofermin_ezcurdia@inf-lab-59667:~$ git config --global user.name "MiNombre MiApellido"
inigofermin_ezcurdia@inf-lab-59667:~$ git config --global user.email miMail@unavarra.es
inigofermin_ezcurdia@inf-lab-59667:~$ git config --list
user.name=MiNombre MiApellido
user.email=miMail@unavarra.es
```

Solicitar ayuda:
`git help`
Solicitar ayuda con un comando concreto:
`git help comando`

Vamos a crear nuestro primer repositorio local, donde se almacenarán todos los ficheros de nuestra página web y de la evolución de la asignatura.
Crearemos la carpeta "iw" en /var/www/html/ de manera que exista la ruta:
`/var/www/html/iw`

Le asignamos permisos de escritura a nuestro usuario para toda la carpeta iw:
`sudo chown -R alumno /var/www/html/iw`

Accederemos a la carpeta y emplearemos el comando para crear el repositorio git vacío:
`cd /var/www/html/iw/`
`git init`

Lo enlazamos con el repositorio remoto (puedes consultar la url correcta pulsando en "clone" en tu proyecto de gitlab) (Recuerda que a pesar de que gitlab nos informe de que el subdominio es *"inf-lvirtual.unavarra.es*" realmente nuestro subdominio es "***eim-laboratoriovirtual.unavarra.es***")
`git remote add origin https://eim-laboratoriovirtual.unavarra.es/gitlab/ingweb2020/apellido-nombre.git`

Ver el estado de los ficheros con respecto al índice
```
git status
```

Crea un fichero vacío, nómbralo como quieras...

Añadir los cambios locales al índice de nuestro repositorio local
```
git add -A
```

Consultar el estado de los ficheros con respecto al índice de nuevo
```
git status
```

Realizar un commit (confirmar cambios) al repositorio local
```
git commit -m'commit inicial'
```

Enviar el commit al repositorio remoto
```
git push -u origin master
```

Ver log de los commit que tiene el repositorio local (consulta tu commit también desde la interfaz web de gitlab)
```
git log
```

Ver el estado de los ficheros con respecto al índice
```
git status
```

3) Traer una primera copia de nuestro repositorio al entorno local o de desarrollo y realizar cambios

Traer a local (máquina desarrollo) una copia de todo el repositorio. (puedes consultar la url correcta pulsando en "clone" en tu proyecto de gitlab)
```
git clone https://eim-laboratoriovirtual.unavarra.es/gitlab/ingweb2020/apellido-nombre.git
```

Añadimos (o editamos si ya existe) un fichero "README.md" en la raiz de nuestro repositorio El fichero README.md es un fichero presente en la amplia mayoría de repositorios, en él se incluye información básica sobre el proyecto, una breve descripción de su uso y despliegue, o cualquier información que el equipo de desarrollo considere relevante. De momento nosotros colocamos en su interior la siguiente información:
-Fecha actual
-Nombre y apellidos

Registramos todos los cambios realizados (en este caso añadir un nuevo fichero)
```
git add -A
```
Realizamos un commit sobre el repositorio local
```
git commit -m'añadir fichero README.md'
```

Y enviamos los últimos commits al repositorio remoto
`git push`

Explora tu repositorio desde gitlab. ¿Está todo correcto y como lo esperabas? ¿Están presentes los ficheros sobre los que hemos realizado commits y push?

4) Actualizar el estado de nuestro repositorio local

Supón que tu mismo, u otro usuario ha modificado tu repositorio remoto, ya sea modificando algún fichero, borrando o añadiendo ficheros nuevos. En tu máquina local, tu repositorio local no refleja el estado actual del proyecto, tienes una versión desactualizada del proyecto.
Para "traer" todas esa actualizaciones del repositorio remoto al local, tenemos que emplear el siguiente comando:
`git pull`

Ahora bien... nos encontramos con distintas casuísticas con las que todavía no tenemos muy claro cómo operar:

- ¿Qué ocurre si dos o más usuarios realizan modificaciones sobre un mismo fichero y uno de ellos "pushea" sus cambios antes que el otro?
  Conflicts, Merge
- Manejamos un proyecto grande, podemos mantener distintas líneas de desarrollo? Branches
- Nuestro proyecto contiene binarios, compilados, o multitud de ficheros que no tiene sentido que sean rastreados mediante git.
  .gitignore
- Sé que un montón de ficheros han sido modificados, pero no sé cuales han sido dichas modificaciones
  git diff
- He metido la pata, quiero volver a un estado anterior del proyecto...
  git checkout

Pero ya veremos todo esto, y mucho más, más adelante...

5) Estilado del README.md

Como ves, el fichero README tiene extensión .md, esta extensión corresponde al lenguaje de marcado "markdown" que nos permite dar formato, estilado y estructura.
En el siguiente enlace se muestran diversos ejemplos de estilado mediante markdown:

https://markdown-it.github.io/

Dedica por lo menos 10 minutos a revisar los ejemplos expuestos en dicho enlace.

Ejercicios prácticos:

Cualquiera de los siguiente textos si se encuentran en el mensaje del commit indican que la issue #xxx se ha solucionado con el commit que se realiza y quedan el commit y la issue enlazados en gitlab.

- fixes #xxx
- fixed #xxx
- fix #xxx
- closes #xxx
- close #xxx
- closed #xxx

1. Crea las siguientes historias de usuario en el board de gitlab:
   -***Como alumno quiero incluir en el fichero README.md información sobre el uso básico de git.***
   -***Como alumno quiero preparar el entorno del repositorio para entregar las prácticas individuales de la asignatura.***
   -Como alumno quiero preparar el entorno del repositorio para entregar las prácticas grupales de la asignatura.

2. Crea las siguientes tareas vinculadas a dichas historias de usuario: (con su correspondiente estimación horaria y vínculo a historia de usuario)
   -Actualizar fichero README.md
   -Crear carpetas de prácticas.
   -Crear carpeta para la primera práctica y dotarla de contenido.
   -Inicializar entorno grupal de trabajo.

3. Modifica el fichero README.md y utiliza markdown para incluir una sección en la que resumas, de manera ordenada, los comandos básicos de git que has aprendido de momento. Cada vez que añadas información sobre un nuevo comando, acompáñalo de su commit y push correspondiente.

4. Crea una carpeta “practicas” en tu repositorio. En su interior irás completando y entregando todas las futuras prácticas.

5. Crea en su interior otra carpeta expresamente para esta primera práctica. Incluye en ella este mismo documento que estás leyendo y un fichero index.html con el siguiente contenido:

```html
<!doctype html>
<html lang="es">
  <head>
    <meta charset="utf-8">
    <title>Test</title>
    <meta name="description" content="Página de ejemplo para Ingeniería Web">
    <meta name="author" content=Nombre Apellido>
    <link rel="stylesheet" href="./css/styles.css">

  </head>

  <body>
    <div>
      <h1> Funciona! </h1>
      <p> Hola mundo! </p>
    </div>
    <script src="js/scripts.js"></script>
  </body>
</html>
```

6. Abre index.html con un navegador web. ¡Enhorabuena, has creado tu primera web! Un poco simple... pero ya evolucionaremos a cosas más complejas...

7. Explora tu repositorio en gitlab, observa cómo se exponen los cambios y qué información aporta. Familiarízate con la interfaz de gitlab.

8. No olvides actualizar el board!

9. Busca 2 compañer@s con los que realizar el proyecto grupal a lo largo del curso e informad al profesor del grupo que conformáis.

10. Actualizad todos los miembros del equipo en vuestros repositorios personales el fichero README.md para incluir al comienzo de este la información sobre a qué grupo pertenecéis.

11. Se os asignará un nuevo repositorio grupal, modificad cada uno desde su PC el fichero README.md de la siguiente manera:
    a. Todos los usuarios crean la carpeta /var/www/html/iw_group en su máquina.
    b. Todos los usuarios configuran git con su nombre y mail y clonan e inicializan el repositorio remoto grupal en /var/www/html/iw_group
    c. El primer usuario creará el fichero README.md e incluirá en él su nombre, hará commit y push de los cambios realizados.
    d. El segundo usuario hará pull del repositorio grupal, incluirá también su nombre y hará commit y push de los cambios realizados.
    e. El tercer usuario hará pull del repositorio grupal, incluirá también su nombre y hará commit y push de los cambios realizados.
    f. Observad desde gitlab los efectos de todos estos cambios.

12. No olvides actualizar el board en gitlab. Marca como completadas todas las tareas completadas y estima de nuevo las no completadas!