

|  |  |
| --- | --- |
| Resultado de imagen para github  MANUAL DE CONTROL DE VERSIONES | EQUIPO |

**¿Qué es un control de versiones y por qué debería utilizarlo?**

Se llama control de versiones a los métodos y herramientas disponibles para controlar todo lo referente a los cambios en el tiempo de un archivo.

Difícilmente un archivo de código o un documento de texto están terminado con la primera escritura; necesita cambios o reescrituras para corregir errores, modificar su contenido... A medida que el documento cambia existen dos opciones, mantener un historial de cambios o dejar que evolucione sin memoria. El control de versiones es un método estándar para mantener esta memoria haciendo además que sea útil para el desarrollo futuro.

En documentos sencillos como un ensayo o un pequeño programa la memoria no es algo esencial, pero en la escritura de un libro o un programa con centenares de páginas y una docena de manos involucradas no hay otra manera de trabajar. Esta es precisamente la palabra clave, mantener un control de las versiones de todos los archivos de un proyecto es una manera de trabajar completamente estandarizada; todas las prácticas tienen un nombre. La buena noticia es que todo este formalismo es generosamente recompensado por el uso de sistemas de control de versiones automáticos como CVS, Subversion, git...

**¿Por qué un control de versiones para documentos?**

A medida que en nuestras investigaciones utilizamos herramientas digitales y almacenamiento en formato digital, se vuelve relevante reflexionar sobre cómo optimizar la gestión de nuestros datos. Más aún, el control de versiones puede ser indispensable si tenemos intención de colaborar con otros investigadores. Aunque el control de versiones fue diseñado en sus orígenes para tratar archivos de código, creemos que la gestión de documentos también se beneficiaría. La lección que proponemos no cubre todas las ventajas del control de versiones pero al finalizar podrás llevar a cabo las siguientes tareas:

* rastrear el desarrollo y los cambios de tus documentos
* registrar los cambios que has hecho de una manera que puedas entender posteriormente
* experimentar con versiones distintas de un documento al mismo tiempo que conservas la más antigua
* fusionar dos versiones de un documento y administrar los conflictos existentes entre distintas versiones
* revertir cambios y volver atrás gracias al historial de versiones anteriores de tu documento

En concreto, el control de versiones es útil para facilitar la colaboración. De hecho, una de las razones que explican el origen del control de cambios es que permitirá a varias personas trabajar al mismo tiempo en un proyecto de considerables dimensiones y utilizar Git para administrar las fuentes del núcleo Linux. Utilizar un control de versiones favorece la colaboración debido a su flexibilidad. Por ejemplo, dos personas pueden trabajar en un mismo documento al mismo tiempo y ‘fusionar’ los cambios. Si existe un ‘conflicto’ entre las dos versiones, el sistema de control permitirá al usuario ver el conflicto y decidir cómo fusionar las dos versiones dando lugar a una ‘tercera’ versión. De esta manera, se conservara la ‘historia’ del documento, es decir, las versiones anteriores y, en consecuencia, se podra revertir el proceso eligiendo una versión más antigua.

No es necesario, sin embargo, poner en marcha un control de versiones para todos tus documentos. En algunas ocasiones resulta muy útil; por ejemplo, para escribir artículos, libros o tesis doctorales.

La implementación del control de versiones que se propone en esta lección está pensada para que los documentos sean públicos. No obstante, puedes utilizar un control de versión y mantener tus documentos ocultos de manera permanente o bien hasta que decidas publicarlos en línea.

**¿Cómo se especifican las versiones?**

El método más habitual de asignar una versión a un documento es mediante un número o un grupo de números. No existe un modo fijo de numerar una versión, se deja al criterio de cada desarrollador. Sí existen ciertas prácticas habituales en la numeración como el uso de tres cifras o la numeración decimal.

**Para este proyecto se utilizaran los criterios de**

**Versiones X, Y, Z.**

* La primera cifra (X) indica la versión mayor del documento. Si empieza con un cero significa que el documento aún no está listo o no cumple con los requerimientos mínimos. Cada cambio en esta cifra denota una reescritura o la incompatibilidad con versiones mayores anteriores.
* La segunda cifra (Y) indica la versión menor del documento. Denota cambios en el contenido o en la funcionalidad del documento pero no lo suficientemente importantes como para decir que ya no es el mismo. Cuando se estrena una versión mayor se deja la versión menor a cero pero aún así se incluye de modo que la segunda versión mayor sería la 2.0
* La tercera cifra (Z) indica la segunda versión menor. Indica que el documento se ha corregido pero que no se ha añadido ni eliminado nada relevante. Cuando se estrena una versión menor, es decir, cuando la segunda versión menor es igual a cero; suele omitirse.
* Pueden añadirse recursivamente versiones menores, algunos proyectos hacen uso de ellas cuando los criterios de selección de versiones no corresponden con los definidos en los puntos anteriores. El núcleo del sistema operativo Linux utiliza una numeración de cuatro cifras.
* Se colocara la fecha del dia de la creación del documento con el formato dd-mm-aaaa

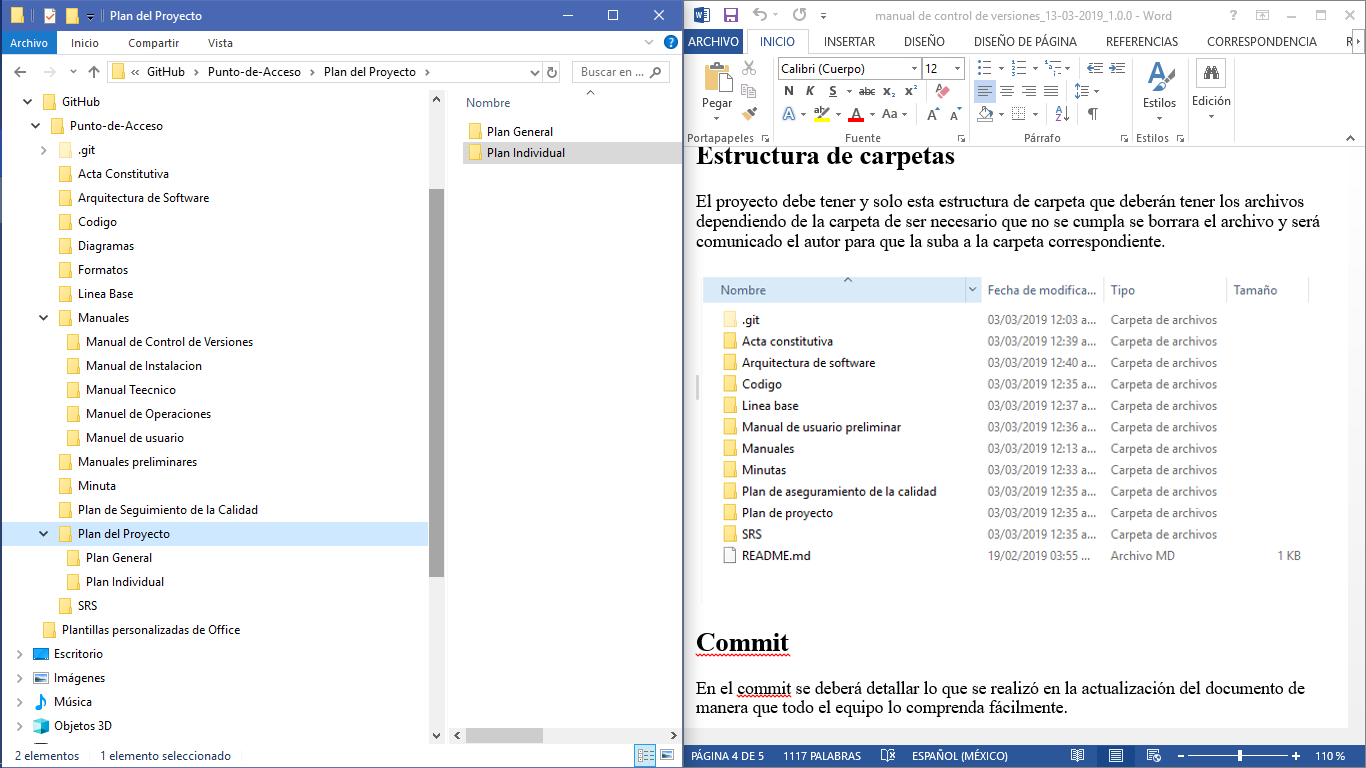
Seguido de un guión bajo y la versión que corresponde siguiendo los criterios anteriores. Por ejemplo: **prueba\_03-03-2019-1.1.1**

* **E**n el caso de las verificaciones la estructura del nombre tendrá que ser de la siguiente manera.
  + Deberá almacenarse en el mismo sitio del documento que se está verificando.
  + La palabra Check\_(nombre completo del archivo verificado)\_fecha de creación(formato dd-mm-aaaa)\_versión del documento. por ejemplo:

**Check\_(prueba\_03-03-2019\_1.1.1)\_04-03-2019\_0.0.1**

**Estructura de carpetas**

El proyecto debe tener y solo esta estructura de carpeta que deberán tener los archivos dependiendo de la carpeta de ser necesario que no se cumpla se borrara el archivo y será comunicado el autor para que la suba a la carpeta correspondiente.



**Commit**

En el commit se deberá detallar lo que se realizó en la actualización del documento de manera que todo el equipo lo comprenda fácilmente.

**El respaldo del proyecto**

Será realizado por el gerente de soporte y la líder de equipo, cada semana se hará el respaldo si algún miembro necesita hacer uso del respaldo tendrá que mandar un oficio al líder que contenga el nombre del que necesita el respaldo y el uso que le dará.

La persona que necesite acceder al respaldo de los datos del proyecto deberá descargar los archivos que necesite que están en el repositorio de GitHub.

Las copias de seguridad de los datos se ubican en la carpeta respaldos pd, al igual que los datos obtenidos del equipo en el process dashboard.

**Carpeta línea base**

Se agregan los archivos que ya están verificados por todas las partes involucradas y no se deben tocar, en dado caso que un miembro del equipo necesite hacer un cambio a un archivo base deberá llenar un formato para enviar al líder y hacer valoración si se requiere hacer el cambio.

nombrar todos los archivos

los archivos en línea base deberá tener el nombre del archivo con la terminación de la versión que es “\_v1.0.0” para estos archivos no deben llevar la fecha por ejemplo:

“nombre\_v1.0.0”.

los archivos finales son:

Manual de usuario

Manual mantenimiento

Manual de operaciones

Código

diagrama (base de datos)

diagrama electronico

gráficas de process dashboard