Documentación y sistema de control de versiones

Despliegue de aplicaciones web

DAW – Despliegue de aplicaciones web (2º - 0614)

Documentación de aplicaciones web

- Aspectos fundamentales que conviene documentar en una aplicación:
  1. **La interfaz**: Qué hace (no como lo hace) una función o un método de una clase, qué parámetros hay que pasar y qué devuelve. Esta información es tremendamente útil para las personas que utilizan funciones o clases diseñadas por otros. Documento independiente del código fuente (manual de uso).
  2. **La implementación**: Indicar cómo está implementada cada función, cómo se lleva a cada paso, por qué se utiliza determinada variable, qué algoritmo se utiliza, qué hacen los métodos privados de una clase. Toda esta información resulta interesante a quienes tengan que depurar o actualizar bloques de código de la aplicación. Normalmente incluida en el código fuente.
  3. **La toma de decisiones**: Por qué se ha implementado de determinada forma y no de otra la aplicación, por ejemplo, para analizar el rendimiento de la aplicación y optimización de recursos. Esto resulta interesante a nivel de implementación para los desarrolladores y a nivel funcional para los responsables del desarrollo.
- Cada vez que se modifica algo en el código (actualizaciones, corrección de errores, etc.) hay que reflejarlo también en el manual de uso, lo cual implica un doble trabajo. Se requiere **automatizar** este proceso.

Documentación de aplicaciones web

- Los entornos de programación modernos (NetBeans, Eclipse, etc.) son capaces de obtener la información de los comentarios, de forma que la muestran en el "autocompletado" de código.
- Existen herramientas que permiten generar documentación de forma automática a partir del código fuente:
  - **JavaDoc**: es la herramienta de Oracle por excelencia para generar documentación en formato HTML a partir del código desarrollado en Java. La mayoría de los IDE de Java generan automáticamente la documentación, entre ellos Netbeans 8.2.
  - **phpDocumentor**: herramienta escrita en PHP para generar documentación automáticamente vía código fuente PHP y como salida el estándar PHPDoc.
  - **Doxygen**: esta herramienta permite generar documentación en bastantes lenguajes de programación, como por ejemplo C++,Java, C, Fortran, VDHL, CLI, Python, IDL y también con ciertos matices PHP y D. Funciona en la mayoría de los sistemas Unix, Windows y MAC OS X

phpDocumentor

- Permite generar la documentación de varias formas/formatos:
  - Desde línea de comandos (php CLI - Command Line Interpreter).
  - Desde interfaz web (incluida en la distribución).
  - Desde código. Dado que phpDocumentor está desarrollado en PHP, podemos incluir su funcionalidad dentro de scripts propios.
- Es necesario especificar los siguientes parámetros:
  1. El directorio en el que se encuentra nuestro código. PhpDocumentor se encargará luego de recorrer los subdirectorios de forma automática.
  2. Opcionalmente los paquetes (@pakage) que deseamos documentar, lista de ficheros incluidos y/o excluidos y otras opciones interesantes para personalizar la documentación.
  3. El directorio en el que se generará la documentación.
  4. Si la documentación va a ser pública (sólo interfaz) o interna (en este caso aparecerán los bloques private y los comentarios @internal).
  5. El formato de salida de la documentación:
     - HTML a través de un buen número de plantillas predefinidas (podemos definir nuestra propia plantilla de salida).
     - PDF.
     - XML (DocBook). Podemos transformar (XSLT) a cualquier otro utilizando nuestras propias reglas y hojas de estilo.

phpDocumentor

- En phpDocumentor la documentación se distribuye en bloques "DocBlock“ que se colocan siempre justo antes del elemento al que documentan.
- Los elementos que pueden ser documentados son: define, function, class, class vars, include/require/include_once/require_once y global variables
- También se puede incluir documentación global a nivel de fichero y clase mediante la marca @package.
- Dentro de cada DockBlock se pueden incluir marcas que serán con un significado especial:
  - @access: Si @access es 'private' no se genera documentación para el elemento (a menos que se indique explícitamente).
  - @author: Autor del código.
  - @copyright: Información sobre derechos.
  - @deprecated: Para indicar que el elemento no debería utilizarse, ya que en futuras versiones podría no estar disponible.
  - @example: Permite especificar la ruta hasta un fichero con código PHP. phpDocumentor se encarga de mostrar el código resaltado (syntax-highlighted).
  - @ignore: Evita que phpDocumentor documente un determinado elemento.
  - @internal: Para incluir información que no debería aparecer en la documentación pública, pero sí puede estar disponible como documentación interna para desarrolladores.
  - @link: Para incluir un enlace (http://...) a un determinado recurso.
  - @see: Se utiliza para crear enlaces internos (enlaces a la documentación de un elemento).
  - @since: Permite indicar que el elemento está disponible desde una determinada versión del paquete o distribución.
  - @version: Versión actual del elemento.
- Sólo en bloques de determinados elementos
  - @global: Para especificar el uso de variables globales dentro de una función.
  - @param: Para documentar parámetros que recibe una función.
  - @return: Valor devuelto por una función.

```php
/**
* Descripción breve (una línea)
*
* Descripción extensa. Todas las líneas que
* sean necesarias
* Todas las líneas comienzan con *
<- Esta línea es ignorada
*
* Este DocBlock documenta la función suma()
*/
function suma()
{
...
}
```

```php
/**
 * This is the summary for a DocBlock.
 *
 * This is the description for a DocBlock. This text may contain
 * multiple lines and even some _markdown_.
 *
 * * Markdown style lists function too
 * * Just try this out once
 *
 * The section after the description contains the tags; which provide
 * structured meta-data concerning the given element.
 *
 * @author  Mike van Riel <me@mikevanriel.com>
 *
 * @since 1.0
 *
 * @param int    $example  This is an example function/method parameter description.
 * @param string $example2 This is a second example.
 */
```

phpDocumentor

- Instalación:
  - Instalación Apache y PHP

```
# apt-get install systemd apache2          # apt-get install php libapache2-mod-php
```

  - Instalación phpDocumentor

```
# apt-get install php-pear                 # pear config-set data_dir /var/www/html
# pear install –-alldeps PhpDocumentor
# mkdir /var/www/html/docs                 # chown www-data /var/www/html/docs/
```

  - Comprobación. `http://localhost/PhpDocumentor`
- Uso:
  - Línea de comandos (`# phpdoc -h`):

```
# phpdoc -o [formato_generado] -d [carpeta_proyectos_php] –t [carpeta_ficheros_generados]
```

    - Ejemplo:

```
# phpdoc -o HTML:frames:phpedit -d /var/www/html/ -t /var/www/html/docs/
```

    Genera la documentación en formato HTML (también es posible en formato PDF, CHM) de los proyectos de la carpeta /var/www/html/ y se almacene dicha documentación en la carpeta /var/www/html/docs/
  - Entorno web (http://localhost/PhpDocumentor)
    - Con fichero de configuración (.ini):
      - En el directorio "/var/www/html/PhpDocumentor/user" creación de ficheros ".ini" a partir de "default.ini"
        - Cambiar (como mínimo) el contenido de etiquetas:
          - "target". Ruta donde guardar la documentación generada.
          - "directory". Ruta donde se encuentran los archivos del proyecto.
      - Utilización de fichero de configuración (".ini") en la solapa "Config"
    - Sin fichero de configuración
      - Rellenar información en las solapas (Files, Output y Options)
      - No utilizar fichero de configuración

JavaDoc

- Genera APIs (Aplication Programing Interface) en formato HTML de un archivo de código fuente Java.
- La documentación se escribe en comentarios que comienzan con /** y que terminan con */ , comenzando cada línea del comentario por * a su vez, dentro de estos comentarios se puede escribir código HTML y operadores para que interprete (generalmente precedidos por @).
- Javadoc localiza las etiquetas incrustadas en los comentarios de un código Java, las cuales comienzan con el símbolo @ y son sensibles a mayúsculas/minúsculas. Una etiqueta se sitúa siempre al principio de una línea, o sólo precedida por espacio(s) y asterisco(s). Tipos de etiquetas:
  - **Etiquetas de bloque**: sólo se pueden utilizar en la sección de etiquetas que sigue a la descripción principal. Son de la forma: @etiqueta
  - **Etiquetas inline**: se pueden utilizar tanto en la descripción principal como en la sección de etiquetas. Son de la forma: {@tag}, es decir, se escriben entre los símbolos de llaves.

JavaDoc

- Etiquetas de las clases
  - @author: este atributo permite declarar el nombre del autor de la clase.
  - @deprecated: para declarar si la clase está obsoleta y no se recomienda su uso.
  - @param: definición de un parámetro de un método.
  - @see: se asocia con otro método o clase.
  - @serial: describe el motivo del campo y sus posibles valores.
  - @since: la versión del producto que se está desarrollando.
  - @version: es la versión numérica de una clase o un método.
- Etiquetas de los métodos
  - @deprecated: para declarar si algún elemento del método está obsoleto y no se recomienda su uso.
  - @exception: es sinónimo de @throws.
  - @param: describe el parámetro o parámetros que recibe el método.
  - @return: describe el valor de salida del método.
  - @see: se asocia con otro método o clase.
  - @serialData: describe el formato de serialización usado en el método.
  - @since: la versión del producto que se está desarrollando.
  - @throws: describe una excepción lanzada por el método que debe ser tenida en cuenta.

JavaDoc

- La mayor parte de los entornos de desarrollo Java incluyen un botón para llamar a javadoc, así como opciones de configuración.
- JavaDoc (javadoc) forma parte del JDK y puede ser ejecutado directamente sobre el código fuente Java.
  - Instalación: JavaDoc forma parte de JDK, y por lo tanto se instala con este

```
# apt install default-jdk
```

  - Uso:

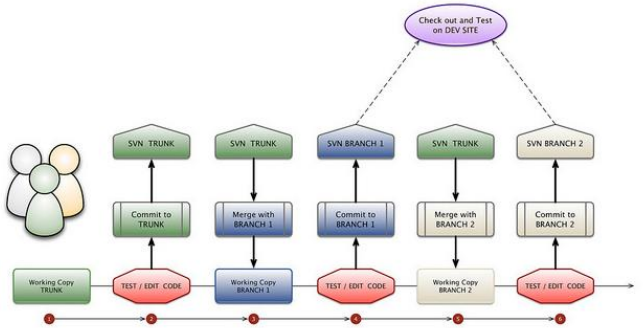
```
# javadoc {packages|sourcefiles} [options] [@argfiles]
```

  - Ejemplo (como root):

```
# mkdir /var/www/html/HolaMundo
# javadoc -d /var/www/html/HolaMundo HolaMundo.java
  http://localhost/HolaMundo
```

http://www.emambrilla.com/HolaMundo.java
http://www.emambrilla.com/ejemplojavadoc.java

Sistemas de control de versiones

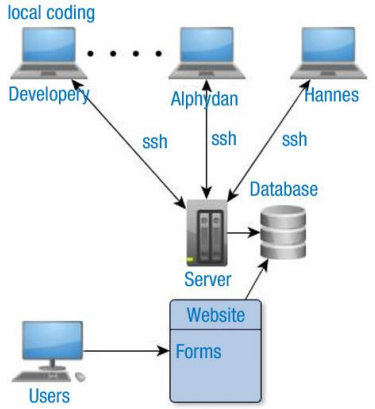


- Los sistemas de control de versiones son programas que permiten gestionar un repositorio de archivos y sus distintas versiones.
- Utilizan una arquitectura cliente-servidor en donde el servidor guarda la(s) versión(es) actual(es) del proyecto y su historia.
- Conceptos básicos:
  - **Revisión**: Es una visión estática en el tiempo del estado de un grupo de archivos y directorios. Posee una etiqueta que la identifica. Suele tener asociado metadatos como pueden ser:
    - Identidad de quién hizo las modificaciones.
    - Fecha y hora en la cual se almacenaron los cambios.
    - Razón para los cambios.
    - De qué revisión y/o rama se deriva la revisión.
    - Palabras o términos clave asociados a la revisión.
  - **Copia de trabajo**: También llamado "Árbol de trabajo", es el conjunto de directorios y archivos controlados por el sistema de control de versiones, y que se encuentran en edición activa. Está asociado a una rama de trabajo concreta.
  - **Rama de trabajo (o desarrollo)**: En el más sencillo de los casos, una rama es un conjunto ordenado de revisiones. La revisión más reciente se denomina principal (main) o cabeza. Las ramas se pueden separar y juntar según como sea necesario, formando un grafo de revisión.
  - **Repositorio**: Lugar en donde se almacenan las revisiones. Físicamente puede ser un archivo, colección de archivos, base de datos, etc.; y puede estar almacenado en local o en remoto (servidor).
  - **Conflicto**: Ocurre cuando varias personas han hecho cambios contradictorios en un mismo documento (o grupo de documentos); los sistemas de control de versiones solamente alertan de la existencia del conflicto. El proceso de solucionar un conflicto se denomina resolución.
  - **Cambio**: Modificación en un archivo bajo control de revisiones. Cuando se unen los cambios en un archivo (o varios), generando una revisión unificada, se dice que se ha hecho una combinación o integración.
  - **Parche**: Lista de cambios generada al comparar revisiones, y que puede usarse para reproducir automáticamente las modificaciones hechas en el código.
- La forma habitual de trabajar consiste en mantener una copia en local y modificarla. Después actualizarla en el repositorio. Como ventaja tenemos que no es necesario el acceso continuo al repositorio.
- Algunos sistemas de control de versiones permiten trabajar directamente contra el repositorio; en este caso tenemos como ventaja un aumento de la transparencia, a pesar de que como desventaja existe el bloqueo de ficheros.

Sistemas de control de versiones

- Ciclo de operaciones:
  - **Descarga de ficheros inicial (Checkout)**:
    - El primer paso es bajarse los ficheros del repositorio.
    - El "checkout" sólo se hace la primera vez que se usan esos ficheros.
  - **Ciclo de trabajo habitual**:
    - **Modificación de los ficheros**, para aplicar los cambios oportunos como resultado de la aportación de cada una de las personas encargadas de manipular el código de los ficheros.
    - **Actualización de ficheros en local (Update)**: Los ficheros son modificados en local y, posteriormente, se sincronizan con los ficheros existentes en el repositorio.
    - **Resolución de conflictos (si los hay)**: Como resultado de la operación anterior es cuando el sistema de control de versiones detectará si existen conflictos, en cuyo caso informa de ello y siendo los usuarios implicados en la manipulación del código afectado por el conflicto, los encargados de solucionarlo.
    - **Actualización de ficheros en repositorio (Commit)**: Consiste en la modificación de los ficheros en el repositorio; el sistema de control de versiones comprueba que las versiones que se suben estén actualizadas.
- Funciones:
  - Varios clientes pueden sacar copias del proyecto al mismo tiempo.
  - Realizar cambios a los ficheros manteniendo un histórico de los cambios:
    - Deshacer los cambios hechos en un momento dado.
    - Recuperar versiones pasadas.
    - Ver históricos de cambios y comentarios.
  - Los clientes pueden también comparar diferentes versiones de archivos.
  - Unir cambios realizados por diferentes usuarios sobre los mismos ficheros.
  - Sacar una "foto" histórica del proyecto tal como se encontraba en un momento determinado.
  - Actualizar una copia local con la última versión que se encuentra en el servidor. Esto elimina la necesidad de repetir las descargas del proyecto completo.
  - Mantener distintas ramas de un proyecto.

Sistemas de control de versiones



- Sistemas de control de versiones centralizados vs. distribuidos
  - Sistemas de control de versiones centralizados (Centralized Version Control Systems o CVCSs en inglés).
    - Tienen un único servidor que contiene todos los archivos versionados, y varios clientes que descargan los archivos de ese lugar central.
    - Implementaciones: Subversion y Perforce
    - Ventajas:
      - Control detallado de qué puede hacer cada uno
      - Administración simplificada frente a la gestión de bases de datos locales en cada cliente
    - Desventajas: Punto único de fallo que representa el servidor centralizado.
  - Sistemas de control de versiones distribuidos (Distributed Version Control Systems o DVCSs en inglés).
    - Los clientes no sólo descargan la última instantánea de los archivos sino que replican completamente el repositorio.
    - Implementaciones: Git, Mercurial, Bazaar o Darcs
    - Ventajas: Si un servidor muere, y estos sistemas estaban colaborando a través de él, cualquiera de los repositorios de los clientes puede copiarse en el servidor para restaurarlo, ya que cada vez que se descarga una instantánea se hace una copia de seguridad completa de todos los datos.

Sistemas de control de versiones: Git

- Git es un sistema rápido de control de versiones, está escrito en C y se ha hecho popular sobre todo a raíz de ser el elegido para el kernel de linux.
- Git NO modela ni almacena sus datos como una lista de cambios en los archivos, sino hace una "foto" del aspecto de todos tus archivos en ese momento, y guarda una referencia a esa instantánea cada vez que se confirma un cambio, o se guarda el estado del proyecto.
- Casi cualquier operación es local, la mayoría de las operaciones en Git sólo necesitan archivos y recursos locales para operar.
- Git posee integridad debido a que todo es verificado mediante checksum antes de ser almacenado, y es identificado/confirmado a partir de ese momento mediante dicho checksum.

Sistemas de control de versiones: Git (Funcionamiento)

- Git tiene tres estados principales en los que se pueden encontrar los archivos:
  - Confirmado (committed) significa que los datos están almacenados de manera segura en la base de datos local. Si una versión concreta de un archivo está en el directorio de Git.
  - Modificado (modified) estado en el que se ha modificado el archivo pero todavía no se ha confirmado a la base de datos. Si ha sufrido cambios desde que se obtuvo del repositorio, pero no se ha preparado.
  - Preparado (staged) significa que se ha marcado un archivo modificado en su versión actual para que vaya en la próxima confirmación. Si ha sufrido cambios desde que se obtuvo del repositorio, pero ha sido añadida al área de preparación.
- Esto nos lleva a las tres secciones principales de un proyecto de Git:
  - El directorio de Git (Git directory): Almacena los metadatos y la base de datos de objetos para tu proyecto. Es la parte más importante de Git, y es lo que se copia cuando se clona un repositorio desde otro ordenador.
  - El directorio de trabajo (working directory): Es una copia de una versión del proyecto. Estos archivos se sacan de la base de datos comprimida en el directorio de Git, y se colocan en disco para que se pueda usar o modificar.
  - El área de preparación (staging area): es un sencillo archivo, generalmente contenido en tu directorio de Git, que almacena información acerca de lo que va a ir en la próxima confirmación. A veces se denomina el índice, pero se está convirtiendo en estándar el referirse a ello como el área de preparación.
- El flujo de trabajo básico en Git consiste en:
  1. Modificar una serie de archivos en el directorio de trabajo.
  2. Preparar los archivos, añadiendo instantáneas de ellos al área de preparación.
  3. Confirmar los cambios, lo que toma los archivos tal y como están en el área de preparación, y almacena esa instantánea de manera permanente en el directorio de Git.

Sistemas de control de versiones: Git (Instalación)

- Desde los repositorios

```
# apt install git
```

- Desde el código fuente

```
# apt install make libssl-dev libghc-zlib-dev libcurl4-gnutls-dev libexpat1-dev gettext unzip
# cd /tmp
# wget https://github.com/git/git/archive/v2.43.0.zip -O git.zip
# unzip git.zip
# cd git-*
# make prefix=/usr/local all
# make prefix=/usr/local install
```

Sistemas de control de versiones: Git (Configuración I)

- Lista completa con todas las opciones contempladas

```
$ git config -help
```

- Establecer el nombre de usuario y dirección de correo electrónico, información que será incluida en las confirmación de los cambios (commits)

```
$ git config --global user.name "alumno"
$ git config --global user.email alumno@example.com
```

- Editor de texto por defecto que se utilizará cuando Git necesite que introduzcamos un mensaje (p.e.: nano, vi, emacs, vim,…)

```
$ git config --global core.editor nano
```

- Herramienta de diferencias por defecto, usada para resolver conflictos de unión (merge). Git admite: kdiff3, tkdiff, meld, xxdiff, emerge, vimdiff, gvimdiff, ecmerge y opendiff.

```
$ git config --global merge.tool vimdiff
```

- Comprobar la configuración que tenemos

```
$ git config –-list
```

- Ayuda

```
$ git help <comando>
$ git <comando> --help
$ man git-<comando>
```

Sistemas de control de versiones: Git (Configuración II)

- Creación e inicialización del repositorio (“proyecto”)

```
# cd /var/lib/git
# mkdir proyecto.git
# cd proyecto.git
# git init
# git add <fichero>
# git commit –a ”Primer commit”
# git branch –M main
```

- Acceso vía git (git://localhost, puerto 9418)

```
# git daemon –-base-path=/var/lib/git –-detach –-syslog --export-all
```

- Acceso vía gitweb (http://localhost/gitweb)

```
# apt-get install systemd apache2 php libapache2-mod-php
# apt-get install gitweb
# a2enmod cgi rewrite
# service apache2 start
```

Sistemas de control de versiones: Git (Trabajando)

- Clonado del proyecto

```
$ git clone git://servidor/proyecto.git proyecto
```