Tema 2 - Despliegue e Instalación (DCA)

HISTORIAL DE REVISIONES

NÚMERO	FECHA	MODIFICACIONES	NOMBRE

Índice

Logo DLSI

Tema 2 - Despliegue e Instalación
Curso 2018-2019

1. Desarrollo de software. Despliegue e instalación.

EN ESTE TEMA VAMOS A TRATAR LOS SIGUIENTES ASPECTOS DEL DESARROLLO E INSTALACIÓN DE SOFTWARE:

- Versiones de desarrollo
- Versiones estables
- Ramas de desarrollo vs. ramas de corrección de fallos
- Versiones debug / release
- Directorios de datos y de archivos binarios.

2. Preliminares

ASUMIMOS QUE...

- Trabajamos en *equipo* y no de forma individual
- Trabajar en *equipo* requiere de *disciplina* y el *uso de normas comunes* conocidas por todos los componentes del mismo.
- Desarrollamos un producto concreto con la aparición de versiones nuevas de forma periódica.
- Para instalar una aplicación... previamente la hemos de desarrollar de manera que esté en un estado `instalable` y `usable`.

3. Consejo: Cread equipos de desarrollo prácticos.

- No dejad pequeños detalles sin arreglar solo porque nadie lo quiere hacer.
- Si una parte del equipo comienza a funcionar mal... muy probablemente lo acabará haciendo todo el equipo.
- Debe haber comunicación entre las diversas partes de un equipo, además debe ser fluida, sin reticencias y estructurada.
- Aunque haya diversos subequipos, de cara al exterior todos deben comunicarse de la misma manera (términos, formato de documentos, etc...), de cara al interior la comunicación debe generar un debate activo y constructivo.
- Evitad repeticiones o duplicados de cualquier tipo (documentación, código, FALLOS!, etc...).
- Las fases de un proyecto: `análisis`, `diseño`, `codificación` y `testeo` no se deben llevar a cabo aisladamente.
- Procurad *automatizar* todas las acciones posibles que un equipo realiza. Esto ayudará a asegurar la *consistencia* del trabajo desarrollado.

4. Desarrollo. Uso de ramas.

- El grupo de desarrolladores ha de fijar una política de creación y uso de ramas.
- Inicialmente tendremos una única rama, normalmente conocida como "`master`" o "`trunk`".
- Estas ramas no tienen en principio nada que ver con los `SCV` que veremos en otro tema, aunque se emplean en algunos de ellos y con estos mismos nombres.
- A lo largo del tema emplearemos uno u otro nombre indistintamente.
- Esta rama representa la línea principal de desarrollo de un proyecto software antes de ser distribuído para su uso.

5. Gestión de la rama *master* (I)

- La gestión debería estar hecha por una única persona
- Esta responsable decide que elementos se aceptan en ella, tales como
  - Añadidos
  - Borrados
  - Modificaciones
- Todo ello encaminado a producir la siguiente versión **estable** de nuestro software.
- Esta rama siempre tiene las últimas modificaciones a nuestro software (**cutting edge**) y, por tanto, puede no funcionar del todo bien o contener nuevos errores.
- El código que **compone** esta rama no es **estable** hasta que el **gestor** de la misma así lo diga.

6. Gestión de la rama *master* (II)

- El gestor de esta rama, en cierto modo, se comporta más bien como un **integrador** que como un desarrollador.
- **Caso de uso**: Desarrollo del nucleo Linux y el papel de **Linus Torvalds** en él:

> `"However, he stated in 2012 that his own personal contribution is mostly merging code written by others, with little programming..."`
>
> — Linus Torvalds *Creador del núcleo linux*

- En este vídeo lo explica la propia Linux Foundation:

7. Gestión de la rama *master* (III)

- Este concepto de la rama *master* aplicada a todo el proyecto... se puede aplicar a subproyectos del mismo.
- Cada subproyecto tendría su rama *master* y un coordinador de la misma.
- Este coordinador se hará cargo de todo lo relativo al código de esta rama...
- Y cuando se encuentre en un estado `apropiado` enviará los cambios al coordinador general.
- No hace falta decir que esta subdivisón se puede aplicar tantas veces como sea necesaria en virtud de la complejidad de cada subproyecto.

8. Gestión de la rama *master* (IV)

- En este vídeo de Greg Kroah Hartman tienes una explicación más detallada de todo el proceso de desarrollo del núcleo Linux, haciendo hincapié en los subsistemas, en cómo se gestionan etc. . . :

9. Gestión de la rama *master* (V)

- En este otro vídeo Greg Kroah Hartman explica como contribuir `parches` al núcleo `linux`, lo cual es una forma de pasar información de una rama a otra. . . :
- Este vídeo también te puede ser útil para el último tema de la asignatura.

10. Rama(s) de desarrollo vs. rama *estable*

- Una vez publicada una nueva versión de nuestro producto, su código fuente constituye *su propia rama estable*, p.e.: *v1.0.0*
- El gestor/coordinador de la rama master comienza a recibir nuevas propuestas que no están en la *v1.0.0*. Estas propuestas se irán añadiendo en la nueva rama de desarrollo o `trunk` o `master`.
- Al mismo tiempo se van recogiendo avisos de fallos que aparecen en la versión actualmente *estable*, es decir: *v1.0.0*
- El gestor de esta rama estable los va incorporando y pasado un tiempo publica la nueva vesión estable con fallos corregidos: *v1.0.1*
- Evidentemente, estos bugs también se corrigen en la rama master.

11. Números o etiquetas de versión

- Representan una información útil tanto para usuarios como para programadores.
- **Para los usuarios**, porque saben qué versión de nuestro software emplean exactamente y eso les vale a la hora de reportar fallos o solicitar mejoras.
- **Para los programadores**, para identificar a qué código exactamente se refiere un informe de fallo o solicitud de añadido de una característica nueva.

12. Formato de etiquetas de versión

EXISTEN MULTITUD DE ELLOS, VEAMOS ALGUNOS:

- En su gran mayoría suelen estar formados por números, varios números separados por el carácter "`.`", p.e.: "`A.B`" o "`A.B.C`".
- En ocasiones suelen incluir alguna letra o también la fecha, de esta manera se indica más claramente de cuándo es el software que usamos.
- Podemos encontrar números de versión formados por:
  - Dos números (`A.B`), p.e.: `5.0`. Al primero se le conoce como número *mayor* y al segundo como número *menor*.
  - Tres números (`A.B.C`), p.e.: `5.0.2`. El tercer dígito se conoce como número *micro*.
  - Algunos desarrolladores emplean cuatro dígitos, siendo el cuarto lo que se llama *número de compilación*.

13. Etiquetas de versión: caso de uso (I)

- Las etiquetas de versión no son más que un modo que tiene el fabricante de un software de identificarlo claramente.
- Las etiquetas de versión no tienen porqué estar formadas por números...
- Por tanto su significado depende de lo que el fabricante quiera. Veamos un caso concreto en el mundo del *software libre*.

14. Etiquetas de versión: caso de uso (II)

- Es bastante común emplear 3 dígitos como número de versión

  (`A.B.C`)
  - Al aumentar el número *mayor* (A) se indica que el software sufre importantes *cambios* y/o *mejoras*.
  - El numero *menor* (B) puede ser **par** o **impar**. Si es `par` indica que estamos en una rama *estable* del código, mientras que si es `impar` indica que es una rama de desarrollo, inestable y que está sufriendo constantes cambios.
  - El número *micro* (C) representa un **avance** en la rama que indica el número *menor*.
- Si la rama es estable este avance se corresponde con corrección de fallos.
- Si la rama es de desarrollo este avance se corresponde con corrección de fallos e incorporación de nuevas características.

15. Etiquetas de versión: caso de uso (III)

Veamos algunos ejemplos concretos y excepciones a lo comentado:

- Proyecto Gtk
- Proyecto Gimp
- Proyecto Kde
- Proyecto Qt
- Proyecto Gcc
- Ikiwiki

16. Despligue. Caso de uso: Núcleo Linux (I)

- A lo largo de su historia ha empleado etiquetas de versión formadas por tres dígitos.
- Durante las versiones con número mayor "`1`" y "`2`", si el número menor era par representaba una versión estable del código y si era impar una versión de *desarrollo*.
- Esto fué así hasta la versión "`2.6.x`". Siendo la última versión con esta etiqueta la "`2.6.39`".
- A partir de este punto la versión pasó a ser la "`3.x.y`".
- En 2015 (y tras solicitar la opinión de desarrolladores) Linus Torvalds hizo un cambio de número mayor de versión: "`3.x.y`" $\rightarrow$ "`4.x.y`".

17. Despligue. Caso de uso: Núcleo Linux (II)

- Cada nueva versión va precedida de la apertura de una "*ventana temporal*" (por parte de Linus Torvalds) de incorporación de añadidos por parte de desarrolladores.
- A las dos semanas se cierra la posibilidad de añadir nuevas características y se publica una versión "*Release Candidate*": 3.x.y-RC1.
- Desde este instante y durante aproximadamente 4 versiones RC, sólo se corrigen fallos en los nuevos añadidos y detectados en código ya existente.
- El resultado final es un código con el mismo número mayor (`3`) y el número menor se incrementa en una unidad (`x =x+1`), mientras que el número micro pasa a valer "`0`".
- El número micro se irá incrementando para representar corrección de fallos.

18. Versiones de desarrollo y depuración

- Durante la fase de desarrollo de un proyecto es normal que aparezcan fallos y haya que depurarlo.
- Posteriormente, una vez finalizado y listo para distribuir, lo que queremos es que funcione lo más rápido posible y/o usando la menor cantidad de memoria posible.
- Para que funcione un depurador a nivel de código fuente, nuestro proyecto debe estar compilado y enlazado con una serie de opciones especiales...
- Del mismo modo, para que funcione lo más rápido posible y/o con el menor consumo de memoria, necesitamos compilar y enlazar con otra serie de opciones...
- Algunos sistemas de configuración automática de proyectos permiten hacer este tipo de cosas de manera sencilla. Veremos el caso concreto de cmake.

19. Prácticas en grupo e individuales:

- **Grupo:** Echad un vistazo a semver, tratad de explicarnos con palabras sencillas la especificación del *versionado semántico*.
- **Individual:** Elige cualquier código de una práctica que tengas de cualquier asignatura... si no tienes nada a mano... recurre al socorrido `hola mundo`.
- Implementa de manera manual dos opciones de compilación, una para producir una versión de "`depuración`" y otra para producir una versión de "`distribución`".
- Simula la existencia de:
  - Una rama `master`.
  - Una rama `estable`.
- Simula el avance y/o modificaciones en cada una de las dos ramas anteriores. Ayúdate de la creación de un subdirectorio en *master* y en *estable* que contenga una copia del código de partida y sobre ella haces la modificación.

ENTREGA:

- La práctica se entregará en pracdlsi en las fechas allí indicadas.

20. Aclaraciones

EN NINGÚN CASO ESTAS TRANSPARENCIAS SON LA BIBLIOGRAFÍA DE LA ASIGNATURA.

- Debes estudiar, aclarar y ampliar los conceptos que en ellas encuentres empleando los enlaces web y bibliografía recomendada que puedes consultar en la página web de la ficha de la asignatura y en la web propia de la asignatura.