

## Herramientas de trabajo (2)

Miguel Angel Piña Avelino

Ingeniería de Software,  
Facultad de Ciencias, UNAM

26 de agosto de 2018

# Índice

- ① ¿Cómo funciona una aplicación web?
- ② Ciclo de vida de peticiones a servidores Java
- ③ Sistemas de control de versiones (Git)
- ④ Usando Git

## ¿Cómo funciona una aplicación web?

# Conceptos

- HTTP

# Conceptos

- HTTP
- Ciclo de vida de una petición HTTP

# Conceptos

- HTTP
- Ciclo de vida de una petición HTTP
- Servlet

# Conceptos

- HTTP
- Ciclo de vida de una petición HTTP
- Servlet
- Ciclo de vida de un servlet

# ¿Qué es HTTP?

Protocolo de comunicación que permite las transferencias de información en la Internet.

- Orientado a transacciones

# ¿Qué es HTTP?

Protocolo de comunicación que permite las transferencias de información en la Internet.

- Orientado a transacciones
- Esquema cliente servidor

# ¿Qué es HTTP?

Protocolo de comunicación que permite las transferencias de información en la Internet.

- Orientado a transacciones
- Esquema cliente servidor
- El cliente (user agent) realiza una petición enviando un mensaje con cierto formato al servidor.

# ¿Qué es HTTP?

Protocolo de comunicación que permite las transferencias de información en la Internet.

- Orientado a transacciones
- Esquema cliente servidor
- El cliente (user agent) realiza una petición enviando un mensaje con cierto formato al servidor.
- El servidor envía un mensaje de respuesta (información).

# Métodos de petición

- Define una serie predefinida de métodos de petición (algunas veces referido como “verbos”) que pueden utilizarse.

# Métodos de petición

- Define una serie predefinida de métodos de petición (algunas veces referido como “verbos”) que pueden utilizarse.
- Tiene flexibilidad para ir añadiendo nuevos métodos.

# Métodos de petición

- Define una serie predefinida de métodos de petición (algunas veces referido como “verbos”) que pueden utilizarse.
- Tiene flexibilidad para ir añadiendo nuevos métodos.
- Cada método indica la acción que desea que se efectúe sobre el recurso identificado.

# Métodos de petición

## HEAD

Pide una respuesta idéntica a la que correspondería a una petición GET, pero en la respuesta no se devuelve el cuerpo. Esto es útil para poder recuperar los metadatos de los encabezados de respuesta, sin tener que transportar todo el contenido.

# Métodos de petición

## GET

Pide una representación del recurso especificado. Por seguridad no debería ser usado por aplicaciones que causen efectos ya que transmite información a través de la URI agregando parámetros a la URL. La petición puede ser simple, es decir en una línea o compuesta.

# Métodos de petición

Ejemplo:

http GET

[https://upload.wikimedia.org/wikipedia/commons/d/dd/Zkip\\_alibaba1.png](https://upload.wikimedia.org/wikipedia/commons/d/dd/Zkip_alibaba1.png)

obtiene un recurso llamado Zkip\_alibaba1.png

Ejemplo con parámetros:

http GET [http://html.net/tutorials/php/lesson10\\_ex1.php?name=Joe](http://html.net/tutorials/php/lesson10_ex1.php?name=Joe)

http GET [http://html.net/tutorials/php/lesson10\\_ex1.php](http://html.net/tutorials/php/lesson10_ex1.php)

# Métodos de petición

## POST

Envía los datos para que sean procesados por el recurso identificado. Los datos se incluirán en el cuerpo de la petición. Esto puede resultar en la creación de un nuevo recurso o de las actualizaciones de los recursos existentes o ambas cosas. Generalmente es usando para enviar datos a través de formularios.

# Métodos de petición

Existen otros métodos que están dentro del protocolo HTTP, pero no ahondaremos en su funcionamiento, pero si mencionaremos cuales son:

- PUT
- DELETE
- TRACE
- OPTIONS
- CONNECT
- PATCH

# Métodos de petición

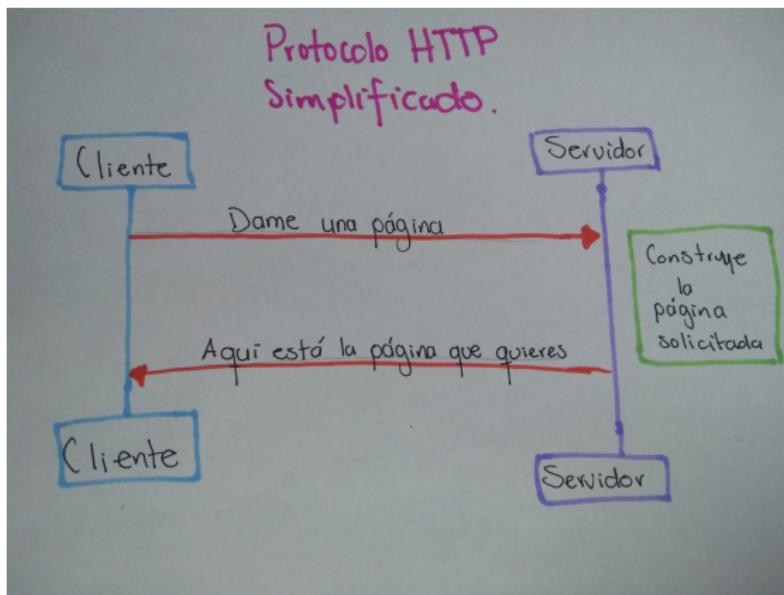


Figura: Protocolo HTTP simplificado

## Ciclo de vida de peticiones a servidores Java

# Ciclo de vida de peticiones JSP

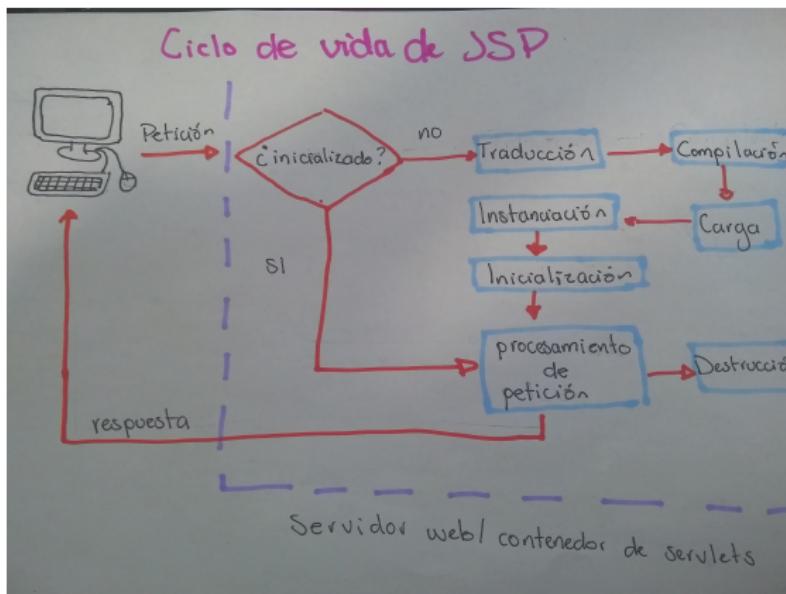


Figura: Ciclo de vida de un JSP

# Ciclo de vida de un Servlet

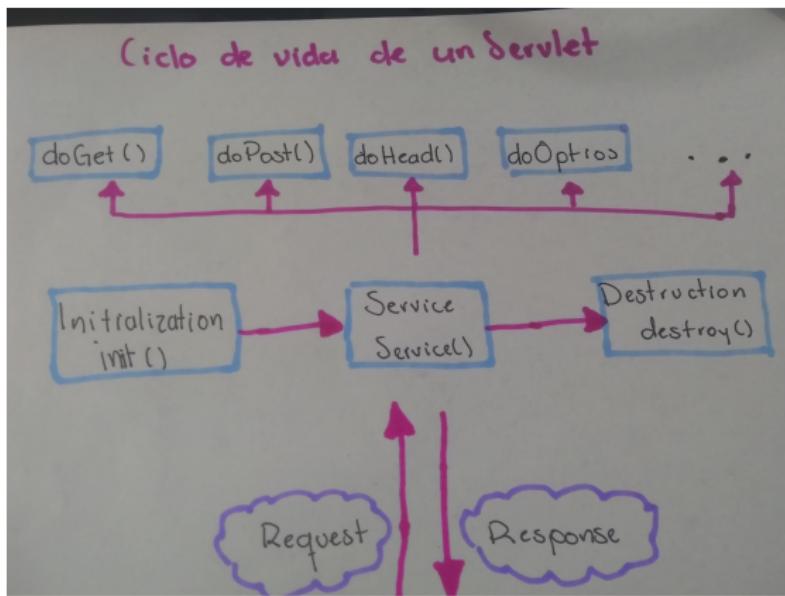


Figura: Ciclo de vida de un Servlet

## Sistemas de control de versiones (Git)

# Sistemas de control de versiones

- Un sistema de control de versiones, es un sistema de software que permite llevar el registro y la gestión de los cambios que se dan sobre los elementos de algún producto.

# Sistemas de control de versiones

- Un sistema de control de versiones, es un sistema de software que permite llevar el registro y la gestión de los cambios que se dan sobre los elementos de algún producto.
- Una versión, revisión o edición de un producto, es el estado en el que se encuentra el mismo en un momento dado de su desarrollo o modificación.

# Sistemas de control de versiones

- Un sistema de control de versiones, es un sistema de software que permite llevar el registro y la gestión de los cambios que se dan sobre los elementos de algún producto.
- Una versión, revisión o edición de un producto, es el estado en el que se encuentra el mismo en un momento dado de su desarrollo o modificación.
- El control de versiones se realiza principalmente dentro del desarrollo de software para controlar las distintas versiones del código fuente dando lugar a los sistemas de control de código fuente.

# ¿Por qué necesito uno?

- Es necesario llevar un control de los distintos tipos de cambios que se realizan sobre el código que implementamos.

# ¿Por qué necesito uno?

- Es necesario llevar un control de los distintos tipos de cambios que se realizan sobre el código que implementamos.
- Además, si estamos trabajando de forma colaborativa, también es necesario llevar un registro de quien o quienes implementaron alguna características, el por qué se implementó de cierto modo, etc.

## ¿Por qué necesito uno?

- Es necesario llevar un control de los distintos tipos de cambios que se realizan sobre el código que implementamos.
- Además, si estamos trabajando de forma colaborativa, también es necesario llevar un registro de quien o quienes implementaron alguna características, el por qué se implementó de cierto modo, etc.
- Además permite llevar el código a un estado anterior sin perder la información más reciente.

# ¿Qué Sistemas de Control de Versiones existen?

Los sistemas de control de versiones más populares en este momento son:

- Subversión (Svn)
- git
- Mercurial

En este curso vamos a utilizar git como **SCV**.

# ¿Cómo obtener Git?

La última versión<sup>1</sup> de **Git** se puede obtener desde la página:

<https://git-scm.com/>

---

<sup>1</sup>Se recomienda que todos los integrantes del equipo tengan la misma versión instalada.

# ¿Qué es Git?

- GIT es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente.

# ¿Qué es Git?

- GIT es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente.
- Git surge como alternativa a BitKeeper, un control de versiones privativo que usaba en ese entonces para el kernel.

# ¿Qué es Git?

- GIT es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente.
- Git surge como alternativa a BitKeeper, un control de versiones privativo que usaba en ese entonces para el kernel.
- Es liberado bajo una licencia GNU GPLv2 y se ha convertido en uno de los más usados alrededor del mundo.

## Usando Git

# Inicializando Git

Para usar git, hay que crear una carpeta donde vamos a llevar el control de versiones de nuestro código, vamos a hacer lo siguiente:

```
$ mkdir mi-proyecto      # Creamos una carpeta para el proyecto  
$ cd mi-proyecto        # Nos movemos a esa carpeta  
$ git init                # Inicializamos el repositorio
```

# Checando el estado del proyecto

La forma en que inicializamos nuestro proyecto con gil, es a través de la instrucción `git init`. Una vez inicializado el proyecto, procedemos a revisar el estado del repositorio con `git status`.

```
$ git status # Checa el estado de los archivos en el repositorio
```

# Agregando archivos y haciendo commit

He creado un archivo llamado README.org que va a contener la descripción del proyecto. Por ejemplo, tendría algo como lo siguiente:

```
#+title: Mi proyecto
#+author: Miguel Piña
#+date: [2017-02-19 dom 19:36]
```

Es es un ejemplo de descripción para mi primer proyecto en git. Seguro contendrá el modo de uso y un proyecto en maven.

- \* ¿Cómo compilar el proyecto?
  - \*\* Como configurar la base de datos para que compile el proyecto
  - \*\* ¿Qué dependencias son necesarias?
- \* Estilo de codificación
- \* Gotchas

# Agregando archivos y haciendo commit

Para agregar este primer archivo a nuestro proyecto, vamos a realizar lo siguiente:

```
$ git add README.org  
$ git status
```

La forma de hacer el commit de una modificación y que quede registrada es con la instrucción `git commit`. Existe una variante en la que se puede agregar un mensaje directamente sin tener que abrir el editor de texto: `git commit -m 'Agregando el README'`, pero esta no la vamos utilizar en este momento.

# Agregando archivos y haciendo commit

Ejecutamos el comando:

```
$ git commit
```

Los pasos anteriores se puede repetir tantas veces como sea necesario. Existen herramientas que facilitan estos pasos en Netbeans como en Emacs. La primera viene integrada dentro de Netbeans y la segunda como un plugin llamado magit.

# Viendo el historial de cambios

Para ver la lista de cambios que tiene un proyecto dado, es posible usar la siguiente instrucción:

```
$ git log
```

También se puede agregar una variante para que sea posible verla en forma de grafo dentro de la terminal:

```
$ git log --graph --decorate --pretty=oneline --abbrev-commit
```

La sentencia anterior se puede agregar a un alias dentro de un archivo de configuración `/.gitconfig`, a algo como:

```
[alias]
tree = log --graph --decorate --pretty=oneline --abbrev-commit
```

Y se puede invocar como:

```
$ git tree
```

# Agregando repositorios remotos

Hay que crear un repositorio remoto (en *github*) y agregamos la siguiente instrucción:

```
$ git remote add origin https://github.com/miguelpinia/something.git
```

Hay que remplazar la url por la del repositorio. Una vez agregado el repositorio remoto, hay que empujar todos los cambios que hemos hecho en nuestro repositorio local:

```
git push -u origin master
```

# Obteniendo cambios desde repositorio

Supongamos que alguien hizo cambios en el repositorio y los empujó al repositorio, para obtener esos cambios, ejecutamos la instrucción:

```
$ git pull
```

Para ver cuales fueron los cambios que se hicieron entre el último commit que teníamos y el que obtuvimos ejecutamos:

```
$ git diff HEAD
```

# Creando ramas

Para agregar cambios sin modificar el contenido de la rama principal (rama master), vamos a crear una nueva rama donde podamos trabajar. Para hacer esto, realizamos lo siguiente:

```
$ git checkout -b mi_nueva_rama
```

A partir de aquí podemos realizar nuevos cambios. Para regresar o moverse entre ramas, usando el comando checkout sin banderas.

```
$ git checkout master
```

## Mezclando ramas

Después de realizar una serie de cambios, es necesario que se mezclen con la rama principal. Una forma limpia de hacerlo, es haciendo un rebase de la rama con la que estamos trabajando respecto a la que vamos a mezclar.

# Mezclando ramas

La acción de rebase, se refiere a desplazar todos los cambios que existan en la rama a mezclar y que no se han integrado con la rama que estamos trabajando.

```
$ git checkout mi_nueva_rama  
$ git rebase master  
$ git checkout master  
$ git merge --no-ff mi_nueva_rama  
$ git push
```

# Eliminando ramas

La forma sencilla de eliminar ramas dentro de git es la siguiente:

```
$ git branch -d mi_nueva_rama
```