Técnicas para mantener estado entre páginas con PHP

Buenos Aires, Septiembre de 2007
Román A. Mussi
romanmussi@gmail.com

1. Alcance de variables en aplicaciones de escritorio y en aplicaciones web

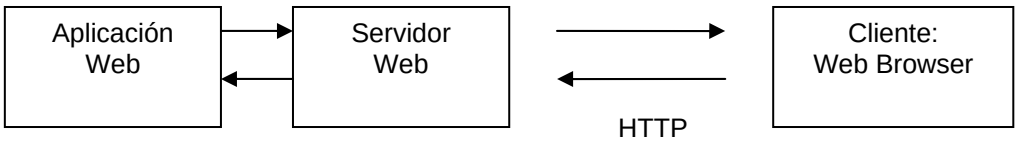
En el presente artículo se aborda la cuestión del ámbito (o alcance) de variables en aplicaciones web, los inconvenientes que los protocolos sin estado introducen en su desarrollo, y las posibles formas de resolver estos problemas utilizando PHP.

Comencemos con un poco de historia (muy simplificada) para que el problema quede claro. Hace unos años era posible crear una aplicación de escritorio, inicializar variables en el módulo principal del programa, y luego consultar y modificar estas variables desde cualquier procedimiento de la aplicación. En algunos casos a lo sumo era necesario declarar a estas variables como “globales” para que se encuentren siempre en alcance. Por cierto, esta facilidad para el uso de variables perennes condujo al desarrollo de programas con alto acoplamiento y problemas de mantenimiento por lo que rápidamente fueron desaconsejadas, pero esa ya es otra historia.

En ese ambiente de desarrollo aún sin llegar a utilizar variables globales se podían utilizar variables locales que sin ser tan persistentes tenían un ciclo de vida de mediano alcance. En realidad unos de los problemas con que el programador se encontraba era el de cómo eliminar de memoria aquellas variables que se habían dejado de usar y consumían innecesariamente recursos; y es en parte por esto que aparecen en algunos lenguajes los “recolectores de basura” (garbage collector).

Lo destacable es que en ese contexto el programador podía decidir cúal sería el ciclo de vida de una variable sin que esto le implicara un esfuerzo adicional de codificación, y más bien tenía que cuidarse de que las variables no queden “vivas” luego de que habían dejado de ser utilizadas.

En el ecosistema Web esta lógica de programación cambia por completo. Ahora la más simple aplicación presenta un esquema de mayor complejidad. En principio podemos observar que el núcleo de la aplicación y la interfaz de usuario se desacoplan: el núcleo de la aplicación se ejecuta en el servidor, la interfaz de usuario se despliega en el navegador web en la computadora del usuario, y el transporte entre cliente y servidor se realiza a través del protocolo HTTP. Esto significa que una aplicación web involucra una variedad de tecnologías complejas: servidores web, lenguajes de scripting, protocolos de transporte, navegadores, etcétera. Y la aplicación web “vive”, para decirlo de algún modo, en ambos espacios del ecosistema: por momentos como una aplicación más tradicional en el servidor web, pero por momentos también interactuando con el usuario a través del browser en la PC del cliente.



En términos secuenciales podemos describir el ciclo de interacción en una aplicación web sencilla del siguiente modo:

1) El usuario apunta su navegador a una dirección web que representa el punto de acceso a una aplicación.
2) El servidor web recibe la petición e invoca a la aplicación web, pasando toda la información recibida desde el cliente. La aplicación web se ejecuta, procesa la información, y genera como salida una página HTML que es devuelta al cliente a través del servidor web.

3) El cliente recibe la página HTML y la despliega en el navegador web. El usuario interactua con esa página (consulta información, completa campos de texto, etc.). Una vez finalizada la interacción el usuario realiza una nueva petición a la aplicación web, si es necesario enviando información mediante distintos métodos para que sea procesada por la aplicación web.
4) Se repiten los pasos 2 y 3, hasta que se cierra la aplicación.

El problema de este ciclo es que HTTP, y por tanto la Web en general, es un protocolo "sin estado" (en inglés: "stateless"). Ello significa que cada petición enviada a un servidor web es independiente de cualquier otra petición. Cuando un cliente solicita una página al servidor, el servidor entrega la página y a continuación olvida todo sobre la petición y el cliente. Cuando el mismo cliente un momento más tarde vuelve a hacer una nueva petición, el servidor no tiene forma de identificarlo, no tiene forma de decir: "a sí, Ud. es fulanito y está consultando sobre naranjas, y hace un momento me dijo que quería 25 con el 10% de descuento". No puede hacerlo porque ningún dato sobre el cliente se guarda en el servidor entre cada solicitud.

La desventaja de esta característica es que limita la complejidad de las aplicaciones web a páginas individuales desconectadas. Para una aplicación web esto significa que si nosotros ingresamos datos en un formulario, esa información no va a estar disponible automáticamente para otras páginas de la aplicación. Obviamente si queremos construir una aplicación web compleja necesitamos superar esta limitación. En otras palabras, necesitamos idear mecanismos que permitan persistir información entre páginas web, necesitamos utilizar alguna clase de "pegamento" para conectar esas páginas y utilizarlas como una sola aplicación.

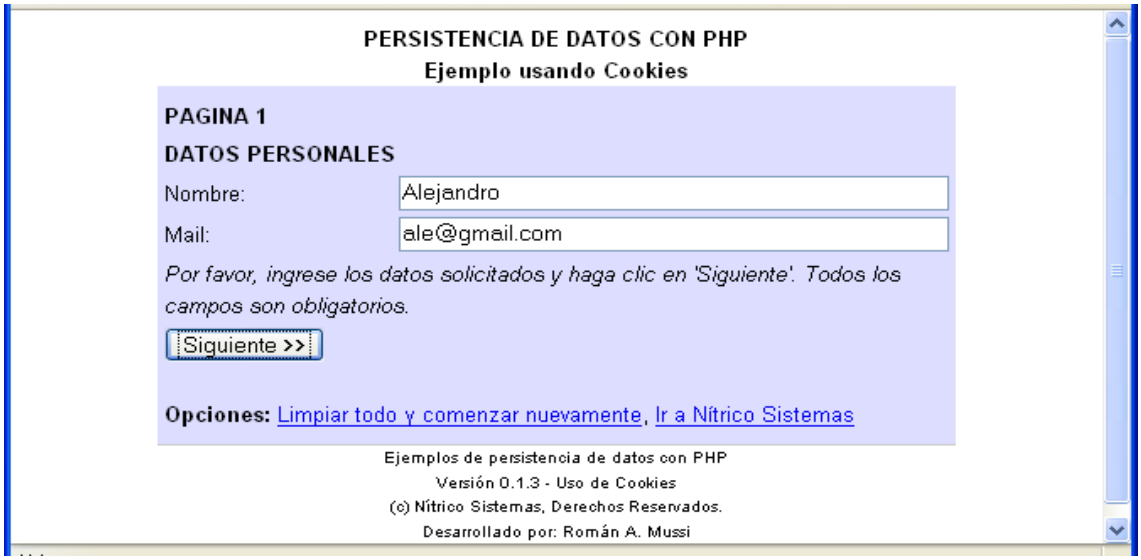
Históricamente varios métodos han sido empleados en programación web para lograr este efecto, y PHP ha agregado algunos mecanismos adicionales. Los principales son:

- Uso de Cookies,
- Uso de información embebida en URIs (método GET),
- Uso de campos ocultos en un formulario (método POST),
- Uso de Sesiones (propio de PHP).

Para algunos autores el mejor modo de mantener estado entre páginas es utilizando la habilidad para administrar sesiones que PHP ha incorporado a partir de su versión 4. No obstante, en el resto del artículo vamos a analizar cada uno de estos métodos para que cada lector pueda sacar sus propias conclusiones.

Para demostrar la forma en que los métodos pueden ser implementados se desarrollará en cada caso un ejemplo de aplicación muy sencilla de 3 páginas:

- Página 1: Mediante un formulario se solicita el ingreso de algunos datos (nombre y mail).



- Página 2: Se muestran los datos ingresados en página 1, y se solicitan nuevos (sexo, edad y hobbie o deporte favorito).

- Página 3: Se muestran los datos ingresados en página 1 y página 2. Los datos ingresados en la página 1 que se visualizan en la página 3 son persistidos según el método que corresponda en cada implementación.

Todos los ejemplos que se mencionan en el artículo se pueden descargar en el sitio ***www.nitrico.com.ar***

2. Uso de Cookies

La información puede ser persistida entre páginas mediante cookies. Las cookies son simples cadenas de texto, asociadas a un dominio, que se guardan en la computadora del cliente. Cuando un cliente solicita una página web las cookies correspondientes al dominio de la página son pasadas al servidor en el encabezado de la petición HTTP. A través de cookies, entonces, una aplicación web que se ejecuta en el servidor puede identificar un usuario en particular y obtener y guardar información asociada al mismo.

Las cookies son útiles para guardar pequeñas cantidades de información en el cliente. Asimismo, las cookies persistentes (aquellas que pueden ser almacenadas por mucho tiempo, incluso años) pueden ser usadas para personalizar una página cuando un usuario ingresa a un sitio web luego de un tiempo.

Algunos autores recomiendan el uso de cookies para aquellos casos en que es necesario almacenar una única información por usuario, pero para aquellas circunstancias en que se hace necesario seguir la pista de un conjunto de información recomiendan el uso de sesiones porque el manejo de múltiples cookies se hace engorroso.

Las cookies ya forman parte de la caja de herramientas que tiene un programador para el desarrollo web. Sin embargo tienen algunas limitaciones, entre las que se destacan:

- Los navegadores sólo deben guardar hasta un total de 300 cookies.
- Los navegadores sólo pueden guardar 20 cookies por cada dominio.
- Una cookie no puede contener más de 4K de datos.
- Los clientes pueden desactivar el uso de cookies en sus navegadores.

Por todas estas razones se recomienda generalmente limitar el uso de cookies.

PHP nos permite guardar y recuperar cookies fácilmente. El script “ejCookies.php” implementa la mini aplicación de ejemplo usando cookies para persistir los datos. A continuación se mencionan algunas cuestiones a tener en cuenta cuando se usan cookies.

Envío de cookies al cliente

En PHP, para almacenar datos y variables en cookies se utiliza la función setcookie(). Por ejemplo, el siguiente código envía al cliente dos cookies denominadas “myname” y “mymail”:

```
setcookie('myname', 'Alejandro');
setcookie('mymail', 'ale@gmail.com');
```

Como se mencionó anteriormente, las cookies son enviadas al cliente web en el encabezado HTTP de la salida que genera PHP.

La función setcookie(), como otras que afectan el encabezado HTTP, debe ubicarse al principio del script PHP, antes de que el programa envíe cualquier otra cosa. En algunos casos esto puede resultar molesto. Puede ocurrir que comencemos a generar la salida HTML y luego decidamos cuáles cookies enviar y con qué valores. Si uno inserta un setcookie() en el cuerpo del script luego de haber enviado algo de HTML se producirá un error.

Para solucionar este problema se puede utilizar ob_start() para activar el almacenamiento en buffer. A partir de su llamada todas las salidas generadas por el script se almacenan en buffer y no son enviadas al cliente. Para terminar y enviar la información al usuario se utiliza ob_end_flush(). Estas funciones permiten utilizar setcookie() en cualquier lugar del script PHP en tanto la información que ha sido almacenada en buffer se envía al cliente en el orden que corresponde (las cookies en la cabecera HTTP, por ejemplo).

Lectura de cookies

Para leer las cookies enviadas por el cliente se puede utilizar el array asociativo $_COOKIE (existen alternativas, pero esto es lo recomendado). En el siguiente ejemplo se evalúa la existencia de las cookies “myname” y “mymail”, y cuando existen se cargan sus valores en variables.

```
if ($_COOKIE['myname']) {
  $nombre = $_COOKIE['myname'];
}
if ($_COOKIE['mymail']) {
  $correo = $_COOKIE['mymail'];
}
```

Una última aclaración para evitar un error común: las cookies no se encuentran disponibles inmediatamente después del llamado a setcookie(). Por ejemplo, veamos el siguiente código:

```
$nombre = 'Prueba';
setcookie('myname', 'Alejandro');
if ($_COOKIE['myname']) {
  $nombre = $_COOKIE['myname'];
}
```

Luego de su ejecución la variable $nombre tiene valor “Prueba”, y no “Alejandro”, porque setcookie() no crea inmediatamente la cookie, sino que prepara el envío de la misma en la cabecera HTTP que está generando PHP. El cliente crea la cookie cuando recibe la página web y a partir de ese momento la devuelve en las siguientes peticiones al servidor. Entonces si, a partir del siguiente envío, la cookie va a estar disponible para el script PHP a través de $_COOKIE.

3. Uso de información embebida en URIs

La información también puede ser persistida entre páginas utilizando el método GET. Este método permite pasar una pequeña cantidad de información al servidor en forma de pares atributo-valor añadidos al final del URI detrás de un símbolo de interrogación “?”.

Un ejemplo de URI con información agregada es:

```
http://www.midominio.com.ar/prueba.php?var1=abc&var2=123
```

En este caso las variables “var1” y “var2” son enviadas al servidor con sus valores y pueden ser accedidas desde un script PHP mediante la variable predefinida $_GET (un array asociativo que permite acceder a las variables del ejemplo mediante: $_GET[‘var1’] y $_GET[‘var2’]).

Existen varias técnicas para enviar la información al servidor mediante el método GET. En primer lugar, se puede usar un formulario HTML y configurar METHOD a GET. En el presente artículo no vamos a analizar esta técnica dado que en el apartado siguiente se muestra como se pueden utilizar formularios HTML con método POST, lo que para persistencia de datos es más potente y de uso más habitual en la programación web. En segundo lugar, se puede escribir la URI a mano en el navegador. Tampoco analizaremos aquí esta técnica dado que al ser de uso manual no permite la automatización y el desarrollo de aplicaciones.

Por último, se puede crear mediante programación links en páginas HTML con la información agregada en la URI. Esta es una técnica de uso común en la programación web. Esta técnica puede ser utilizada para llamar mediante links a determinados módulos de una aplicación web pasando variables con opciones simples. Por ejemplo:

```
/main.php?reiniciar=si
```

llama al módulo main.php y le pasa la variable “reiniciar” con valor “si”.

Otro ejemplo de uso común puede ser el de una grilla o listado de clientes, con sus correspondientes links para ver, editar ó borrar cada uno de los registros (se llama a un módulo y se le pasan dos variables: el modod de ejecución y el id del cliente a procesar).

Cliente A (id: 5)	Ver	Editar	Borrar
Cliente B (id: 17)	Ver	Editar	Borrar
Cliente C (id: 22)	Ver	Editar	Borrar

En el ejemplo, cada Ver, Editar y Borrar del “Cliente B” (cuyo identificador es 17) puede apuntar a algo similar a lo siguiente:

`/clientes.php?modo=vertodo&idcliente=17` (para ver toda la información)
`/clientes.php?modo=editar&idcliente=17` (para editar el cliente)
`/clientes.php?modo=borrar&idcliente=17` (para borrar el cliente)

El uso de esta técnica es útil para pasar al servidor información sencilla y que no es modificada por el usuario. Su uso evita la creación innecesaria de formularios HTML para enviar información al servidor y por ello hace más flexible la programación. Y sin dudas, el envío de información embebida en URIs convive perfectamente con las técnicas de envío de información que utilizan el método POST en tanto son técnicas complementarias.

Por cierto su uso también tiene algunas limitaciones que es importante destacar:

- En principio a esta técnica le caben las recomendaciones generales sobre uso de GET: se recomienda su uso para operaciones de consulta o búsqueda básica, pero no para actualizar o modificar datos de la aplicación ó para enviar mails (para eso se recomiendo POST).
- La longitud de la petición GET es limitada, por lo que para mandar una gran cantidad de información al servidor ha de utilizarse necesariamente el método POST, más aún cuando la información es encriptada.
- La información de la URI es visible en la barra de direcciones del navegador, por lo que no debe ser información reservada. Además, puede ser modificada por el usuario tanto cambiando manualmente la petición en el navegador como a través de programación.
- Finalmente, la técnica no permite enviar al servidor datos ingresados por el usuario (en campos de texto, por ejemplo), lo que limita de manera definitiva su uso para persistir datos entre páginas (recordemos que no estamos hablando en este apartado de uso de formularios con método GET, como aclaramos *ut supra*).

PHP nos permite manipular este tipo de envío y recepción de información con sencillez. Cabe aclarar que no se ha desarrollado la aplicación de ejemplo para esta técnica dado que como se menciona en el último punto de las “limitaciones” la misma no permite el envío al servidor de información ingresada por el usuario.

4. Uso de campos ocultos en un formulario

Otro modo de pasar información entre páginas es mediante el método POST, utilizando formularios HTML con campos ocultos para alojar los datos. Cuando se utiliza el método POST los pares nombre-valor de cada elemento de un formulario son enviados al servidor en el HTTP header.

Un ejemplo de formulario con campos ocultos es:

```
<form name="form1" method="POST" action="prueba.php">
  <input name="var1" type="hidden" value="abc">
  <input name="var2" type="hidden" value="123">
  <input type="submit" name="Enviar" value="Enviar">
</form>
```

En este caso las variables “var1” y “var2” son enviadas al servidor con sus valores y pueden ser accedidas desde un script PHP mediante la variable predefinida $_POST (un array asociativo que permite acceder a las variables del ejemplo mediante: $_POST[‘var1’] y $_POST[‘var2’]).

Los campos ocultos se pueden utilizan para mantener estado entre páginas del lado del cliente. Para ello se debe cumplir con los siguientes requisitos:

- Las páginas deben tener formularios con campos ocultos que contengan los datos que se desean persistir.
- Los valores deben ser pasados utilizando los botones Submit del formulario.

Como ventajas de su uso podemos destacar:

- Con esta técnica se puede persistir más información que si se utilizan cookies ó el método GET.
- La técnica es buena para persistir información sin sobrecargar al servidor dado que la información se mantiene del lado del cliente. En casos de mucha concurrencia o de servidores compartidos con recursos limitados esto puede ser una ventaja.
- La técnica se pueden utilizar aún cuando el uso de cookies se encuentre deshabilitado en los navegadores, lo que da independencia de la configuración del cliente.

Las principales desventajas son:

- Cuando las variables a persistir son muchas la programación de campos ocultos en los formularios HTML se torna engorrosa.
- La técnica requiere un formulario en cada página de la aplicación con sus correspondientes campos ocultos (y en algunos casos el uso de javascript para facilitar la navegabilidad de las páginas sin que se pierda la información), lo que puede introducir cierta rigidez en el diseño. Por ello lo recomendable es utilizar esta técnica de persistencia de modo localizado, combinada con otras técnicas alternativas (uso de sesiones, uso de bases de datos).

PHP nos permite manipular este tipo de envío y recepción de información con sencillez. El script “ejPost.php” implementa la mini aplicación de ejemplo usando campos ocultos en formularios para persistir los datos. A continuación se mencionan algunas cuestiones a tener en cuenta para el uso de esta técnica.

Envío de variables al cliente

Supongamos que se desea persistir las variables “nombre” y “correo”. Entonces, lo que se debe hacer es incluir esas variables en campos ocultos de formulario en la salida que se genera con PHP. Puede ocurrir tanto que los campos ocultos compartan el formulario con campos visibles que se utilizan para tomar nuevos datos, como que en el formulario sólo se utilicen campos ocultos para persistir datos. En el siguiente ejemplo se envían las dos variables (solo se muestra la parte del código que genera el formulario). Nótese que en el formulario además se está solicitando el ingreso de nueva información.

```
$nombre = 'Alejandro'
$correo = 'ale@gmail.com'
echo "
<form method=post action=$_SERVER[PHP_SELF]>
<p>Hobbie ó deporte favorito:
<input type=text name=newhobbie value=''></p>
<input type=hidden name=myname value='$nombre'>
<input type=hidden name=mymail value='$correo'>
<p><input type=submit value='Siguiente >>'></p>
</form>";
```

Lectura de variables

Como se mencionó al principio, para leer los valores enviados por el cliente se puede utilizar el array asociativo $_POST. Siguiendo con el ejemplo del apartado anterior, se puede incluir el

siguiente código al principio de un script PHP para evaluar la existencia de las variables “myname” y “mymail”, y cuando existen tomar sus valores. De esta manera se recrean las variables enviadas el formulario mediante campos ocultos.

```
if ($_POST['myname']) {
  $nombre = $_POST['myname'];
}
if ($_POST['mymail']) {
  $correo = $_POST['mymail'];
}
```

El desarrollador debe tener en cuenta que para PHP (en realidad para POST en general) no hay forma de saber si los pares nombre-valor provienen de campos ocultos ó visibles (campos de texto, comboboxes, u otros). Por eso es recomendable utilizar algunos prefijos en los nombres de los campos que nos permitan rápidamente identificar si lo que recibimos es nueva información, ó un campo oculto usado para persistencia. En nuestro ejemplo utilizamos el prefijo “new” para cada nombre de campo que contiene ingresos del usuario, y el prefijo “my” para los campos ocultos que se usan para persistir datos (obviamente cada programador puede utilizar su propia “hungarian notation”).

5. Uso de Sesiones

Para muchos autores el mejor modo de mantener estado en una aplicación web con PHP es usando sesiones. La administración de sesiones fue incorporada a partir la versión 4 de PHP y nos permite tratar distintas peticiones de un usuario como un todo unificado de manera automática.

En líneas generales las sesiones funcionan del siguiente modo: La primera vez que un usuario llama un script de PHP que utiliza sesiones el interprete detecta que no tiene una sesión activa dado que no recibe un identificador de sesión (“session id”). Entonces crea un nuevo “session id” para el usuario e inicializa un array $_SESSION vacío. Allí se van a almacenar las variables que el usuario quiera preservar. A continuación PHP procesa el script, y, al terminar, guarda la información de $_SESSION en un archivo de texto en el servidor web.

Cada usuario recibe un identificador de sesión diferente, lo que le permite a PHP mantener pilas de datos separadas para cada cliente web. Para que todo funcione PHP se encarga de propagar el “session id” hacia el cliente mediante Cookies ó utilizando parámetros en la URL. Las Cookies son la mejor opción, pero como no funcionan en todos los clientes (porque pueden estar deshabilitadas) el intérprete también puede incrustar el "session id" directamente en las URLs.

En los siguientes accesos de un usuario a un script de PHP el cliente web envía el “session id” al servidor. PHP ve el identificador y carga la información guardada en el servidor para ese usuario en el array $_SESSION, que queda así disponible para el nuevo procesamiento. El array $_SESSION se comporta como cualquier otro array de PHP, por lo que se puede leer, modificar y agregar información con los operadores normales. La ventaja de $_SESSION es que sus datos permanecen en alcance a través de varias peticiones.

Como ventajas del uso de sesiones se puede mencionar:

- Toda la persistencia de datos entre peticiones se encuentra automatizada por lo que simplifica mucho la programación. El almacenamiento de la información persistida se realiza en un array normal de PHP ($_SESSION), por lo que el acceso a los datos resulta sencillo y natural.
- Se puede persistir una mayor cantidad de información que con cookies ó el método GET.
- Las sesiones se pueden utilizar aún cuando el uso de cookies se encuentre deshabilitado en los navegadores, lo que da mayor independencia de la configuración del cliente.

- Las sesiones no permiten al usuario la modificación de la información que se envía al servidor (como ocurre con el envío de información en URLs con método GET).
- Los datos de sesión no están restringidos a cadenas o números como en el caso de las cookies (también se pueden guardar arrays, por ejemplo).

Una desventaja de esta técnica es que exige más al servidor que otras analizadas (como por ejemplo el uso de POST con formularios y campos ocultos). Esto es así porque el interprete PHP debe reservar memoria para mantener los arrays $_SESSION de cada cliente mientras se está procesando un script, y luego debe grabar y leer en disco para almacenar y recuperar esa información del servidor. Todo esto implica un costo; en casos de mucha concurrencia o de servidores compartidos con recursos limitados esto puede resultar inconveniente.

En PHP administrar sesiones resulta prácticamente transparente para el programador. El script “ejSesiones.php” implementa la mini aplicación de ejemplo usando sesiones para persistir los datos. A continuación se ofrecen algunos ejemplos sencillos de uso.

Lo primero es activar el uso de sesiones en PHP. Esto se consigue mediante la configuración de session.auto_start configurado en ON (en php.ini), ó en el script utilizando session_register() (función que implícitamente invoca la administración de sesiones). Lo que recomendamos es utilizar explícitamente session_start() al principio del script: es independiente de la configuración de PHP en el servidor, y más ordenado que la invocación implícita.

Una vez que el uso de sesiones ha sido activado es posible utilizar $_SESSION como un array que está siempre disponible de manera transparente. Por ejemplo, para crear una variable y asignarle un valor:

```
$_SESSION['myname'] = 'Alejandro';
```

También se pueden realizar evaluaciones en base a su valor:

```
if ($_SESSION['myname'] = 'Alejandro') {
  echo “Hola Alejandro”;
}
```

O para almacenar los ingresos del usuario de manera persistente:

```
if ($_POST['newname']) {
   //carga el valor ingresado por el usuario en
   //una variable persistente
   $_SESSION['myname'] = $_POST['newname'];
}
```

A diferencia de las cookies, que no están disponibles de manera inmediata a su declaración, $_SESSION se encuentra siempre en alcance a partir del momento en que se invoca la administración de sesiones en PHP, lo que hace muy sencilla y natural su utilización.

6. Conclusiones

En este artículo se ha pasado revista a los distintos métodos que nos permiten mantener estado entre páginas con PHP, y se han mencionado pros y contras de cada uno. Lo cierto es que muchas veces los métodos conviven en la misma aplicación. En realidad podemos utilizar uno u otro según el contexto y el problema específico a resolver. Lo importante es conocer las alternativas para poder seleccionar lo que mejor se adapta a nuestras necesidades.

Para finalizar, una pequeña historia personal. Hace un tiempo desarrollé para el sitio de un cliente una encuesta on line relativamente compleja: el usuario tenía que pasar por diferentes páginas seleccionando y cargando información hasta llegar a la página final en la que se podían guardar los datos; la aplicación buscaba información en una base de datos de acuerdo a las selecciones que el usuario iba realizando, etc. Como antes de llegar a guardar los datos

en la última página el usuario tenía que pasar por 4 ó 5 páginas anteriores había que mantener estado entre páginas. O sea, era un caso típico del tema que analizamos en éste artículo. Pues bien, desarrollé la primera versión del programa tal como indican los manuales: usando sesiones de PHP. El problema fue que la aplicación no funcionó como esperaba: detrás de proxys ó en momentos de mucha concurrencia la aplicación se comportaba de manera errática (la sesión se perdía y el usuario volvía al principio luego de haber pasado ya por 3 ó 4 páginas de selecciones), lo cuál era muy desalentador para quién estaba completando la encuesta. La verdad, nunca pude analizar detenidamente qué es lo que estaba ocurriendo con las sesiones que fallaban: por falta de tiempo, y porque el sitio está en un hosting contratado por lo que no puedo acceder al servidor (importante para hacer un análisis detallado y en profundidad). Como la aplicación tenía que seguir funcionando sin pérdida de tiempo la rediseñe utilizando la técnica de persistencia mediante POST y campos ocultos en formularios, y funcionó perfecto en todos los contextos y con altísimos niveles de concurrencia. La conclusión de esta pequeña historia no es que una técnica es mejor que otra (en realidad, en términos generales el uso de sesiones es lo recomendable), lo importante es ver como el conocimiento de las alternativas disponibles nos permite avanzar en una situación compleja, y en un contexto en el que debemos garantizar el funcionamiento de una aplicación.