Sessions, Cookies and Files

The server can remember you — learn sessions that live on the server, cookies that live in the browser, and reading and writing files on disk.

Introduction

HTTP forgets everything. Every request is a stranger: the server does not remember that you logged in two seconds ago, or that you picked dark mode yesterday. **Sessions** and **cookies** are PHP's two kinds of memory, and they work as a pair.

A **cookie** is a small note the server asks the browser to keep — a name and a value, with an expiry date. The browser sends it back with every request. Cookies are good for small preferences, like a theme choice.

A **session** is a locker on the server. PHP gives the visitor a random session ID (kept in a cookie), and everything you store in `$_SESSION` lives in that locker on the server — safe from the visitor's eyes. Sessions are good for login state and anything private.

Files are the third kind of memory: plain text on the disk. Before you meet databases in PH.11, files are the simplest way to save data that must survive after the page ends — a guestbook entry, a counter, a log line.

(Roman Urdu: Cookie browser mein rehti hai (chhoti yaad), session server par rehta hai (mehfooz locker), file disk par rehti hai (mustaqil yaad).)

Key points

- `session_start()` **must run before ANY output** — before even one space or HTML tag — or PHP cannot send the session cookie.
- `$_SESSION["key"] = $value` **stores;** `$_SESSION["key"]` **reads.** The data lives on the server, keyed by the visitor's session ID.
- `session_destroy()` **logs the visitor out** — it empties the locker; also clear `$_SESSION` itself.
- `setcookie("name", "value", time() + 86400)` asks the browser to keep a cookie for one day (86400 seconds). It must also run before any output.
- **Read a cookie with** `$_COOKIE["name"]` — and always check `isset()` first, because the visitor may have deleted it.
- **A cookie is visible to the visitor and can be changed.** Never store a password or a "is admin" flag in a cookie.
- `file_get_contents("notes.txt")` reads a whole file into a string; `file_put_contents("notes.txt", $data)` writes a whole string.
- `FILE_APPEND` **flag appends instead of overwriting:** `file_put_contents("log.txt", $line, FILE_APPEND)`.

- `fopen` / `fwrite` / `fclose` give line-by-line control; `file_exists($path)` checks before you read.
- **Session data beats cookie data for anything private.** Cookies are for preferences; sessions are for identity.

Examples

Example 1 — A session that counts your visits

```
<?php
session_start();
if (!isset($_SESSION["visits"])) {
    $_SESSION["visits"] = 0;
}
$_SESSION["visits"]++;
echo "You have visited this page " . $_SESSION["visits"] . " times.";
?>
```

What you will get:

```
First visit: You have visited this page 1 times.
Reload the page: You have visited this page 2 times.
```

The number lives on the server in your session locker. Close the browser and the session cookie dies — the count starts again. (That is normal for a default session.)

Example 2 — A tiny login with a session

```
<?php
session_start();
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    if ($_POST["name"] == "Ali" && $_POST["pass"] == "1234") {
        $_SESSION["user"] = "Ali";
    }
}
if (isset($_SESSION["user"])) {
    echo "Welcome back, " . htmlspecialchars($_SESSION["user"]) . "!";
} else {
    echo '<form method="post" action="">
      <input type="text" name="name" placeholder="Name">
      <input type="password" name="pass" placeholder="Password">
      <button>Login</button></form>';
}
?>
```

What you will get:

```
Before login: the form. After login with Ali / 1234: Welcome back, Ali!
Reload the page: still Welcome back, Ali! — the session remembers.
```

The password check here is a teaching demo only — real logins hash passwords (LV.7 in the Laravel series). The point is the memory: once `$_SESSION["user"]` is set, every page can see it.

Example 3 — A cookie that remembers a theme

```
<?php
if (isset($_POST["theme"])) {
    setcookie("theme", $_POST["theme"], time() + 86400 * 30); // 30 days
    $_COOKIE["theme"] = $_POST["theme"]; // use it right away
}
$theme = $_COOKIE["theme"] ?? "light";
echo "Theme: $theme";
?>
<form method="post" action="">
  <button name="theme" value="dark">Dark</button>
  <button name="theme" value="light">Light</button>
</form>
```

What you will get:

```
Click Dark: Theme: dark — and it stays dark on every reload for 30 days.
```

`$_COOKIE["theme"] ?? "light"` reads the cookie or falls back to "light" — the `??` operator means "use this, or that default when it is missing".

Example 4 — Writing and reading a file

```
<?php
file_put_contents("notes.txt", "Ali was here.\n");
echo file_get_contents("notes.txt");
?>
```

What you will get:

```
Ali was here.
```

One line writes, one line reads. Run the page twice and the file still holds one line — `file_put_contents` overwrites by default.

Example 5 — A visitor log that grows

```
<?php
$line = date("Y-m-d H:i") . " — someone visited\n";
file_put_contents("visits.log", $line, FILE_APPEND);
echo "Logged. Lines so far: " . count(file("visits.log"));
?>
```

What you will get:

```
Logged. Lines so far: 1 (then 2, then 3 — one more line per visit.)
```

`FILE_APPEND` adds to the end instead of wiping the file. `file()` reads the whole file into an array, one line per item — so `count` tells you how many visits were logged.

Example 6 — Logging out: destroying the session

```
<?php
session_start();
if (isset($_GET["logout"])) {
    $_SESSION = [];
    session_destroy();
    echo "You are logged out.";
} else {
    $_SESSION["user"] = "Ali";
    echo 'Logged in as Ali. <a href="?logout=1">Logout</a>';
}
?>
```

What you will get:

```
First open: Logged in as Ali. [Logout link]
Click Logout: You are logged out.
```

Clearing `$_SESSION` empties the locker; `session_destroy()` ends the session itself. Do both — one without the other leaves leftovers.

Quick recap

- `session_start()` first, before any output; `$_SESSION` stores on the server.
- `setcookie()` before any output; `$_COOKIE` reads; cookies are visible and changeable.
- Sessions are for identity (login); cookies are for preferences (theme).
- `file_put_contents` writes; `file_get_contents` reads; `FILE_APPEND` grows a file.

- `session_destroy()` + clearing `$_SESSION` logs a user out properly.

Common mistakes

1. **Output before session_start() or setcookie()** — even one space before `<?php` triggers "headers already sent". Fix: `session_start()` must be the very first thing in the file.
2. **Reading a cookie in the same request you set it** — the browser only sends it back on the NEXT request. Fix: set `$_COOKIE["theme"]` yourself too if you need it immediately (see Example 3).
3. **Storing secrets in a cookie** — the visitor can read and edit cookies freely. Fix: keep secrets in `$_SESSION` on the server.
4. **Forgetting FILE_APPEND** — every visit overwrites the log with one line. Fix: pass `FILE_APPEND` when the file must grow.
5. **Reading a file that does not exist** — `file_get_contents` warns and returns false. Fix: check `file_exists($path)` first.
6. **Assuming a session lasts forever** — default sessions die when the browser closes. Fix: for "remember me", use a long-lived cookie with a random token (Laravel's Breeze does this for you).
7. **Writing user input straight into a file** — a visitor can inject lines or paths. Fix: validate first, and never let a filename come from user input unfiltered.

Practice

1. Build the visit counter from Example 1, reload the page 3 times, and write down the count each time.
2. Build the tiny login from Example 2, log in, reload, and confirm the welcome message survives the reload.
3. Add a Logout link to that login that destroys the session — confirm the form returns.
4. Build the theme cookie from Example 3 and check in DevTools (Application → Cookies) that the cookie exists with a 30-day expiry.
5. Write "Hello file" into `test.txt` with `file_put_contents`, then read it back and print it.
6. Build the visitor log from Example 5, visit the page 3 times, then print the whole log with `file_get_contents`.
7. Deliberately put one space before `<?php` in a file that calls `session_start()` — read the error, fix it, and write down in one line why output must come after.

Practice quiz

1. Where must session_start() run?

- Anywhere in the file
- Before ANY output, even one space, or PHP cannot send the session cookie
- After the HTML
- Only on the login page

Answer: Before ANY output, even one space, or PHP cannot send the session cookie

2. What is the difference between a session and a cookie?

- They are the same
- A session lives on the server (safe, for login); a cookie lives in the browser (visible, for preferences)
- A cookie lives on the server, a session in the browser
- Sessions expire in seconds, cookies never expire

Answer: A session lives on the server (safe, for login); a cookie lives in the browser (visible, for preferences)

3. What does file_put_contents("log.txt", $line, FILE_APPEND) do?

- It overwrites the file with one line
- It adds the line to the end of the file, keeping what was there
- It deletes the file
- It reads the file

Answer: It adds the line to the end of the file, keeping what was there

---

Watch the Code Scope channel: https://www.youtube.com/@CodeScopePK — this lesson's own video joins the channel when it is recorded.

Read online: https://scopetreasure.com/notes/ph-sessions-cookies-files · More free tools and notes at https://scopetreasure.com