

COMP10120 Lab Session 6

PHP and HTML forms

Contents

6.1	Objectives	56
6.2	Setting up PHP on your Pi	56
6.2.1	Creating your first PHP script	57
6.3	Task	58
6.4	Process	59
6.5	Knowledge	59
6.6	Assessment	60

These notes are available online at
studentnet.cs.manchester.ac.uk/ugt/COMP10120/labscripts/101lab6.pdf

6.1 Objectives

To make sure that everyone in your tutorial group:

- Has a basic familiarity with PHP, a framework that will allow you to create dynamic content for your group website.
- Has begun to think about security issues with PHP and HTML forms.

6.2 Setting up PHP on your Pi

Once again, you're going to be doing this exercise on your Raspberry Pi, so set it up in the usual way. In this exercise it is probably better to use it 'headless' and log into it from the desktop PC, rather than using the Pi's graphical environment.

PHP is what is known as a **server-side scripting language**^w, which means that it's a mechanism for writing programs that are interpreted *on the server*, and the **HTML results then passed to the web browser that requested a particular page**. You may have also heard of **JavaScript** as another way of creating dynamic or interactive content in web pages; JavaScript is a **client-side scripting**^w language, where the program you've written is executed in the browser itself. Because PHP runs server-side, we need to install the **PHP interpreter**^w in your Apache server before it can make use of any PHP programs you write.

Assuming you have already installed Apache, you'll need to use `apt-get install php5` to fetch and install the PHP interpreter (remember, you'll have to run this command with super-user privileges, and may need to get `apt-get` to update its list of repositories too).



You should now be able to write PHP programs on your Pi. Create a temporary file somewhere called `phptest.php` with the following content:

```
<?php
    echo "Hello world\n";
?>
```

and from the command line execute this file using the newly-installed PHP interpreter by typing:

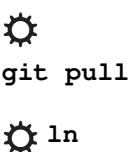
```
$ php ./phptest.php
```

If all has gone to plan you should see `Hello world` echoed on the terminal. There shouldn't be anything very surprising going on here; in the same way that you used the Python interpreter to run the `wormy.py` program back in your first encounter with the Raspberry Pi, all you're doing here is using the PHP interpreter to run your `phptest.php` program.

Unlike Python, which was designed to be a general purpose programming language, and only later on adapted to be used for generating web-content, PHP was intended for web-development from the very start. Although many will argue that PHP is a rather ugly language, because it was designed specifically for creating web-pages, it means that you can get going with it rather more quickly than you could with other systems; you can make your own mind up later as to whether it is clever, awful or somewhere in between when you've learned more about how web technologies work, and how different frameworks have tackled the various issues involved.

6.2.1 Creating your first PHP script

Make sure that you have used `git pull` to bring the COMP10120 repository on your Pi up to date with any changes you've made elsewhere, and then create a new `ex6` directory in the project for this lab's work. Make a symbolic link in `/var/www` called `php`, linking to `~/COMP10120/ex6` using `ln -s` as in the previous session, and check that the file and directory permissions are set so that the web server can see the contents.



In `ex6` create yourself a file called `index.php` containing the minimum amount of HTML necessary for the web server to be able to display its contents, and check that you can see this using a web browser. *It is very important at this stage that you don't open the file in your browser locally, using the `file://` protocol, but via the web server using `http://`.* Also make sure that call this file `index.php` and not `index.html`; why will become clear shortly.

Next, add the following somewhere in the body of your `index.php` file:

```
<?php
    echo "Hello world\n";
?>
```

and refresh your browser to see the effect. If everything has worked as intended, you should now see the text `Hello world` inserted at the appropriate point in your test web page. Notice


that you're not seeing any of the other bits of PHP syntax, but only the result of the `echo` function (if you are seeing things like 'php' and 'echo' appearing in the browser, something has gone wrong, so best to call for help at this stage).

To explain what's going on here: the web server is reading your `.php` file and displaying any bits of HTML as normal; but when it reaches the `<?php` tag, it hands over control to the PHP interpreter, which then treats everything it sees up until the closing `?>` tag as being a PHP program; the PHP instructions are executed, and any output (which in this case comes from the `echo` function) gets included in the stream of HTML seen by the web browser. If this isn't clear to you, ask a member of lab staff for more of an explanation.

Before moving on there's one more bit of configuration to do that will save you a lot of pain in future. By default, the Apache/PHP setup on your Pi assumes that you are using the Pi as a production server (i.e. as something that real users are going to be accessing) rather than for development. Because of this it's configured to hide the output of any errors that may occur when processing PHP from the browser, and instead to put them in a log file on your Pi that you can inspect later; this may be a sensible choice for a finished piece of software, but when you're developing code and making mistakes, it makes debugging your work pretty hard.

To change this default behaviour you're going to need to create a local PHP configuration file to switch the error reporting feature so that it shows errors in the browser. Now, be very careful with the following instructions! To make this change you are going to need to create a file using superuser privileges.

The file in question is `/etc/php5/conf.d/zz-local.ini`. Start the editor of your choice using the `sudo` command to give you superuser access rights, and create a file that contains just the lines

 `sudo`

question

```
display_errors = On
error_reporting = E_ALL
```

Now save this file with the name given above. Once you've done that you need to restart the apache server. To do this use `apachectl` to restart the server. Do this by typing:



`apachectl`

```
$ sudo apachectl restart
```

and remember that `apachectl` ends with a lower-case letter 'l' not the number '1' because it is short for 'Apache Control'. If it complains at this stage that 'Could not reliably determine ...', then don't worry, all should still work.

Any errors that are caused by your PHP code should now be visible in the browser (you'll probably want to change this setting back to 'Off' at some point if you're using your Pi as a production server for anything).

6.3 Task

Create some web-pages that use PHP and HTML forms in a realistic but simple way:

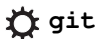
- ✓ Create a page that uses HTML forms to allow users to input their name and see it output on another page as part of a welcome message, using PHP.
- Upgrade your pages to test the name to see if it belongs to one of the members of your group, and output a different message if so.

- Upgrade your pages to accept an email address as well, and check the format of the name and address to avoid possible problems.
- If you have done something like this before, use this as an opportunity to learn more about the available technologies. By the end of this exercise you should be able to demonstrate some web-pages making simple uses of PHP and HTML forms, all in your School file-store.

6.4 Process

You probably need to:

- Find out about PHP, e.g. by working through some or all of an introductory tutorial.
- Find out about using HTML forms with PHP.
- Find out about common security problems with PHP and what to do about them.
- Keep any files you create under version control using `git add` and `git commit`.



If you use any code that isn't your own (e.g. from a tutorial) in your web pages, you should add comments to make it clear what you've used and where you got it from. In COMP10120 labs there is absolutely no problem using code from elsewhere as long as you make sure that you credit the author.

6.5 Knowledge

You should try to understand and make sensible use of the following pieces of PHP and HTML:

- `<?php ... ?>`
- `//` and `/* */`
- `include`
- `echo`, `printf`
- `;` and `{ ... }`
- `if`, `if-else`, `foreach`, `while`, `exit`
- variables, arrays
- `function`
- `<form action=... method=...>`, `<input type=... name=...>`
- `$_POST...`?
- simple string handling e.g. `strcmp`, `strpos`
- simple regular expressions e.g. `preg_match`.

6.6 Assessment

When you have finished, make sure that you have committed your work and pushed it to GitLab, and `submit` the work you've done in `ex6` for marking: in this case `submit` will collect up any `php`, `html` and `css` files from your `ex6` directory; don't worry if there are any other files such as images in there, they are not needed for marking, but you should make sure they are added to your project.



You can view a copy of the marking scheme used by `submit` and `labprint` at studentnet.cs.manchester.ac.uk/ugt/lp/ms.php?unit=COMP10120&ex=ex6.xml