

COMP10120 Lab Session 5

HTML and CSS

Contents

5.1	Objectives	49
5.2	Task	49
5.3	Process	49
5.3.1	Set up your Pi	49
5.3.2	Link to the right place	50
5.3.3	Make things accessible	51
5.3.4	Finally—create some web pages!	52
5.4	Knowledge	53
5.5	Assessment	54
5.6	Taking your understanding further	55

These notes are available online at

studentnet.cs.manchester.ac.uk/ugt/COMP10120/labscripts/101lab5.pdf

For the rest of this semester, you will be exploring various technologies for making web-sites and web-based applications, and starting to design your group project. To complement your work for the tutorials, these labs are an opportunity to pick up relevant practical knowledge and skills by building a simple web-site. The more you do now, the easier you will find creating your real web-site next semester.

Previous labs have largely consisted of us explaining some material to you in a fair amount of detail, and then leading you through some activities. You'll notice that from now on, these sessions have a different style, where we will expect you to find stuff out for yourself. There will still be plenty of pointers and hints in the notes, but it'll be up to you to search for suitable materials online or elsewhere to help you understand the concepts.

If you already know something about building a web-site then you should have slightly different objectives for these labs. We still want you to do these exercises, but you have the opportunity to go beyond the minimum specifications to try out new things, that might be useful next semester. Furthermore, keep in mind that, although all but one of these labs are to be done individually, the majority of the work for this project is a group effort. This is a chance to help get everyone up to speed, so you will all be able to do your fair share of the work next semester. Feel free to help your colleagues find and understand information and debug problems, but remember that, in the end, you want each member of your group to be as skilled as possible, so don't do the work for them!

5.1 Objectives

To make sure that everyone in your tutorial group:

1. Has a basic familiarity with some of the tools and technologies that you could use to build your project web-site.
2. Has started to think about what kinds of content and styles might be appropriate or otherwise.

5.2 Task

You're going to be using your Raspberry Pi to create yourself a personal web-site using HTML and CSS. The site should consist of at least a home page and an implementation page (which we'll describe a bit further down). Although there are plenty of tools such as **Mozilla Composer**[™] or **Adobe Dreamweaver**[®] that will do this task for you, for this exercise you must write the HTML and CSS 'by hand' using a text editor such as nano. You'll also need to put the files you create under version control, and you'll be using git and GitLab to transfer the files between your Raspberry Pi (where you'll be building the site) and the lab PC (from where you'll need to submit your work at the end of the exercise).

You should by now have already created some basic HTML in the previous lab sessions and you're welcome to use any web pages you made in Intro Lab 3 as a starting point; if for whatever reason you didn't manage to make your own pages then, or you don't like what you made, now is the time to start from scratch. If you're already quite familiar with HTML and CSS, use this session to improve on your understanding and to explore new related technologies.

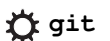
By the end of this exercise you should be able to demonstrate a home page, an implementation page, and any other web-pages you care to create.

5.3 Process

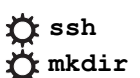
5.3.1 Set up your Pi

You're going to be doing most of this lab's work on your Raspberry Pi, so begin by connecting it up in the usual way, and start up the window manager.

We will be making heavy use of git in the lab for transferring files between desktop and your Pi. Before you can use it, you need to get a copy of the SSH-key from your CS account (you created this in the previous session). This allows git and GitLab to know who you really are, even though on the Pi your username is different. The two files that you need to copy over are `id_rsa` and `id_rsa.pub`, and they will live in a directory called `~/.ssh` in both the Pi and your CS account.



Check on your Pi to see whether a directory called `~/.ssh` already exists (it most likely does, because using `ssh` to log into other machines creates this automatically). If for some reason there's no sign of it, then use `mkdir` to create it, and check that worked using `ls` (remember, you'll need to use a switch to the `ls` command to make it show hidden files that start with a full stop). `ssh` requires specific file permissions on the directory `.ssh`, so if you have just used `mkdir` to create it, now run the command



```
$ chmod 700 ~/.ssh
```

Once you're certain you have a `~/.ssh` directory, you can copy the two files you need over using the `scp` command:



```
$ scp [USERNAME]@[HOSTNAME-OF-A-DESKTOP-PC]:.ssh/id_rsa* ~/.ssh
```

If the syntax of that `scp` command doesn't make sense to you at this stage, please ask a member of lab staff for help.

Next we want to get your COMP10120 project from GitLab onto the Pi. You should have created this in Lab 4; if you didn't do it then, go back and do it now.

Using the Pi's web-browser, log into GitLab and go to your COMP10120 project, and copy the project URL into the last parameter of a `git clone` command (which you should run in a terminal from the home directory of your Pi):

```
$ git clone [COMP10120-URL-GOES-HERE] COMP10120
```

Remember, you're not using the URL of the web page for the project in GitLab, but rather the URL of the git project itself; the former will start with `https://` and isn't the one you want; the latter, which is the one you want, **should start with `ssh://` and end with `.git`**. Note that in the above command **we've given a second argument to the `git clone` command to specify that we want the clone to be created in a directory with name `COMP10120`**. By default GitLab has the somewhat annoying habit of creating clones with all lower-case names, rather than using the name of the original repository.

If at this stage you find that **you don't have the entire contents of your `COMP10120` directory on your Pi, this is almost certainly because you haven't kept the copy of the repo on GitLab up to date. Go back to the desktop machine, run `git status` and then commit and push as necessary; finally re-clone.**

Assuming that's all worked as planned, you should now have a copy of your `COMP10120` directory on your Pi. Use `cd` to check that it's what you expect (you should be able to see your **LaTeX `ex3` and GitLab `ex4` in there, for example**). If it's not what you expect and you don't know why, now is a good time to ask for help.

At this point, **create a new directory inside `COMP10120` called `ex5`**; that's where you're going to be doing the work for this lab.


5.3.2 Link to the right place

The next thing that we need to do is to arrange for the Apache web-server that you installed in the last Pi session to be able to see the pages that you're about to create in the `ex5` directory. If you remember, by default, **Apache will look in a directory called `/var/www` for files to serve**. Now you could at this point **just do your work in the `ex5` directory, and copy the files manually from `ex5` to `/var/www/` every time you change them, but that would be really frustrating**. Or you could do it the other way around, and work in `/var/www/` and then copy stuff back into `ex5` when you're finished. But both of these approaches are fiddly and error prone. On Unix we can do something nicer: we can make a link from within `/var/www/` to your `ex5` directory that essentially allows a directory to appear in two places at once.

Unix filestores make a distinction between a **file name** and a file's **contents**. Files are accessed via their name, which is a **link** to the contents. This enables flexibility, such as the possibility

for a file to be removed (**unlinked**) while a program, which has already opened it, continues to read the contents, or a file to be renamed while a program has access to it via its old name. It even allows a file to have two or more names: i.e. two **links** in separate parts of the file system to the same file contents. Such links between file name and file contents are known as **hard links**. This applies to directories as well as files.

Unix also offers a similar facility, known as a **soft link** or **symbolic link**. This is similar in concept to the idea of a **shortcut** in Windows. A symbolic link is like a little file that contains the path name of another file (which itself may be a symbolic link) or a directory. When a symbolic link is opened, the operating system reads the contents and then opens the file that the link redirects to (and so on if that too is a symbolic link).

In classically terse Unix style, the `ln` command (short for 'link') can be used to create a new link  `ln` to a file: with the `-s` option it creates a symbolic link instead of a hard link.

You want to set things up so that when your web server opens `/var/www/aboutme/index.html` it will actually get the contents of `~/COMP10120/ex5/index.html`. Do this as follows (make sure you use the `-s` option for `ln`).

From a terminal, type:

```
$ ln -s ~/COMP10120/ex5 /var/www/aboutme
```

This will create a **symbolic link** with the name `/var/www/aboutme` that refers to your `ex5` directory in `~/COMP10120`. If you use `ls` to inspect the content of `/var/www`, you should see that there's what appears to be a directory called `aboutme` in there now; this isn't a real directory, it's just another way of getting to `~/COMP10120/ex5`. If you use `ls -l` to look at the content of `/var/www`, you'll see that the entry for `aboutme` actually says `aboutme -> /home/pi/COMP10120/ex5`, so that you can tell the difference. The nice thing about Unix links is that they exist at the filesystem level; so any program that now tries to access `/var/www/aboutme` will automatically get redirected to the contents of `~/COMP10120/ex5` without it having to do anything special.

Now, any files that you create in your `ex5` directory should immediately be visible in the directory `/var/www/aboutme/`, which means that a web browser should be able to see them via a URL such as

```
http://localhost/aboutme/[FILENAME]
```

if you are looking at them on the Pi, or

```
http://[IP-ADDRESS-OF-YOUR-PI]/aboutme/[FILENAME]
```

if you are viewing them from another machine on the CS network. *Note that you should be using the web server on the Pi to view your files using the `http` protocol, not viewing them locally using `file://`.*

5.3.3 Make things accessible

In the Intro labs where you set up Apache on your Pi, we told you that any files that you want the web server to read must have suitable file permissions to enable this. Because Apache runs as a user without any special permissions, any files that it needs to read must be readable by anyone. To do this you should use a command such as

Breakout 5.1: Git-removing clutter



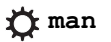
You may have noticed by now that when you use `git status`, it reports the names of many files that are not under git control, such as the `.log` and `.aux` files created by LaTeX, and the backup files created by your favourite editor.

Handily, there's a way of telling git to ignore this sort of file; we do this by creating a file `.gitignore` in the top-level directory of the repository (the one that contains the directory `.git`). This file contains a list of patterns that match names of files we want git to ignore. Details can be found in its `man` page, using `man gitignore`, but an example file might contain the lines

```
*.log
*.aux
*~
```

This would cause git to ignore all files with names ending in `.log`, `.aux` and `~`. You could use this as a starting point for making your own `.gitignore` file. `.gitignore` is just a file, like the other contents of the repository; you should use `git add` and `commit` to include `.gitignore` in your repository.

Beware: if you are tempted to add `*.pdf` to the file in order to ignore all pdf files created by LaTeX, which you probably want to do. Doing so will *also* ignore PDF files not created by LaTeX that contain images that you might want under git control.



```
$ chmod og+r myfile.html
```

Any directory containing such files also needs to be searchable by anyone; to fix this use something like

```
$ chmod og+x mydirectory
```

You also need to do this for any directory *above*, right up to your home directory. So, for example

```
$ chmod og+x COMP10120
```

If you don't get these permissions right you will get errors such as

```
You don't have permission to access /~mbaxy23/ on this server.
```

5.3.4 Finally-create some web pages!

For the next part of this session, you're going to be creating and editing files in `ex5`. You can either start from scratch, or use any previous HTML files that you've created as long as you end up with at least a 'home page' and an 'implementation page' that are linked to one another. You'll need to do at least the following:

1. Find out how to write basic HTML to create the content and structure of a web-page.
2. Find out how to use CSS to modify how the web-page is displayed.

3. Find and use a HTML Validator to confirm that your web page is correct (keep in mind that web pages served from your Raspberry Pi can only be seen from within the School of Computer Science when you do this bit!).

If you are already familiar with HTML and CSS explore some of the concepts listed in Section 5.6.

Otherwise, you probably need to:

1. Find out about what a home page should and should not contain. For example, you could use Google to search for information about how to create a home page.
2. Find out about HTML, and then about CSS.
3. You probably need a tutorial introduction (but some people might prefer an essay or article about these topics, or even a reference manual). Decide what would suit you best and try to locate some appropriate information - just make sure that it is reasonably up-to-date and trustworthy!
4. If you are already familiar with HTML and CSS, you should do something extra. Compare the different versions of HTML and CSS. Find out what is available, and add information to your implementation web-page to show which you have used and why. Pick something sensible, but if possible, something that you are not already familiar with.

Your implementation page should say what software you used to create your web-pages and to check them for correctness, e.g. HTML and CSS versions, editor, operating system, browser, validator, etc.. It should also list the information sources you used. Remember to create hyperlinks between your implementation page and your home page, and any other pages you create. You could also hyperlink to the home-pages of the rest of your group, including your tutor, and to relevant places on the School Moodle site, and so on.

Use this exercise as an opportunity to hone your version control skills. Remember you'll need to `git add` any files (and the `ex5` directory itself) to bring these under `git`'s control, and you'll also have to `git add` files after any changes that you don't want to discard. Use `git commit` periodically when you have made changes to the files that 'make sense' as a whole (it's better at this stage to over-use `git commit` than to under-use it; you'll only get a good feel for what makes a 'good' commit by trying it out). Get into the habit of making liberal use of `git status` to check that any `git` commands you've run have had the effect you expect: you'll find that even the most experienced `git` users do this regularly, since its much easier to fix any problems or mistakes if you spot them early on.

5.4 Knowledge

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

- The purpose of the `<!DOCTYPE...>` declaration at the top of a well-formed HTML file.
- How to structure a HTML document using `<html>`, `<head>`, `<title>`, `<meta>`, `<body>`.
- How to include comments in a HTML file (i.e. things that appear in the source file but which aren't rendered when you look at the file in a browser).

- How HTML tags are paired, and how to use self-closing tags.
- How to create different types of headings, paragraphs, enumerated lists and simple tables.
- How to create hyperlinks and anchors.
- How to include images.
- How to represent **reserved characters** if you want them to appear in your web page.
- How to emphasise, quote or cite things.
- How to structure your document using `` and `<div>`.
- How to add extra information to components of your document, such as images, to help those using screen-reading software.
- CSS selectors: tags (e.g. `h2`), classes (e.g. `.centre` or `p.centre`) and pseudo-classes/elements (e.g. `a:hover`)
- CSS properties and values: e.g. `text`, `color`, `font` etc..

5.5 Assessment

When you have finished, and your web pages are in a state you are happy with, do a final `git commit`, and then push this commit to your GitLab account. Then switch back to the desktop PC, and use `git pull` to update the version of COMP10120 that's in your CS filestore. You'll then need to run the `submit` command from within `ex5`, which will submit any `.html` or `.css` files that are in there for marking (it won't submit any image or other media files in there, but don't worry those should be safely stored in GitLab by now). You can then `labprint` while in the `ex5` directory as usual.



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=ex5.xml

Show your web-site to a member of lab staff, and be prepared to answer any questions about what you have done. For full marks, you should have:

1. Produced web pages using HTML.
2. Linked your pages together.
3. Used CSS to enhance the look of your web-pages.
4. Used CSS in a separate style file, rather than included in each page.
5. Edited all your HTML and CSS by hand.
6. Used comments to identify code copied from tutorials etc..
7. Produced a home page with sensible content (e.g. not embarrassing to you or to the School).

8. Made your page accessible for disabled, elderly, low bandwidth etc..
9. Produced correct HTML and checked your pages for HTML errors.
10. Produced an implementation page, with details of software and information sources used e.g. HTML and CSS versions, editor, operating system, browser and validator.

5.6 Taking your understanding further

If you already know a lot about HTML and CSS, or want to deepen your understanding further, this list is a good place to start. Roughly speaking, the topics get more complex as you go down the list.

- The different types of HTML and their pros and cons; decide which you think is most appropriate for your project and why.
- The `` tag was deprecated in HTML 4.01. Find out why.
- Find out how to make web pages more easily accessible for visually impaired readers.
- Read up on **Responsive Web Design** and decide whether you think it might be useful for your team project.
- Find out what **MathML** is. How does this relate to what you've learned about \LaTeX ?
- Explore the **Model View Controller Pattern**, and see how it relates to the distinction in roles between HTML and CSS.