# lecture one: tools & standards

Connect to your bcitcomp.ca account via FTP

**Connect to your bcitcomp.ca account via FTP**

Use Filezilla, or any FTP client to connect to bcitcomp.ca. The bcitcomp.ca server provides you with a space on the internet where you can publish your homework and assignments.

1. choose: File->SiteManager->New Site, then describe the FTP server, eg "bcitcomp.ca"
2. Host: bcitcomp.ca
3. Port: 21
4. User: {yourStudentNumberUsingAnUppercaseA}@bcitcomp.ca
5. Password: {determined by you via email to your instructor}
6. Servertype: FTP File Transfer Protocol
7. LogonType: Normal
8. click: Connect

**Tools: Web Development**

There are a LOT of tools to choose from these days. Desktop and mobile apps, online tools and services, etc.

**What kind of tools do you use?**

Workflow is defined as the process you take getting from start to completion on a task or series of tasks. For web development, it generally results in a page that has been created or updated by the use some kind of tool.

There is no perfect platform or workflow, there is only **your** platform and workflow that matters. Once you can get the job done, you are off to the races, so find a workflow that suits you the best.

Some of the tools used by web developers...

**Text Editors**

* [Notepad++](http://notepad-plus-plus.org/) - Text Editor (Win) \* installed in BCIT lab computers \*
* [Sublime Text 2](http://www.sublimetext.com/2) - Text Editor (Mac/Win)
* [Text Wrangler](http://www.barebones.com/products/textwrangler/) - Text Editor (Mac)

**FTP (File Transfer Protocol) Clients**

* [Filezilla](http://filezilla-project.org/) - FTP tool (Win) \* installed in BCIT lab computers \*
* [Cyberduck FTP](http://cyberduck.ch/) - FTP tool (Mac)
* [WS FTP](http://www.wsftple.com/) - FTP tool (Win)

**IDEs (Integrated Development Environments)**

* [Coda](http://www.panic.com/coda/) - Web Dev IDE (Mac)
* [Dreamweaver](http://www.adobe.com/products/dreamweaver.html) - Web Dev IDE (Mac/Win)
* [Aptana](http://www.aptana.com/) - Web Dev IDE (Win/Mac)
* [HTMLKit](http://www.htmlkit.com/) - Web Dev IDE (Win)
* [Textastic](http://www.textasticapp.com/) - Web Dev IDE (iPad)
* [Komodo](http://www.activestate.com/komodo-edit) - Web Dev IDE (Mac/Win)

**Team Development**

* [Dropbox](http://dropbox.com/) - File Sharing (Web)
* [GitHub](https://github.com/) - Collaborative Development (Web/Client)

**Tools: Browser-based development**

Today, most modern browsers have some kind of development environment that can be turned on or triggered via extensions and/or plug-ins.

There are presently two major development platforms, with a few variations for examining and developing web pages.

* [Firefox](http://www.mozilla.org/en-US/firefox/new/)
  + Mozilla Foundation's web platform created by web developers
  + Gecko rendering engine: fast and progressive
  + Excellent level of standards compliance
  + Corporate independence
  + No built-in development tools
  + Plethora of add-ons that make web development easy
  + Add-ons have a downside: bloat
* [WebKit](http://www.webkit.org/)
  + This is the platform that gives us [Chrome](http://www.google.com/chrome), [Safari](http://www.apple.com/safari/) and [Opera](http://www.opera.com/), with a few other smaller browsers out there as well.
  + Excellent platform that is very fast and standards compliant
  + Third parties can download the source, make local adjustments and release their own version, as Google and Apple have both done
  + Standalone version can be downloaded at any time for the latest release
  + Development is done via the built-in Developer Tools
  + Initially a thin environment, recent releases have made Webkit-based browsers excellent for developers
  + Macintosh based developers may need to [enable web developer options](http://www.jonhartmann.com/index.cfm/2011/4/28/Enabling-Safari-Developer-Tools) in Safari before accessing the devloper tools

**Remember to test in as many browsers as you can! Useful tools for this are**[BrowserShots](http://browsershots.org/)**(Chrome, Firefox, Safari, Opera) and**[Netrenderer](http://netrenderer.com/)**(For IE)**

**Tools: Firefox Extensions**

Extensions are a big part of Firefox, especially considering the background the browser has in web development.

In fact, web development even has it's own category dedicated to [Web and Developer Tools](https://addons.mozilla.org/en-US/firefox/browse/type:1/cat:4)

One of the best tools for Web Developers is the [Web Developer toolbar](https://addons.mozilla.org/en-US/firefox/addon/web-developer/?src=search)

The author, Chris Pedrick, maintains a [site for the Web Developer extension](http://chrispederick.com/work/webdeveloper/). He has even created a version for seeral other browsers, though it isn't quite as full-featured.

Some of the amazing tools this has to offer web developers:

* Disable styles - by embedded/linked/inline as well as all styles, or just a single sheet
* Edit CSS & HTML - Make live changes to your CSS or HTML!
* Images - outlining, file sizes, alt information... lots more
* Form detail - useful for working with form elements
* Cookie examination and manipulation
* Built in validators
* Line Guides & Rulers
* Element inspector

**Let's take a quick look at the Web Developer Toolbar**

* Install it if you have not already done so
* Validate the HTML of this page with the Tools tab (Tools > Validate HTML)
* Validate the CSS of this page with the Tools tab (Tools > Validate CSS)
* View the CSS of the page (CSS > View)

**Browser Based Diagnostic Tools**

* As web pages get more complex, it can become difficult figuring out what is going on. For example, CSS rules that override other CSS rules, if spread across several files, can be challenge to make sense of. Having a diagnostic tool to assist in developing will make it easier to examine every detail of web pages.
  + Diagnostic Tool: Firefox Firebug
  + Diagnostic Tool: Chrome
  + Diagnostic Tool: Safari
* Get familiar with using one of these tools. You will be required to use them in the classes ahead.
* NOTE: changes you make with the tool are client based, this does not actually affect the HTML/CSS files
* These tools go beyond outlining, examining and editing into deep troubleshooting of network, javascript and even XHR (XMLHttpRequest).

**Developer Tools Wrap up**

Firefox was the first browser to have real tools embedded and, with extensions, it's a real powerhouse now

The last few years have seen a constant progression of improved tools in other browsers, bringing everything up a significant notch

Not a bad idea to have a "dev browser" that can be weighted down with tools

**Standards**

**What are standards?**

* Definition of standards?
* What are some examples of standards in real life?
  + Lightbulbs
  + Grades of gasoline
  + Train rail widths
  + TV Broadcast signals

**Why use them?**

* Interoperability
* Predictable results/dimensions/values
* A new specification is not needed for each new manufacturing run
* Need a new microwave? You can count on:
  + Plug/Voltage/Power
  + Unit size
  + Operation

**Web Standards**

When you build without standards, you are investing in the here and now, without thinking of tomorrow.

* Code that doesnt follow standards cannot be expected to behave predictably in the various web browsers
* Non standard code is hard to support
* Re-engineering and/or maintaining poorly coded websites can be very difficult, costly and time consuming

**Standards: Web**

**ISO: International Organization for Standards (**[http://www.iso.org](http://www.iso.org/)**)**

* **1906**: As the Intn'l Electrotechnical Commission (IEC): in order to better access pre-determined equipment that was interchangeable and interoperable
* **1947**: 25 countries got together to form ISO: "to facilitate the international coordination and unification of industrial standards"
* Companies agree to become a part of ISO, and gain access to the complete library of standards set by the organization in concert with their many members
* While it started slowly, many companies ignored standards, in favor of their own "proprietary" methods
* Momentum, corporate marketing (companies advertising that their product or method conforms with ISO XXXX), and some government legislation (forcing the adoption of certain standards) led to a larger adoption of standards

**W3C: World Wide Web Consortium (**[http://www.w3.org](http://www.w3.org/)**)**

* Web standards are going through the same process that the ISO went through
* Many companies see the benefits of interoperability - many are still "forging their own"
* Many companies have standards: Adobe, Microsoft, Apple, Ford, GM
* Many are proprietary standards, and are not supported by many companies
* W3C - World Wide Web Consortium: founded by Tim Berners-Lee to try and uniformly put forward standards through a single communication vehicle
* Leading browser makers did not follow, initially
* Designers were 'taking sides' in the browser wars: users were losers

**W3C Goals**

1. **Universal Access**: To make the Web accessible to all by promoting technologies that take into account the vast differences in culture, languages, education, ability, material resources, access devices, and physical limitations of users on all continents;
2. **Semantic Web**: To develop a software environment that permits each user to make the best use of the resources available on the Web;
3. **Web of Trust**: To guide the Web's development with careful consideration for the novel legal, commercial, and social issues raised by this technology.

**Making Progress**

* Eventually, the drive towards standards began to win out
* Users made their voices known and momentum started to slowly shift toward supporting standards
* Seen as a marketing opportunity: *Our browser supports standards!*
* Mozilla Project emerged after a large number of Netscape developers were let go upon the acquisition by AOL
* Mozilla is now a standards-compliant browser with many features
* Microsoft and AOL followed shortly with adoption of W3C standards (with limitations)
* Large companies started joining the W3C to be seen as part of the Standards game

**For Web Developers**

Designing to current W3C specifications means that:

* your pages will be viewable in the largest number of browsers,
* the largest number of devices
* and (if used correctly) will gracefully degrade as support for CSS or other presentation layers are absent.

Some more words about [Web Standards](http://www.webstandards.org/learn/faq/) worth reading

**Discussion:**

* what is the current W3C recommended version for HTML?
* what is the current W3C recommended version for CSS?
* what are the consequences of using a non-standard version?
* Include a [referer link](http://validator.w3.org/check?uri=referer) on all your COMP1950 pages:   
  **<a href="http://validator.w3.org/check?uri=referer">validate</a>**.

**Details of Standards-Driven Development**

**Three goals of standards-driven development:**

1. compatibility (creating web pages that work well in all browsers)
2. portability (writing code that works well in any computing environment/operating system)
3. accessibility (produce web pages that are optimized for search engines, screen readers, etc)

**Requirements**

* HTML + CSS + DOM = Agile Web Sites
* Structurally valid HTML code (eg: tags that open are closed. parent-child tag relationships are valid, etc)
* Logically valid HTML code (eg: ids are not applied more than once per page)
* Use CSS to develop sites that separate content (HTML) from presentation (CSS)
* Using Semantic Markup to establish content relationships and logical deployment of content

**Future Compatibility**

* Code is easier to use for future developers
* Less presentational HTML means easier design changes
* Keep the same code for longer periods of time
* Optimally, **never** change your code (if all presentation is gone)

**Reduced Development Costs**

* Shorter development times for style changes
* CSS based layouts are easier to redesign, superior to HTML table layouts
* Easier to integrate with CMS (Content Management System) tools

**Special Needs**

* HTML lends itself to assistive readers (low vision, etc)
* Can support multiple stylesheets for different user needs
* An eye-opening exercise: try browsing with [Webbie](http://www.webbie.org.uk/), a browser for the blind

**Single Codebase**

* No complex browser sniffing scripts
* Avoid having to support multiple browsers with multiple home/site pages (code forking)
* Much less upkeep (only one source to update)

**Moving Forward**

* HTML5: utilize the new tags to further increase semantic meaning of our web pages
* The better your HTML elements, classes and IDs, the easier it will be to manipulate them with advanced frameworks such as jQuery

**Standards: file and folder naming**

* There are a few things that bear remembering:
* file and folder names should NOT contain:
  + capital letters
  + spaces
  + special characters: ! @ # $ % ^ & \* ( ) + { } [ ] : ; " < > ? | etc...
* acceptable non-alphanumeric characters: . \_ - ~
* simple default page: index.htm, index.html or index.shtml (or whatever your server default is)

**Standards: file management**

* including an **index.html** file in every folder on your server will ensure users cannot go snooping around your filesystem. the index.html file can be a blank HTML page if you like, as long as the user does not see a list of files when they request a folder
* browser caching can reduce excessive request/response transactions between client and server. however, some browsers may cache things like the CSS, so if the styles have changed since caching the client may not get the latest styles.
  + **Ctrl + F5** (**Cmd + F5** for Mac) will force a page reload, but what if you users don't think to do so?
  + if a file name has changed, the browser will have to make a new request from the server. versioning your file names will ensure users always have the latest version of any file (for example **styles\_1.0.css, styles\_1.1.css, styles\_1.2.css, styles\_2.0.css** etc)
* if there are files/pages on your server no longer being used, be sure to either remove them or locate them in a place where users are not going to accidentally find them

**Semantic Markup**

**What is Semantic Markup?**

* The practice of organizing your content into a structured hierarchy
* Using appropriate tags for the content
* H1 ... 6 tags for summary headings
* UL tags for unsorted, unranked, related points
* OL tags for sorted, ranked, related points
* Paragraph tags instead of double BR tags
* CSS instead of FONT
* HTML tables used for semantic value (tabular data), not for layouts
* Key: Separate Content (HTML) from Presentation (CSS)

**Building Content Relationships**

* Use appropriate tags for headings, content, sub-content, lists
* This builds a relationship that descends from the title of the page
* Each section is a discrete piece of information
* Headings are related to following paragraphs
* Citations, quotes, code samples can be contextualized by nearest heading

**Logical vs. Physical Styles**

* Tend to display identically in browsers, but are fundamentally different
* Physical styles define what content *looks like*
* Logical styles define what content *is*
* Logical styles are important for screen readers, physical styles are not
* Given a choice, logical styles are usually preferred
* Some physical styles:
  + **b** - Bold
  + **i** - Italic
  + **s** - Strikethrough
  + **u** - Underline

**Logical Styling**

* Best way to markup content for long-term usage
* Use CSS to change style (and some behaviour)
* Use DOM to change behaviour (via scripting)
* Requires thinking ahead
* Examples of logical styles:
  + **code** - **Code sample**
  + **kbd** - Keyboard entry
  + **samp** - Programming sample
  + **var** - *Variable*
  + **cite** - *Citation*
  + **dfn** - *Definition*
  + **abbr** - Abbreviation
  + **acronym** - Acronym
  + **del** -
  + **ins** - Inserted text

**Key Advantages Of Semantic Content**

* Content is relational - context can be determined easily
* Extracting important pieces of information is simple (CITE or CODE tags for example)
* Search engine optimization (SEO)
* Syndication (RSS)
* Big win for frameworks such as jQuery (again)!
* Screen reader friendly
* Separates style (CSS) from structure (HTML), making it easier to maintain

**Standards Resources**

WASP http://www.webstandards.org/

W3C http://www.w3c.org

W3C Sites http://www.w3csites.com/

**Summary**

**Prerequisites**

While there is plenty of individual homework to do, there will be several team oriented exercises in this class. Be sure to build pages you are proud of, and be ready and willing to share the them with your colleagues. All COMP1950 students must verify they are capable of connecting to their course webspace via FTP. Students are expected to be fluent with the use of FTP, so be sure to remember your connectivity information.

**COMP1950 Best Practices**

* get acquainted with a browser based developer tool(s). they are crucial to making your development more efficient and effective
* practice standards based development at all times:
  + encourage user trust by building pages that are coherent and well presented
  + facillitate accessibility options for the disabled
  + HTML: used for describing the **semantics** and **structure** of page content
  + CSS: used for applying **presentation** rules (colors, fonts, content location/position)
  + Javascript: enables page **behavior**, response to user interactions
* recognize that only structuraly valid code can be expected to operate predictably
  + [W3C HTML validator](http://validator.w3.org/)
  + [W3C CSS validator](http://jigsaw.w3.org/css-validator/)
  + Include a [referer link](http://validator.w3.org/check?uri=referer) on all your COMP1950 pages:   
    **<a href="http://validator.w3.org/check?uri=referer">validate</a>**.

**Hypertext Markup Language**

It all starts here in one form or another: HTML

* What started as a language defined by [SGML](http://www.w3.org/MarkUp/SGML/) *(Standard Generalized Markup Language)*, HTML is used to mark up billions of pages, making up the bulk of the web.
* From a common origin, 3 distinct version have emerged:
  1. **HTML 1 - 4.01**: This was developed in rapid succession in the 1990's
  2. **XHTML 1, 1.1, 2**: A stricter version, created as an answer to the loose development of HTML
  3. **HTML5**- a reaction to the draconian ruleset proposed by the W3C for the future of MLs beyond XHTML 1.1

**HTML Created**

* Late 1991, Tim Berners-Lee releases the first iteration of what would become HTML 2.0 (there was no real 1.0+)
* HTML was a language created at the dawn of the Web, using SGML (Standard Generalized Markup Language) as a template
* Over the next decade, various additions, improvements and changes are introduced
* The last official specification was the service release of HTML 4.01 in December 1999
* Initially used for documents and a place for the technically inclined, the web did not have the international pervasiveness it does today
* Most developers were new to the language and there were no real classes in it
* People had to learn as they went along
* A lot of early sites are poorly designed

**HTML 4.01 template**



**HTML on the loose!**

* Despite it's reputation as a loose language, HTML 4.01 can be written strict just like XHTML and HTML5 can
* The problem was not *entirely* with the language or the authors. It was with the *browsers*.
* Browser manufacturers created User Agents that would read code, interpret it, fix any mistakes and render a page, without any visible errors.
* It was this that led to coders exploiting problems in browsers and creating pages that broke when new specs or updated User Agents (browsers) came out
* Okay, so now there is a problem. **Millions** of pages are being produced with sloppy code because the UAs let it happen
* The web is becoming filled with Tag Soup that makes it hard to index, with search tools having to trust things like META keywords and descriptions
* A better solution is required: Get Strict!

**Attempting to fix the problem**

**Moving on to XHTML1**

* After years of HTML development, 4.01 became the "Last Official Release"
* The Web Working Group had taken a look at XML (Extensible Markup Language) and liked what they saw: a requirement for valid markup in order for an XML application to work
* Why not take the ruleset from XML (lower case attributes, everything in quotes, close every tag) and apply it to HTML? Yay!
* That should fix everything, right?
* Coding wasn't actually fixed by the introduction of XHTML. Fixes were suggested by the specification, but most browsers would render the page anyway.
* This is because 99% of XHTML pages are reporting their MIME type as: text/html
* To get pages to render using the appropriate DOCTYPE of XHTML, pages would have to be delivered with the MIME type of application/xhtml+xml
* Using that MIME type, however, brought with it a rather dire proviso:
* The error handling specified in XHTML 1.1 meant all errors will cause the browser to cease rendering
* Yes, you read that correctly. That means any error will throw an warning to the user and the page will not be shown
* Try to imagine the Web actually working in this way
* Why was this done? Because the developers of XHTML wanted to force strict coding rules so that the Web could be seen as a giant application. Reliable code on every page meant you could use the data anywhere

**Tighten up!**

* X = eXtensible
* X = XML related
* A reformation of HTML reformulated in XHTML
* First major update to HTML since the 4.01 specification was released in 1999.
* Note - XHTML Development has ceased entirely. More on that later
* Better, more predictable coding patterns
* "verifiable" or "well-formed" code can be produced
* Forces a higher standard of HTML
* Porting to HTML5 is easy (if you also observe semantic guidelines)

**What is different about XHTML?**

* Almost identical to HTML 4.01, but a much stricter syntax in XHTML
* Technically: **XHTML is just HTML defined as an XML application.**

**XHTML Rules:**

* All elements must be properly nested
  + This is not valid: **<p><b>Bolded Text</p></b>**
  + This can be a problem with things like Lists <ol> and <ul>.
  + Especially when closing the List Item <li> tag that contains the <ul> or <ol> tag.
* All tags must close
  + Example: **<p>Some text here</p>**
  + Empty Elements are closed as well: **<br />** or **<img src="#" />**
  + An extra space is required before the slash to work with all browsers.
* All tag names must be in lower case
  + **<Img Src="some\_image.gif">** is not valid
* Attribute names are always in lower case
  + **<p ALIGN="center">** should be **<p align="center">**
* All attributes must be quoted
  + **<body bgcolor=black>** will not work, you need: **<body bgcolor="black">**
* Attributes cannot be minimized
  + **<input type="checkbox" checked>** cannot be used. The valid code is: **<input type="checkbox" checked="checked" />**
* Name Attribute is no longer used (except with forms). Replace "name" with "id".
  + **<img src="picture.gif" name="myimage" />** will not validate. Use **id="myimage"**instead.
* Mandatory Elements: Every XHTML document **must have** these elements:
  + **<html>**
  + **<head>**
  + **<title>**
  + **<body>**
  + The **<!DOCTYPE>** declaration *must* be there, but it is *part* of the document itself rather than an element of the document.
* Documents must be well-formed
  + The document must conform to all of the above rules
* Optional XML declaration
  + Not *required*, but good practice:
  + **<?xml version="1.0" encoding="ISO-8859-1"?>**
* Optional Element: An XHTML document should also declare the type and character encoding in the head section to ensure the server sends the correct content-type header:   
  **<meta http-equiv="Content-Type" content="text/html" charset="UTF-8">**

**XHTML Documents**

**DTD: Document Type Definition**

* A description of the document, and the legal definitions that fall within it.
* The DTD defines the syntax of the page in SGML.
* This translates to: Because XHTML is just the application of XML to HTML, you need to tell the browser (the interpreter) how to interpret all of the code on the page.

**Three kinds of DTD:**

**Strict**

* **<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">**
* Mostly used for technical documents or content that requires very little markup.
* This is the original intention of the W3C"s recommendation.

**Transitional (most common)**

* **<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">**
* This is by far the most common DTD in use today
* Supports most HTML features
* Was created to provide a middle ground for users trying to convert from HTML to XHTML in stages.

**Frameset**

* **<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">**
* Same as Transitional, but allows the use of frames

**XHTML 1.0 template**



**XHTML2**

* Between 2002 and 2006, a significant amount of development was attempted for XHTML2
* An early decision was to abandon backward compatibility with XHTML1 and HTML4.x leading to significant controversy
* What does that *mean*? "abandon backward compatibility"?
  + Deprecated tags would no longer work
  + All "old" code would have to be updated if you wanted to deliver XHTML
  + Browsers would have even more work to do when rendering pages
* Some additional changes:
  + Forms will now be controlled by XForms, a new system that allows for error checking, preformatted input fields and more
  + DOM events will be replaced with XML Events - a system for registering and handling events within an XML document, requiring very strict coding standards
  + Lists for Navigation can be created with a new **n1** tag
  + Any element can have **href** or **src** attributes, allowing for links and/or image replacement
* Some of the more contentious issues were split out into modules to be worked on by the W3C XHTML Working Group
* At one point, a question was asked:
  + Shouldn't we be taking into account the billions of pages that are already out there, make the upgrading part easy and focus on the kinds of things people are actually doing today?
* Maybe not *exactly* that, but close
* A vote was called and the motion was defeated. But the people who thought it was a good idea wanted to move forward
* As a result - they created the WHATWG (Web Hypertext Application Technology Working Group) and began working on a competing specification, which was originally called Web Applications 1.0

### **Arising out of the morass**

* HTML5 was born out of frustration with languishing XHTML2 development
* The last official spec of XHTML had come in 2000 and by 2005, there was little forward movement, with the development bogged down in committees and problems reaching consensus
* There were two distinct camps: Strict and Non Strict (overly simplified terms)
* After a series of decisions against a more forgiving, backward compatible version, a number of HTML Working Group members created their own group: WHATWG
* After 3 years, Web Applications 1.0 gained considerable momentum when Mozilla, Apple and Opera joined the Working Group and requested that it become part of the W3C's HTML Working Group
* In 2007, the W3C HTML working group adopted the development, renamed it HTML5 (a successor to HTML 4.01) and released a Public Working Draft in 2008
* A
* As of October 28, 2014, [**HTML5 is the official W3C recommendation**](http://www.w3.org/TR/html5/)
* Broswer vendors are now expected to fully support HTML5 in new browsers they produce.
* Not all browser vendors are up to date yet, and, much more notably, not all internet users will be using new, HTML5 compliant browsers
* Support for some of HTML5s features is already available in several browsers, so if applied with care, developers may begin writing HTML5 now.

**HTML 5 template**

<!DOCTYPE html>

<html>

<head>

<title></title>

<meta charset="utf-8" />

</head>

<body>

</body>

</html>

**Character Sets And Languages**

The W3C recommends the [UTF-8 charset](https://www.w3.org/International/questions/qa-choosing-encodings) for web development. It includes an impressive array of languages and special characters, and is well supported in common devices.

**Assign Character Set**

If it is not specified, a web browser will choose the character set it thinks is most appropriate, but it is a better practice for the developer to ensure the correct character set by applying the **meta** tag to the **head** section of every HTML page:

**<meta charset="utf-8" />**

* Write all your text-based code (**.html**, **.css**, **.js**, **.php**, etc) in UTF-8. At development time, ensure your text editor is saving files in this format. This setting is usually found in the application's Preferences menu option.
* Unless your audience includes very old browsers, you can [code special characters](https://tools.oratory.com/altcodes.html) directly into the HTML (you dont need to use HTML entities for most special characters).
* You cannot copy-paste rich text (such as Microsoft Word, or WordPad) into your HTML. The rich text formatting will result in an unreadable mess.
* An HTTP server can over-ride your charset preferences when it generates the HTTP header. Details on how to control this will be discussed later in this course.

**HTML lang Attribute**

Use the **lang=""** attribute to define the language within. This is usually applied to the **html** element. Assigning the appropriate language code will apply to all contained content, unless over-ridden with another **lang=""**.

Use the language abbreviations from the [standard ISO 639-1](http://www.loc.gov/standards/iso639-2/php/code_list.php) code set (don't use ISO 639-2).



**HTML5**

**HTML5: What is It?**

**The basics of HTML5**

HTML5 is more evolutionary then revolutionary. The bulk of previous HTML elements and attributes are maintained, with a few deprecations, and several additions.

* Some of the most important parts of HTML5 are:
  + Ultra simple Doctype: **<!DOCTYPE html>**
  + Specifies how browsers should behave with imperfect code
  + XML can be included as a markup language from within HTML5
  + Ability to embed [MathML](http://www.w3.org/Math/) or [SVG](http://www.w3.org/Graphics/SVG/) elements
  + Standardized Javascript API increases client side script compatibility across browsers
  + Backwards compatible - older versions of HTML can effectively be updated by simply changing the doctype to HTML5
  + Guiding principles: Enhance semantic coding, Support existing content, Pave the cowpaths
* Some of these seem insignificant, but they are great strides forward

# lecture ten: methodology

## Collaborative Workflow

1. A team member makes updates to local files

2. After testing their work, they commit to the changes

3. After committing the changes, local files are synced with publicly shared files

4. Publicly shared files are updated, and the updates are propagated to the local files of other team members

What is Git?

* A Git is a Version Control System (VCS). It is open source, created by Linus Torvalds of Linux fame. It allows you to update, change, and track all of your changes to any code you are working on, and to share the progress of a project with several team members.
* Git is installed on your system (Win/Mac/Lin) and "watches" files and directories for changes
* A hidden directory (.git) is added to the location where you want your files tracked
* From then on, any changes to files in that directory or any sub-directory are tracked
* You can then keep versions of them within a repository
* A repository (or repo) is an area that tracks your files and any changes made to them
* You can make changes to a file, realize the changes caused a problem and "roll-back" those changes to an earlier version
* You can also share code with others without having to create login credentials for people on an FTP server
* Repos can be public or private, can be on local or remote systems and can be very large or very small (1 file)
* Version control is a great way to avoid having 15 versions of index.shtml (index.shtml.old, index.shtml.orig, index.shtml.do-not-delete, etc.)
* Beyond just having different versions of a file, you can actually SEE what has changed
* Some diff (seeing the changes - ie: difference) applications are built into some Git management tools, others are standalone

**Create A GitHub Account**

**Install Git and GitHub GUI**

**Create your own Repo**

**Adding Files**

* From here on in, adding files is fairly easy. You can add them in Explorer/Finder and git will track which ones are new or have changed. Choose the *Changes* option to see GitHub's monitoring of the files you add and the changes you make.
* Once file(s) have been added, you can *commit* the file. This adds it to a *local staging* area, but it has not yet been *pushed* to GitHub.
* Back in the GitHub GUI, view the *changes* section.
* This shows all the files that have been changed or updated
* In order to ready your files to be *pushed* to GitHub, enter a *commit* summary message and (optionally) a more detailed description. Explain what and/or why you have changed the code, then click **commit**.
* Now your files are *committed* but still not *pushed*.
* At the top-right of the GitHib GUI, click on "publish" or "sync". This *pushes* all the *commits* to the *remote, public repo*.

**Reviewing Changes**

* Your files should now be on [GitHub](https://github.com/). You can check by going back to your account and checking to see what repos are there, and what files are in that repo.
* There are a few ways to see changes - clicking on the time it was edited, or viewing the changed files through the GUI.
* You can also *edit* any of the text-based files right on GitHub, and then get the GUI to *pull* the changed back to your local machine.

### **Clone a Repo**

* I have created a small repo (just 1 file) that is from our jQuery lecture, and I want to share it with all of you.
* The easiest way for you to do it is to search, browse or find the repo. Here is a link to it directly: [**jQuery Code Example**](https://github.com/comp1950github/jQuery_example)
* Then choose Clone In Desktop (be sure your Git Client is running)
* This will bring down the repo file(s) to your client machine, allowing you to develop your own copy. Note: cloning a repo in this way does not allow you to contribute updates or changes to the master repo
* Learn more about [**how to clone a repo**](https://help.github.com/desktop/guides/contributing/cloning-a-repository-from-github-to-github-desktop/)

### **Contributing to a Repo**

Repository Owner:

* The owner of a repo can decide who, if anyone, is allowed to contribute to the code.
* Log in to the remote Github.com server
* Find and select the repo
* Choose Settings, then Collaborators
* Add the GitHub username of each individual to your collaborative team
* Learn more about [**how to contribute to a repo**](https://help.github.com/desktop/guides/contributing/)

### **Branch A Project**

Repository Collaborator:

 Before you can contribute to a project, you must be invited to be a contributor by the repository owner (described above). Log into GitHub.com and go to the notifications section (the bell icon at top right) to accept the invitation

 Choose "fork" from the top right of the GitHub.com page. This will create your own copy of the master files, associated with your account (note this branch of the repo is located at your account URL, not the repo owner's).

 Clone this forked repo as described above

 Add, edit, change the local files as needed

 Commit, and sync when satisfied with the stages of your development

 When your contributions are complete, choose to create a "Pull Rrequest" in the Git Client. This will notify the owner of the repo that your code is ready to be reviewed and/or merged

 Some [**help with GitHub branching**](https://help.github.com/desktop/guides/contributing/creating-a-branch-for-your-work/)