HTML & Git

Programming with Web Technologies

THE UNIVERSITY OF AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND

THE UNIVERSITY OF WAIKATO
Te Whare Wānanga o Waikato

Auckland ICT Graduate School

HTML

- **H**yper**T**ext **M**arkup **L**anguage, used to describe the content and structure of documents on the web (web pages)
- The first version of HTML
  - text-only documents
  - define parts of a document content semantically (e.g. a title, paragraphs, headings, lists)
  - define links as anchors
- Now it is more complex – more than just text in web pages
  - more complex document structures
  - enhanced user interaction with pages

HTML

- Different versions as standards evolved
  - Current version is HTML5
  - https://en.wikipedia.org/wiki/HTML

- Standards are developed by W3C (World Wide Web Consortium)
  - http://w3.org
  - International body that defines HTML
    - Members include Google, Apple, Dropbox
  - And other standards (e.g., CSS, DOM, HTTP, XML, SVG etc.)

HTML

- An HTML document consists of two types of information
    - The content of the document
    - Markup that describes the content
- Markup is in the form of element tags
    - The content is (generally) surrounded by a start tag and an end tag
    - A tag is a letter or word, surrounded by angle brackets
        - `<b>`, `<body>`, `<table>`
    - A closing tag is additionally has a forward-slash after the first bracket
        - `</b>`, `</body>`, `</table>`
    - Excepts for a few exceptions which will be discussed as needed, opening tags should have a corresponding closing tag

HTML Example

A simple HTML document

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>
            My First HTML page
        </title>
    </head>
    <body>
        <p>This is the content of my first HTML page</p>
    </body>
</html>
```

HTML Example

A simple HTML document

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>
            My First HTML page
        </title>
    </head>
    <body>
        <p>This is the content of my first HTML page</p>
    </body>
</html>
```

The <!DOCTYPE> declaration is the very first thing in the HTML document. It is an instruction to the web browser about what version of HTML the page was written in

This tag does not require a matching closing tag

HTML Example

A simple HTML document

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>
            My First HTML page
        </title>
    </head>
    <body>
        <p>This is the content of my first HTML page</p>
    </body>
</html>
```

The content of the HTML document needs to be enclosed by the `html` tag. All other tags will be nested inside this

HTML Example

A simple HTML document

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>
            My First HTML page
        </title>
    </head>
    <body>
        <p>This is the content of my first HTML page</p>
    </body>
</html>
```

A container for all informative tags. Tags inside the head of the document do not appear directly on the page, but can influence how elements are displayed

HTML Example

A simple HTML document

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>
            My First HTML page
        </title>
    </head>
    <body>
        <p>This is the content of my first HTML page</p>
    </body>
</html>
```

`meta` tags contain information about the HTML document. This can include a description of the page, the author and keywords. This information can be used by search engines to help index your page and by the browser to determine how to display the content

This tag does not require a matching closing tag

HTML Example

A simple HTML document

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>
            My First HTML page
        </title>
    </head>
    <body>
        <p>This is the content of my first HTML page</p>
    </body>
</html>
```

Start and End point of the text that is the title of the document. This text will appear as the tab and window name when the page is loaded in a browser

HTML Example

A simple HTML document

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>
            My First HTML page
        </title>
    </head>
    <body>
        <p>This is the content of my first HTML page</p>
    </body>
</html>
```

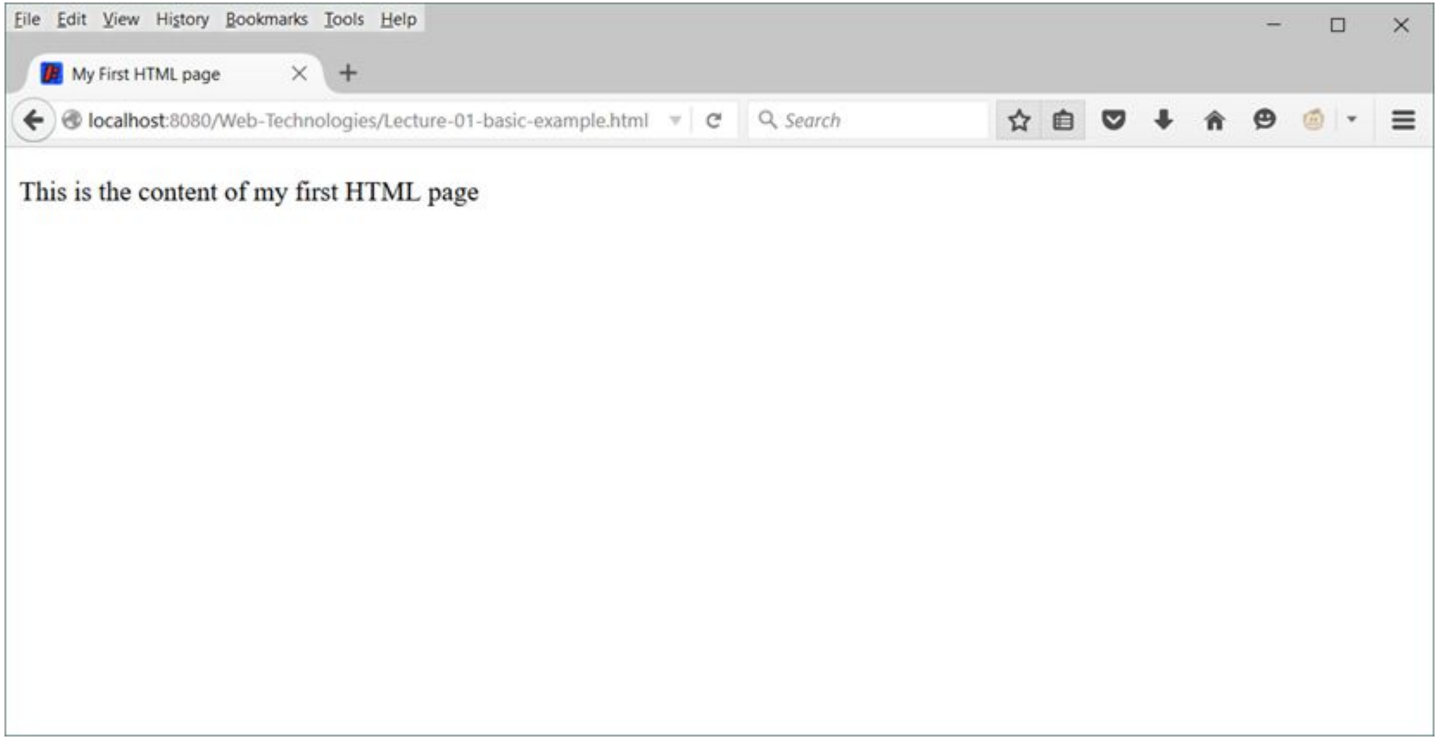
A container for the tags that make up the document. Tags inside the `body` of the document appear directly on the page

HTML Example

A simple HTML document

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>
            My First HTML page
        </title>
    </head>
    <body>
        <p>This is the content of my first HTML page</p> ← A paragraph of text
    </body>
</html>
```

HTML Example - Rendered


File Edit View History Bookmarks Tools Help
My First HTML page
localhost:8080/Web-Technologies/Lecture-01-basic-example.html
Search

This is the content of my first HTML page

HTML Example - Notes

- Whitespace has no meaning most of the time, so the previous code is equivalent to this
  - `<!DOCTYPE html><html><head><meta charset="UTF-8"><title>My First HTML page</title></head><body><p>This is the content of my first HTML page</p></body></html>`
- Code like the above is impossible to debug and maintain, so you should use whitespace liberally to indent and provide visual structure to your code
  - Your IDE and text editor will likely try to do this for you

HTML Notes

Tags should be correctly nested – Tags should be closed in the opposite order to which they were opened

```
<p><em> Some content </em></p>
```

✓

```
<p><em> Some content </p></em>
```

✗

Bad HTML

- When the web started to become popular with the general public, people with no experience or training began to create HTML web pages
    - A good illustration of what the web used to be
- Modern browsers are resilient to many HTML errors
    - Even with an error something would be displayed
    - It can however, be difficult to determine what something will look like if it is invalid HTML
        - Each browser will respond differently
- We do know that browsers deal correctly and predictably with valid HTML
    - We should make sure that the HTML that we write is valid
    - Validation tools are available to ensure that our HTML is valid
        - Built-in IDE tools or online validators

HTML - Spot the Errors

```
<head>
</head>
<title>
    My First HTML page
</title>
    This is the content of my first HTML page
</body>
</html>
```

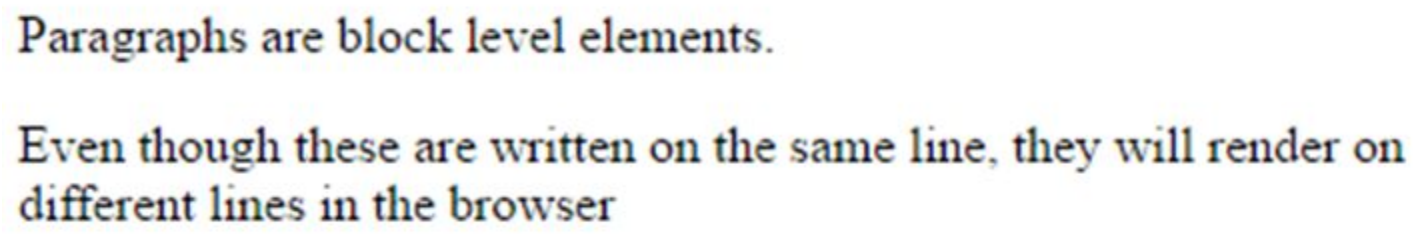
HTML Element Types

There are two types of elements which can be used for content in HTML

- Block level elements
    - Always starts on a new line
    - Expands to fill the available width (stretches out to the left and right as far as it can)
- Inline elements
    - Does not start on a new line
    - Only takes up as much width as necessary

HTML Elements - Block Level

```
<p>Paragraphs are block level elements.</p><p>Even though
these are written on the same line, they will render on
different lines in the browser</p>
```

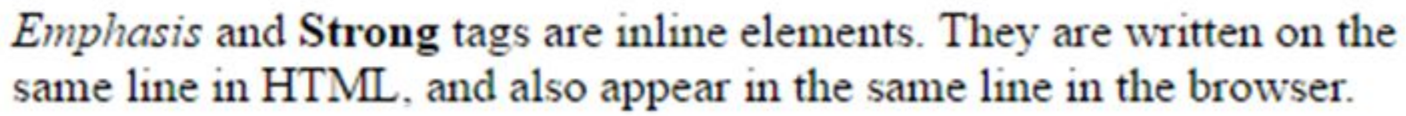


Paragraphs are block level elements.

Even though these are written on the same line, they will render on different lines in the browser

HTML Elements - Inline

```
<p><em>Emphasis</em> and <strong>Strong</strong> tags are
inline elements. They are written on the same line in HTML,
and also appear in the same line in the browser.</p>
```


Emphasis and **Strong** tags are inline elements. They are written on the same line in HTML, and also appear in the same line in the browser.

HTML Elements - Inline and Block Nesting

Inline elements

- Can be nested inside block level and other inline elements

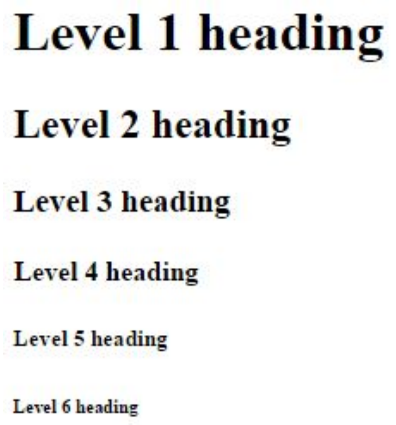
Block level elements

- Can be nested inside other block level elements
- **Cannot** be nested inside inline elements

Headings

- HTML supports six levels of heading, which can be used to organize documents into sections
- `<h1>` is the most important, and `<h6>` is the least important
- Search engines and other tools using heading importance to index web pages, so use them appropriately

```
<h1>Level 1 heading</h1>
<h2>Level 2 heading</h2>
<h3>Level 3 heading</h3>
<h4>Level 4 heading</h4>
<h5>Level 5 heading</h5>
<h6>Level 6 heading</h6>
```



Level 1 heading

Level 2 heading

Level 3 heading

Level 4 heading

Level 5 heading

Level 6 heading

Text in HTML

Text written in the `<body>` of an HTML document will be rendered into the browser directly, however it is good practice to surround text with appropriate tags to convey meaning/intent

- `<p>` - Indicates the text enclosed is a paragraph. Default styling will force a line break between paragraphs
- `<pre>` - Preformatted text. Text will be displayed in the browser exactly as it is typed in the HTML, with whitespace preserved
- `<code>` - Source code, by default displayed using a monospaced font

Text in HTML

- `<blockquote>` - A blockquote originating from a different source, by default displayed using a monospaced font
- `<q>` - A short quote, displayed inline
- `<cite>` - An inline citation of the title of an article, movie, book, etc
- `<address>` - An address, displayed in italics by default
- `<ruby>` - A ruby annotation, a pronunciation guide for Japanese or Chinese text to aid pronunciation of unfamiliar words

And many more, ranging from general to extremely special cases

Lists

HTML5 supports three varieties of list

- Unordered lists
  - Bullet points. Used where order doesn’t matter
- Ordered lists
  - Numeric list. Used where order matters
- Description lists
  - Term, Description pairs. Used when you want to define or describe terms

Unordered List

HTML Markup

```
<ul>
  <li>Entry 1</li>
  <li>Entry 2</li>
  <li>Entry A</li>
  <li>Entry B</li>
</ul>
```

Rendered Result

- Entry 1
- Entry 2
- Entry A
- Entry B

Ordered List

HTML Markup

```
<ol>
  <li>Entry 1</li>
  <li>Entry 2</li>
  <li>Entry A</li>
  <li>Entry B</li>
</ol>
```

Rendered Result

1. Entry 1
2. Entry 2
3. Entry A
4. Entry B

Description List

HTML Markup

```
<dl>
  <dt>Coffee</dt>
  <dd>- Hot black drink</dd>
  <dt>Milk</dt>
  <dd>- Cold white drink</dd>
</dl>
```

Rendered Result

Coffee

    - Hot black drink

Milk

    - Cold white drink

Nesting Lists

HTML Markup

```
<ol>
  <li>Entry 1</li>
  <li>Entry 2</li>
  <li>Nested
    <ul>
      <li>Entry A</li>
      <li>Entry B</li>
    </ul>
  </li>
  <li>Entry 4</li>
</ol>
```

Rendered Result

1. Entry 1
2. Entry 2
3. Nested
    - Entry A
    - Entry B
4. Entry 4

HTML Formatting

HTML is designed to be used for content and structure, not presentation

Modern HTML5 formatting tags imply intent rather than appearance

Older HTML tags are still supported but their meanings have changed with HTML5

- Instead of <`b`>, you might use <`strong`>
- Instead of <`i`>, you might use <`em`>

HTML Formatting

`<strong>`	**Important** text[1]	`<del>`	~~Deleted~~ text
`<em>`	*Emphasised* text[2]	`<sup>`	$^{\text{Superscript}}$ text
`<ins>`	Inserted text	`<sub>`	$_{\text{Subscript}}$ text
`<b>`	**Stylistically** offset text[1]	`<mark>`	Marked text
`<i>`	*Offset* the mood of text[2]	`<small>`	Smaller text

[1], [2]: These tags will have the same default appearance, however their intents are different.

HTML Formatting

- `<hr>` - Indicates a "Thematic break" in the document content, often shown as a horizontal line across the page. Does not need a closing tag
- `<br>` - Forces a newline in some text, which is often used in representing addresses. Does not need a closing tag
- `<!--    -->` - A comment. Anything written between the `<!--` and `-->` markers will not be displayed on the page

HTML Entities

- Some characters have special meaning in HTML; such as angle brackets and quotes; or can't be typed with a typical keyboard
- If we need to display one of these, we can use HTML entities

Result	Description	Entity Name
	Non-breaking space	`&nbsp;`
>	Greater than	`&gt;`
<	Less than	`&lt;`
&	Ampersand	`&amp;`

Result	Description	Entity Name
"	Double quote	`&quot;`
'	Single quote	`&apos;`
©	Copyright Symbol	`&copy;`
™	Trademark Symbol	`&trade;`

More entities here

Tables

- Often we are given data that makes sense to present in a tabular fashion
- Tables are very flexible and have been used for page layout in HTML document for many years
    - No longer recommended as it distracts from the purpose of a table – displaying tabular data
- Trend with HTML5 is to avoid structures like tables for layout, and instead leave that to CSS
    - More on this in future lectures
- Tables make use of a collection of HTML tags

Tables

- `<table>` - Defines an HTML table, and groups the components of the table together
- `<tr>` - A row in the table that contains a number of cells
- `<td>` - A data cell in the table
- `<th>` - A header cell in the table
- `<thead>` - Table header, represents the top-most row of a table
- `<tbody>` - Table body, the main content of a table
- `<tfoot>` - Table footer, represents the bottom-most row of a table
- `<caption>` - A caption to attach or associate with the table, must appear immediately after the opening `<table>` tag

A Basic Table

```
<table>
    <tr>
        <td>col1row1</td>
        <td>col2row1</td>
        <td>col3row1</td>
        <td>col4row1</td>
    </tr>
    <tr>
        <td>col1row2</td>
        <td>col2row2</td>
        <td>col3row2</td>
        <td>col4row2</td>
    </tr>
</table>
```

col1row1	col2row1	col3row1	col4row1
col1row2	col2row2	col3row2	col4row2

Basic Table Observations

The table will have as many rows as there are `<tr>` elements

The table will have as many columns as the largest number of `<td>` elements inside a `<tr>`

- If one row has five cells, and any others have 3 cells each, the table will have five columns
- Cells are placed left-to-right

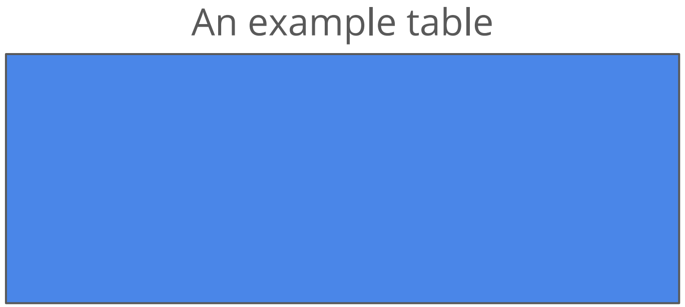
Intermediate Table

```
<table>
```

```
</table>
```

Intermediate Table

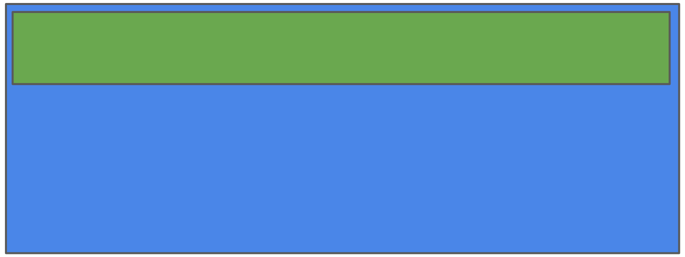
```
<table>
    <caption>An example table</caption>









</table>
```



Intermediate Table

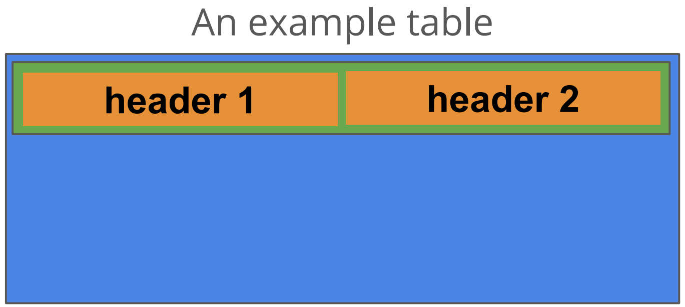
```
<table>
    <caption>An example table</caption>
    <thead>
        <tr>


        </tr>
    </thead>










</table>
```

An example table



Intermediate Table

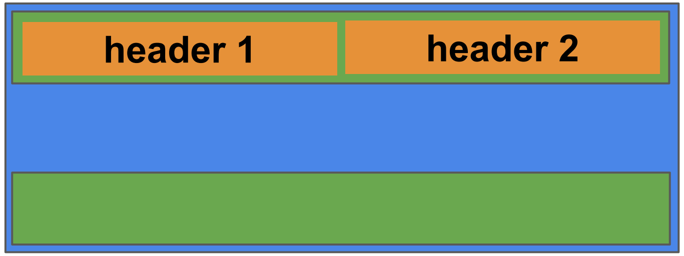
```
<table>
    <caption>An example table</caption>
    <thead>
        <tr>
            <th>header 1</th>
            <th>header 2</th>
        </tr>
    </thead>






</table>
```



Intermediate Table

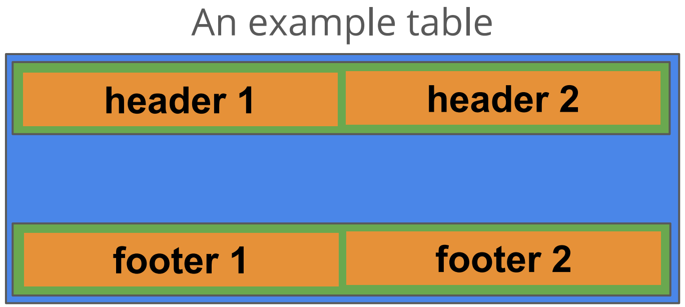
```
<table>
    <caption>An example table</caption>
    <thead>
        <tr>
            <th>header 1</th>
            <th>header 2</th>
        </tr>
    </thead>
    <tfoot>
        <tr>


        </tr>
    </tfoot>






</table>
```

An example table



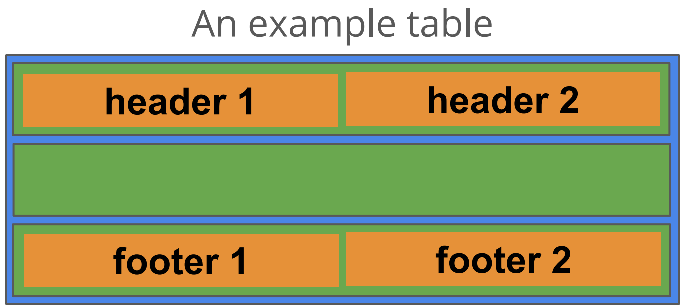
Intermediate Table

```
<table>
    <caption>An example table</caption>
    <thead>
        <tr>
            <th>header 1</th>
            <th>header 2</th>
        </tr>
    </thead>
    <tfoot>
        <tr>
            <td>footer 1</td>
            <td>footer 2</td>
        </tr>
    </tfoot>




</table>
```



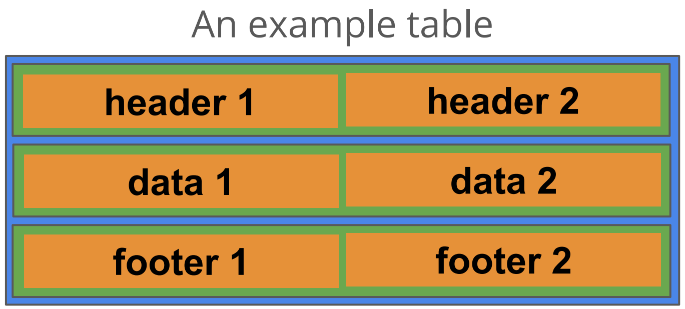
Intermediate Table

```
<table>
    <caption>An example table</caption>
    <thead>
        <tr>
            <th>header 1</th>
            <th>header 2</th>
        </tr>
    </thead>
    <tfoot>
        <tr>
            <td>footer 1</td>
            <td>footer 2</td>
        </tr>
    </tfoot>
    <tbody>
        <tr>


        </tr>
    </tbody>
</table>
```



Intermediate Table

```
<table>
    <caption>An example table</caption>
    <thead>
        <tr>
            <th>header 1</th>
            <th>header 2</th>
        </tr>
    </thead>
    <tfoot>
        <tr>
            <td>footer 1</td>
            <td>footer 2</td>
        </tr>
    </tfoot>
    <tbody>
        <tr>
            <td>data 1</td>
            <td>data 2</td>
        </tr>
    </tbody>
</table>
```



Intermediate Table Observations

- The order of `<tbody>`, `<thead>` and `<tfoot>` inside the `<table>` does not matter
  - `<thead>` will always be rendered as the top-most row in the table, above the `<tbody>`
  - `<tfoot>` will always be rendered as the bottom-most row in the table, below the `<tbody>`

- The order of `<tr>` elements does matter. They will be displayed in the order in which they are declared in their respective sections

HTML Attributes

Attributes are name/value pairs that are seen inside the opening tags of HTML elements. We have seen some of these already

```
<html lang="en">

<meta charset="UTF-8">
```

All HTML elements can have attributes, some just have more than others

HTML Attributes

Attributes are used to provide additional information about an element and come in 4 major varieties

- Required – The tag needs these attributes in order to function correctly
- Optional – The tag does not require these attributes to function, but useful functionality can be added using them
- Standard – Available to most tags, provide general functionality
- Event – Will discuss these in future lectures

HTML Attributes

- Tables
  - `colspan="2"` – Makes a <`td`> span across multiple columns (merging cells horizontally)
  - `rowspan="2"` – Makes a <`td`> span across multiple rows (merging cells vertically)
- Ordered lists
  - `reversed` – Makes the list numbers count down rather than up (this attribute does not need a value)
  - `type="i"` – Pick the variety of marker to use in the list (roman numerals, alphabet, etc)
- A large variety of attributes exist, and many apply to specific tags
  - Many and difficult to remember options
  - To find the available attributes for a tag, use the w3schools tag reference
    - Locate the tag you are interested in and check the "Attributes" section
    - Following links to attributes will tell you available options and how to use them

Browser Inspections

- Most modern web browsers allow you to see the HTML (and other files) of a webpage in a couple of ways
- View Source
    - This option can typically be found in the right-click menu of a browser, and will show you the complete source of a page in a new window.
- Inspection
    - Again, this option can often be found in the right-click menu. This differs from view source by being dynamic – You can change the source and see your changes reflected.

Backing Up Your Data

- As with any work, you want to make sure you backup your code. You may have used cloud services like Google Drive or Dropbox in the past for other work
    - While these are great tools, they are not the best choice for code
- Cloud services don't typically keep incremental backups, and if they do they do not last forever
    - This means you can't 'go back' several versions, especially versions that are several months or years old
- Cloud services can't 'diff' all files, showing changes from one version to another
    - This is something that is useful to be able to do with code

Version Control Systems

- Programmers use a special type of software to manage changes to their code: Version/Source Control Systems
- These systems track changes to files. Additionally, they are aware that they are working with code which allows for smart behaviour
  - Keeps a 'history' of your code
  - Allows you to go back in time to an earlier version if something goes wrong
  - Lets you see exactly what changed between versions
- These systems can also be used to backup and share your code
  - They can connect to remote repositories
  - Your changes can be sent to configured 'remotes'

Git

- We will be using 'Git' in this course
  - Most widely used Distributed Version Control System (DVCS) in use today
  - Many commands in common with other DVCS
  - Git knowledge is desired in industry
- Git is primarily a command-line based program, but there are a number of graphical front-ends available
  - We will be working with the command-line initially
  - As the course progresses we will start using an IDE integration
  - You are welcome to use other graphical interfaces if you choose to do so

Command-line Git

- As mentioned, git is primarily a command-line application
  - Consists of a suite of subcommands
- Command-line git consists of a suite of subcommands. These subcommands take the form of
  - `git <subcommand> <parameters>`

- `git help <subcommand>`
  - Gives you help with a specific subcommand
  - If no argument is given, a list of common commands is shown
- `git help commit`
  - Using the subcommand `help` to get information on the `commit` subcommand

Your First Git Commands

- `git config <scope> <setting> <value>`
  - Lets you configure git behaviours, and lets git know who you are
    - `git config --global user.name "Cameron Grout"`
    - `git config --global user.email cameron.grout@waikato.ac.nz`
  - The `user.name` and `user.email` options **need to be set**. If they are not, git will prevent you from performing certain actions until they are

- `git config --list`
  - View all set options in order of system ◗ global ◗ local

Your First Git Commands

- `git init`
  - Creates a new repository in the current directory
  - Can be used in an empty directory, or one with existing files. It will not harm any of your files

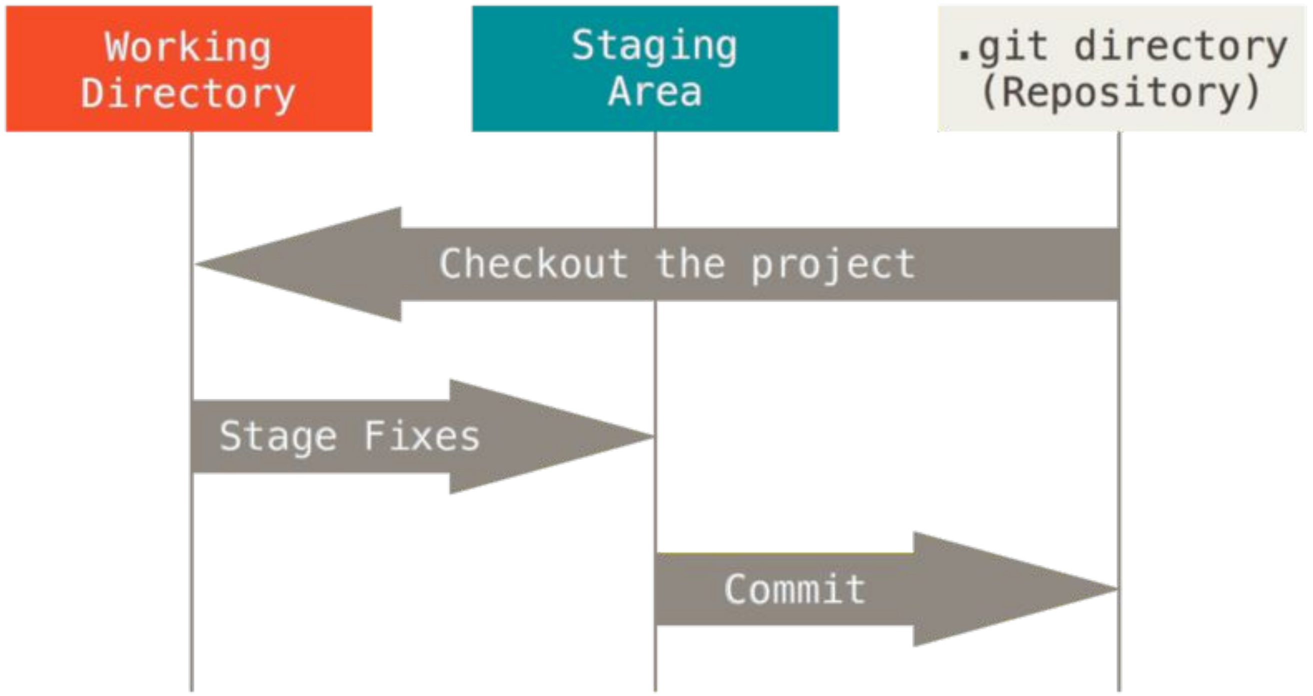
- `git clone <uri> <directory>`
  - Retrieves an existing repository from the path supplied in the `uri` and places the files in the specified `directory`
  - `directory` is optional, if not supplied the repository will be placed in a new folder in the current directory with the same name as the repository

Understanding Your Repository

A git repository (repo for short) has 3 'areas' where your files can be during their lifecycle

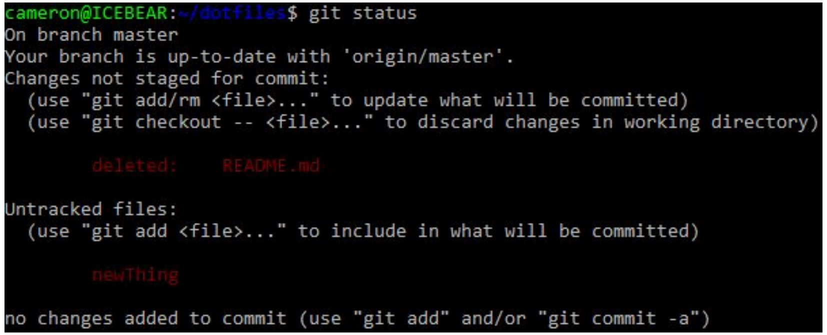
- The Working Directory is what you see in your folders and editors. This represents the current state of your files
- The Staging Area is where files are held for pending operation
- The Repository is where the history of your files are kept

Understanding Your Repository



Working With Your Repo

- `git status`
  - One of the most-used subcommands
  - Tells you what is going on in your repository
  - Are there staged or unstaged changes
  - Have files been added or deleted
  - What branch is checked out
  - Are you up to date with other repositories



```
cameron@ICEBEAR:~/dotfiles$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        deleted:    README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        newThing

no changes added to commit (use "git add" and/or "git commit -a")
```

Tracking and Staging Files

- `git add <parameters>`
  - Tracks new files and adds modified files to the Staging Area
  - Can supply files, directories or patterns for the parameter
- `git add '*.java'`
  - Adds any new or modified files that end with the `.java` extension to the stage
- `git add --all`
  - Adds all new or modified files to the stage
- `git add -u`
  - Add all modified files to the stage
- `git add -p <filename>`
  - Add select parts of a file to the stage rather than the whole thing

Removing and Unstaging Files

- `git rm <parameters>`
    - Remove a file from the repository
    - Can supply files, directories or patterns for the parameter
- `git rm text.txt`
    - Removes the file `test.txt` from both the repository and the Working Directory
- `git rm --cached test.txt`
    - Removes the file `test.txt` from the repository, but not from the Working Directory
- `git rm -r subdirectory`
    - Remove the directory `subdirectory`, as well as any other files or folders within it
- `git reset HEAD test.txt`
    - Removes `test.txt` from the stage, without modifying the Working Directory

Committing Changes

After adding some changes to the Staging Area, the commit command is used to move the staged changes into the Repository

- `git commit <arguments>`
  - Git will open the default text editor and ask you for a commit message. Save and close the editor to continue
  - This commit message will be attached to your changes, and acts as notes for future reference
  - Staged changes will be stored in the Repository
- `git commit -m "My commit message"`
  - Provide the commit message inline, rather than going through the text editor

The Basic Git Workflow

With basic commands out of the way, we can outline a basic git workflow

1. Initialize a repository with `git init`, or obtain an existing one with `git clone`
2. Create or modify files
3. Track and stage new files or changes using `git add`, or remove files with `git rm`
4. Commit your changes with `git commit`
5. Loop back to #2

The Basic Git Workflow

- Be reasonably granular with your commits
    - Try to commit after each logical step in your development process
    - Try to find a happy medium between single line and entire file changes

- Try to have commits be about a single task or topic
    - You should be able to explain the main thrust/intention of the commit in a single line (80 characters) of a commit
    - If explaining your commit this concisely is too difficult, then you probably have too much in the commit
    - Provide as much information as you like for a commit, there is no character limit and it can be really helpful to read them back in the future

So far, so good, but what does using git give you that just zipping up a copy of the code folder after each change doesn't?

- Built-in commit messages
  - Could imitate this with a 'change log' file that you manually update
- Less disk space wasted
  - Git stores 'diffs' or 'deltas' that store just the changes in the file, not the whole thing with each commit. This saves a lot of space over time
- Built in functions in git allow you to explore and manipulate the history
  - Help you work out where bugs were introduced
  - Find exactly when you made a change and undo it automatically
  - More advanced than we want to cover today, we will revisit this later

Backup and Sharing With Git

- Your git repository can point to other 'remote' git repositories
  - These can be on your local computer, another computer on your network or even on a server somewhere in the world
  - You can perform operations on these remote repositories
  - This can be extremely useful if you want to publish your changes, or get changes from others
- To configure these in git, the `remote` subcommand is used
- `git remote -v`
  - Show a list of all configured remote repositories
- `git remote add <alias> <uri>`
  - Add a new remote, located by the `uri`, provided with an `alias` so that you don't need to remember the whole `uri` each time

Communicating With Remotes

- `git push -u <remote-alias> <local-branch>`
  - Send the changes from your `local-branch` to the remote repository identified by the `remote-alias`
  - The `-u` indicates that if a branch with the same name is not found at the remote uri, one will be created and associated with the `local-branch`
- `git pull <remote-alias> <remote-branch>`
  - Get changes from the `remote-branch` identified by the `remote-alias`, and merge the changes with the locally checked-out branch
- `git fetch <remote-alias> <remote-branch>`
  - Get changes from the `remote-branch` identified by the `remote-alias`, without merging
- `git merge <remote-alias>/<remote-branch>`
  - Merge fetched changes from the `remote-alias`'s `remote-branch` with the locally checked-out branch

Remote Source Hosting

- Hosting your own remotes can be tricky, but there are a number of options available to you
  - GitHub, GitLab and Bitbucket are three popular choices
  - All allow you to create remote repositories where you can push your code and share with others
  - GitLab and Bitbucket allow you to make these repositories private, and provide fine-grained access control
- We will be using Bitbucket for this course
  - Sign up at https://bitbucket.org/account/signup
  - Use your university email address
  - Remember your credentials, you will be using this for the rest of the course

Recommended Reading

W3schools HTML Tutorial

W3schools tag reference

HTML, CSS and JavaScript (Meloni, 2015) Chapter 2

Chacon, S & Straub, B. 2014. Pro Git. Chapters 2-4

tryGit - An interactive git tutorial