**Web Developers**

* Build and maintain websites
* Often work for clients who are trying to get their product or service onto the web.

**Types of Web Developers**

1. **Front end** – is the stuff you see on the website in your browser, including the presentation of content, and user interface elements like the navigation bar. Front-end developers use HTML, CSS, JavaScript, and their relevant frameworks to ensure that content is presented effectively and that users have an excellent experience. It is the part that users interact with. Everything that you see when you’re navigating around the internet, from fonts and colors to dropdown menus and sliders, is a combo of HTML, CSS, and JavaScript being controlled by your computer’s browser

* A framework is defined as “basic structure underlying a system”. It provides support and basic “guide” of the structure being built.

1. **Back end** – refers to the guts of the application, which live on the server. The back-end stores and serves program data to ensure that the front end has what it needs. This process can become very complicated when a website has millions of users. Back-end developers use programming languages like Java, Python, and Ruby to work with data. The back-end of a website consists of a server, an application, and a database. Back-end developers builds and maintains the technology that powers those components which, together, enable the user-facing side of the website to exist in the first place.
2. **Full-stack** ­– are comfortable working with both the front and back ends.

**HOW DOES THE INTERNET WORK?**

**INTERNET**

* It is the backbone of the Web, the technical infrastructure that makes the web possible. At its most basic, the Internet is a large network of computers which communicate all together.
* It is a way to connect computers all together and ensure that, whatever happens, they find a way to stay connected.
* The internet is a wire. It might be fiber optics, copper or disk satellites, or through cellphone networks. But internet is simply a wire.
* The *Internet* is an infrastructure, whereas the *Web* is a service built on top of the infrastructure.

**ICANN**

* Internet Corporation for Assigned Names and Numbers is an American multistakeholder group and nonprofit organization responsible for coordinating the maintenance and procedures of several databases related to the namespaces and numerical spaces of the internet, ensuring the network’s stable and secure operation.

**ROUTER**

* It is a special tiny computer. Its job is like a signaler at a railway station. It makes sure that a message sent from a given computer arrives at the right destination computer.

**MODEM**

* It is a device to connect our network to the telephone infrastructure. It turns the information from our network into information manageable by the telephone infrastructure and vice versa.

**INTERNET SERVICE PROVIDER (ISP)**

* ISP is a company that manages some special routers that are all linked together and can also access other ISPs’ routers. So the message from our network is carried through the network of ISP networks to the destination network.

**IP ADDRESS**

* Internet Protocol Address is unique address that identifies any computer that is linked to a network. It is an address made of a series of four numbers separated by dots.

**INTRANETS**

* These are private networks that are restricted to members of a particular organization. They are commonly used to provide a portal for members to securely access shared resources, collaborate and communicate.

**EXTRANETS**

* These are very similar to Intranets, except they open all or part of a private network to allow sharing and collaboration with other organization. They are typically used to safely and securely share information with clients and stakeholders who work closely with a business.
* Both *Intranet* and *Extranet* run on the same infrastructure as the *Internet*, and use the same protocols. They can therefore be accessed by authorized member from different physical locations.

**WEB PAGE**

* It is a document which can be displayed in a web browser such as Firefox, Google Chrome, Opera, Internet Explorer, or Safari. These are also often called just “pages.” Such documents are written in **HTML** language.
* A webpage can embed a variety of different resources such as:
  + Style information – controlling a page’s look and feel
  + Scripts – which add interactivity to the page
  + Media – images, sounds, and videos.

**WEBSITE**

* It is a collection of web pages which are grouped together and usually connected together in various ways. Often called a “web site” or a “site.” Web pages share a unique domain name. Each web page of a given website provides explicit links – most of the time in the form of a clickable text – that allow the user to move from one page of the website to another.
* To access a website, type its domain name in your browser address bar, and the browser will display the website’s main page or *homepage* (casually referred as “the home”)

**WEB SERVER**

* It is a computer that hosts a website on the Internet. “Hosting” means that all the web pages and their supporting files are available on that computer. The web server will send any web page from the website it is hosting to any user’s browser, per user request.
* Web servers can host multiple websites. They are never used to designate a website, as it could cause great confusion.

**SEARCH ENGINE**

* It is a web service that helps you find other web pages, such as Google, Bing, Yahoo, or DuckDuckGo. These are normally accessed through a web browser (Firefox, Chrome) or through a web page. (bing.com or duckduckgo.com)
* Don’t confuse the infrastructure (browser) with the service (search engine).

**TCP/IP**

* **Transmission Control Protocol and Internet Protocol** are communication protocols that define how data should travel across the internet. It is a set of rules and procedures that function as an abstraction layer between internet applications and the routing and switching fabric.
* It specifies how date is exchanged over the internet by providing end-to-end communications that identify how it should be broken into packets, addressed, transmitted, routed and received at the destination.

**DNS**

* Domain Name servers are like an address book for websites. When you type a web address in your browser, the browser looks at the DNS to find the website’s IP address before it can retrieve the website. The browser needs to find out which server the website lives on, so it can send HTTP messages to the right place.

**HTTP**

* Hypertext Transfer Protocol is an application protocol that defines a language for clients and servers to speak to each other.
* It is a protocol for fetching resources such as HTML documents. It is the foundation of any data exchange on the Web, and it is a client-server protocol, which means requests are initiated by the recipient, usually the web browser. A complete document is reconstructed from different sub-documents fetched, for instance, text, layout description, images, videos, scripts, and more.

**COMPONENT FILES**

* **Code files**: Websites are built primarily from HTML, CSS, and JavaScript.
* **Assets**: This is a collective name for all the other stuff that makes up a website, such as images, music, video, Word document, and PDFs.

**PACKETS**

* It describes the format in which the data is sent from server to client.

**COMMAND LINE**

**PRESENT WORKING DIRECTORY (pwd)**

- It will list the “logical” path of your current directory. This means it will treat symlinked paths as if they were actual paths.

**LIST FILES AND DIRECTORIES (ls)**

- It is often useful to be able to see what files or directories are in your current directory, or another directory. To accomplish this we can us the ls command.

**LIST ALL FILES (-a)**

- It is common on Unix and Linux based machines to have “invisible”, or “hidden”, files that are prefixed with a “.” (dot). Using -a flag we can see both “visible” and “hidden” files and directories.

**LONG FORM LISTING (-l)**

- One of the most useful flags for ls that will list out the names of the files and directories as well as give more detailed information about them.

- If the -l option is given, the following information is displayed for each file: file mode, number of links, owner name, group name, number of bytes in the file, abbreviated month, day-of-month file was last modified, hour file last modified, minute file last modified, and the pathname.

**HUMAN READABLE SIZES (-h)**

- It gives us more useful informa tion as it gives human readable terms, such as 1.7K or 35M. (ls -lh)

**SORTING BY SIZE (-S)**

- It is a flag which will sort the result of ls by file size, instead of the default sorting by name. (ls -lhS)

**SORTING BY LAST MODIFIED TIME (-t)**

- It can often be useful to be able to sort your ls results by the time the files were modified.

**REVERSE SORT (-r)**

- It is used with other options to reverse their sort order. (ls -lSr)

**LINKS (ln)**

- It allows us to create an association from one file or directory to another. It allows us to “point” one file or directory to another.

- Using links can create a link called “current” that your web server points at, and you can change where the “current” link points to.

**HARD LINKS**

- By default, the ln command will create a hard link between these files.

- It creates an identical copy of the linked file on disk that gets updated automatically as the source file gets updated. This means if the content of the source is change, so will the target file.

- To create a link to a file, we call the ln command – first giving it the name of the file we want to link to (our source file), followed by the name of the linked file we want to create (the target).

**FORCING A LINKS (-f)**

- If the source file already exists you will get an error. To fix the error, you can pass the -f flag to force the link.

**SYMBOLIC LINKS (-s)**

- It is used to create a link to a directory. This can also be used for linking to files as well, not just directories.

- These can also link to files or directories on other file systems. This makes symlink more powerful, and more common that the default hard link.

- With Symbolic link, the operating system creates a new, small file that points to the target file or directory.

- The linked file can’t exist without the source file.

**CHANGE DIRECTORIES (-cd)**

- It is used to change or navigate, into another directory. It takes one argument – the path to the directory you wish to navigate to.

**NAVIGATING UP (..)**

- (..) allows you to move up to the parent directory.

**NAVIGATING TO THE HOME DIRECTORY**

- When you are in a directory and want to navigate back to your home directory, you can call cd without any arguments.

**CREATING DIRECTORIES (mkdir)**

- To create a directory we can use the mkdir command followed by the name of the directory we wish to reate.

**CREATING INTERMEDIATE DIRECTORIES (-p)**

- The mkdir command will allow us to create nested directories using the -p flag. (mkdir -p a/b/c)

**VERBOSE OUTPUT (-v)**

- Adding -v flag will print the results of mkdir console.

- It can provide useful reporting when writing scripts that will execute a lot of commands for you.

**COPYING FILES (cp)**

- Copying files and directories can be accomplished using the cp command.

- To copy a file, we can call the cp command with two arguments. The first argument is the file we want to copy, the source file. The second argument, or target, is the location we want to copy the source file to. This target can either be the name of the file you wish to copy to, or the directory you wish to copy to. If you specify just a directory then the original file name of the source file will be used to create the new file in the target directory.

**COPYING MULTIPLE FILES**

- cp will allow you to list several files you would like to copy. When you do this, however, the last argument MUST be a directory, and the original file names of the source files will be used as the names of the new files in the target directory.

**COPYING DIRECTORIES (-R)**

- By default cp expects the source to be copied to be a file. If we try to copy a directory to another directory we will get an error.

- To fix the error, we can use the -R flag to recursively copy the directory’s contents to the new directory

**FORCE OVERWRITING OF A FILE (-f)**

- If a user tries to copy a file into another user’s file, he will get an error due to lack of permission. When errors such as this occurs, we can use the -f flag to force the opying of the source file to the target file.

**CONFIRM OVERWRITING OF A FILE (-i)**

- When you are about to copy over several files at once, and some of the target files already exist, it may be beneficial to confirm that you do in fact want to copy the source file and replace the target file.

- cp will happily prompt you to confirm that you are about to overwrite another file if you use the -i flag.

- i for interactive.

**DELETING FILES (rm)**

- the rm command is used to delete files and folders. It supports the same flags and arguments as the cp command.

**MOVING FILES (mv)**

- the process for moving files is almost identical of copying files.

- mv supports the same flags as cp. In reality, the mv command is really just a combination of cp and rm to achieve the desired outcome of moving a file from one location to another.

**INPUT/OUTPUT (| , >)**

- With these seemingly small individual commands, we can build some pretty impressive work flows by redirecting the output of one command to the input to another command. This is made possible by the “pipe” operator, |.

- When using the | operator, we can chain together any number of commands.

- In addition to redirecting the output from one process and sending it to another process we can also write that output to a file using the > operator.

**READING FROM A FILE (<)**

- We saw that we can use > to write data to a file. If we instead wanted to read data from a file, we can use <.

**GIT**

- Git is like a really epic save button for your files and directories. It is a version control system.

- Git records differences in the files and folders AND keeps a historical record of each save. As any individual developer, Git enables you to review how your project grows and to easily look at or restore files states from the past. Once connected to a network. Git allow you to push your project to GitHub for sharing and collaborating with other developers.

- The major difference between Git and any other VCS is the way Git thinks about data. Conceptually, most other systems store information as a list of file-based changes. These other systems think of information they store as set of files and changes made to each file over time (commonly described as delta-based version control).

- Git thinks of its data more like a series of snapshots of a miniature file system, With Git, every time you commit, or save the state of your project, Git basically takes a picture of what all your files look like at the moment and stores a reference to that snapshot. To be efficient, if files have not changed, Git doesn’t store the file again, just a link to the previous identical file it has already stored.

- Most operations in Git need only local files and resources to operate – generally no information is needed from another computer on your network.

**VERSION CONTROL**

* It is a system that records changes to a file or set of files over time so you can recall the specific versions later. It allows you to revert selected files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more. Using VCS also generally means that if you screw things up or lose files, you can easily recover.

**LOCAL VERSION CONTROL**

* To deal with issue keeping copies of files, programmers long ago developed local VCSs that a simple database that kept all the changes to files under revision control.
* One of the most popular VCS tools was a system called RCS (Revision Control System). It works by keeping patch sets (that is, the differences between files) in a special format on disk; it can then re-create what any file looked like at any point in time by adding up all the patches.

**CENTRALIZED VERSION CONTROL SYSTEMS**

* To deal with issues people encounter where they need to collaborate with developers on other systems, CVCSs were developed.
* These systems have a single server that contains all the versioned files, and a number of clients that check out files from that central place.
* One of the advantages of CVCSs is that everyone knows to a certain degree what everyone else on the project was doing. Administrators have fine-grained control over who can do what, and it’s far easier to administer a CVCS that it is to deal with local databases on every client.
* This setup also have some serious downsides. The most obvious single point of failure that centralized server represents. If that server goes down for an hour, then during that hour nobody can collaborate at all or save versioned changes to anything they’re working on. If the hard disk the central server is on becomes corrupted, and proper backups haven’t been kept, you lose absolutely everything – the entire history of the project except whatever single snapshot people happen to have on their local machines. Local VCSs also suffer from the same problem – whenever you have the entire history of the project in a single place, you risk losing everything.

**DISTRUBUTED VERSION CONTROL SYSTEMS**

* DVCS (such as Git, Mercurial, Bazaar, or Darcs), clients don’t just check out the latest snapshot of the files; rather, they fully mirror the repository, including its full history. Thus, if any server dies, and these systems were collaborating via that server, any of the client repositories can be copied back up to the server to restore it. Every clone is really a full backup of all the data.
* Many of these systems deal pretty well with having several remote repositories they can work with, so you can collaborate with different groups of people in different ways simultaneously within the same project. This allows you to set up several types of workflows that aren’t possible in centralized systems, such as hierarchical models.

**THE THREE STATES IN GIT**

* **Modified** – means that you have changed the file but now committed it to your database yet.
* **Staged** – means that you have marked a modified file in its current version to go into your next commit snapshot.
* **Committed** – means that the data is safely stored in your local database.

**THREE MAIN SECTIONS OF A GIT PROJECT**

* **Working tree** – is a single checkout of one version of the project. These files are pulled out of the compressed database in the Git directory and placed on disk for you to use or modify.
* **Staging area** – is a file, generally contained in your Git directory that stores information about what will go into your next commit. Its technical name in Git parlance is the “index”, but the phrase “staging area” works just as well.
* You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your **Git directory.**
* If a particular version of a file is in the Git directory, it’s considered committed. If it has been modified and was added to the staging area, it is staged. And if it was changed since it was checked out but has not been staged, it is modified.

**HTML (HYPERTEXT MARKUP LANGUAGE)**

- It is the raw data that a web page is built out of. All the text, links, cards, lists and buttons are created in HTML. It is used to put I nformation on a webpage.

- It is a language that determines how documents and web pages are displayed in a web browser, the language for the building blocks of any website.

- **HTML Tags** are like keywords which defines how web browsers should format and display the content. With the help of tags, a web browser can differentiate a simple text, and HTML text.

- One thing to remember, web browsers read an HTML document from top to bottom, and left to right.

- **HTML Elements** are pieces of content wrapped in opening and closing HTML tags.

- We should always name the HTML that will contain the homepage of our websites **index.html**. This is because web servers will by default look for an index.html page when users land on our websites – and not having one will cause big problems.

**THE DOCTYPE**

- Every HTML page starts with a doctype declaration. The doctype’s purpose is to tell the browser what version of HTML it should use to render the document. The latest version of HTML is HTML5, and the doctype for that version is simply <!DOCTYPE html>.

**THE HTML ELEMENT**

- <html> is what’s known as the root element of the document, meaning that every other element in the document will be a descendant of it.

**THE HEAD ELEMENT**

- <head> is where we put important meta-information about our webpages, and stuff required for our webpages to render correctly in the browser. Inside the <head>, we **should not** use any elements that displays content on the webpage.

**THE TITLE ELEMENT**

- One element we should always include in the head of an HTML document is the title element.

- <title> is used to give webpages a human-readable title which is displayed in our webpage’s browser tab.

- If we didn’t include a title element, the webpage’s title would default to its file name. (ex. index.html, about.html, etc.)

**THE CARSET META ELEMENT**

- Another important element inside <head> is <meta charset=”utf-8”>. Setting the encoding is very important because it ensures that the webpage will display special symbols and characters from different languages correctly in the browser.

**BODY ELEMENT**

- <body> is where all the content that will be displayed to user will go – the text, images, lists, links, and so on.

**PARAGRAPHS**

- <p> is used to create paragraphs in HTML, it will add a newline after each of our paragraphs.

**HEADINGS**

- They are different from other HTML text elements: they are displayed larger and bolder than other text to signify they are headings.

- There are 6 different levels of headings starting from <h1> to <h6>. The number within a heading tag represents that heading’s level. <h1> is the most important and is larger than the other headings, and h6 is the lowest level and therefore the smallest of all the headings.

**STRONG ELEMENT**

- <strong> element makes text bold. It also semantically marks text as important; this affects tools, like screen readers, that users with visual impairment will rely on to use your website. The tone of voice on a screen reader will change to communicate the importance of the text within a strong element.

**EM ELEMENT**

- <em> element makes text italic. It also semantically places emphasis on the text, which again affect things like screen readers.

**HTML COMMENTS**

- These are not visible to the browser; they allow us to *comment* on our own code so that other developers or our future selves can read them and get some context about something that might now be clear in the code.

- To write an HTML comment: simply put <!-- and --> at either end of the comment.

**CSS (CASCADING STYLE SHEET)**

- CSS is what adds style to HTML’s plain elements. It is also used to position the information given by HTML, give it color, change the font, and make it look great.

- It is a popular style sheet language that determines how a documented created in HTML is styled (colors, font styles, layout, and responsive features).

- It targets various screen sizes to make web pages responsive.

- It primarily handles the “look and feel” of a web page.

**JAVASCRIPT**

- It allows you to change CSS and HTML elements on your website after the site has been loaded, which gives you the ability to make your site more interactive and engaging for users.

- It adds interactivity to a web page. It is a programmatic code which enhances functionality and handles complex features as well.