Final Study Guide

Monday, May 01, 2017 1:35 AM

Git: Local Version Control

- Git is used to track evolution of a software project. It is used for nonlinear, non-monotonic development, which means that multiple versions can be made at the same time, and you can revert to previous versions. It helps with team based projects.
- A git project is split up into 3 sections, and all together are called a repository:
    - A working tree is the project itself, AKA the code and folders and resources.
    - A store is the history of the projects, like the log of all changes made.
    - The index is a virtual snapshot of the project. This is where your most recent code change sit, and is a gateway for moving changes from the working tree into the store. The history is made up of a bunch of saved indices called commits. This history is represented as a DAG.
- A commit is identified by a 160 bit hash, which *should* be unique. So every commit in the DAG is a node identified with a hash.
- A git branch is a pointer to a specific commit node within the history DAG. The HEAD is a special reference and usually points to a branch. We think about the HEAD as being attached to a particular branch. It is supposed to refer to the commit you are building new code off of in the working tree. A detached HEAD is a reference to a specific commit not labeled with a branch. This usually occurs when making a change to a remote commit.
- A clean repository has an identical HEAD, index, and working tree.
- When you make changes in your working tree, you can add them to the index using git add.
- When wishing to create a new commit node within the DAG, you use git commit. This will take the changes you put in the index, and put them into the store. The branch and HEAD pointers will move to this new commit node!
- Creating a new branch will point a new branch pointer at the current HEAD. When using git checkout, the HEAD will attach to this branch pointer. You should only move the HEAD/checkout when your working tree is clean.
- Moving things from your index to the store/committing on an earlier branch pointer will create a sort of "branch" in the DAG from that old commit node. Its changes will not be affected by your other updates on the other branch.
- Git merge is a tool to bring work from one branch into another. It will update the current branch HEAD is looking at, not the branch you are merging with.
- There are three cases for merging:
    - Fast Forward Merge - This occurs when HEAD is pointing at an ancestor you want to merge with. The branch HEAD is attached to will just point to the merging branch/node.
    - Non-Ancestor With No Conflicts - This will occur when the branch HEAD is attached to has a commit node with no updates that would edit code in the branch/node you are merging with. A new node will be committed to the store, making it the child of both branches, and then the branch you are merging with and HEAD will point at the new commit/node.
    - Above, but with Conflicts - This happens when the branch HEAD is attached to DOES have updates that would edit code in the branch/node you are merging with. The first thing that will occur is that your index will be populated with files that WILL merge with no conflicts, and the second thing that will occur is that your working tree will be edited, so that the conflicts will be marked. You have to choose which code to go in, add it to your index, and then commit it. This commit will do the same as the second case from here on.

Git: Distributed, Remote Version Control

- Since Git is distributed version control, multiple programmers can code across a network. Everyone has a local repository, with their own store. Changesets are the units of data movement,

which are commits/nodes received from different users. We wish to have an idea of how all the changesets are organized so stores of each team member can be synchronized.
- Git fetch can synchronize stores. When the git fetch operation is called, all changesets from the specified remote repository to fetch from will be put into our local store. The ancestor of these new changesets (just commits/nodes) will be the commit/node that was last identical between our repo and the remote repo.
- A git pull will fetch changesets from a remote repo, and then merge your HEAD branch with the remote changes. You should fetch and merge yourself, instead.
- Git push will send your local commits to the remote store. You should usually only push one branch at a time. There are a couple of requirements and common practices:
    - Requirement - The branch you are pushing to the remote repo must not be the HEAD branch. This is so we don't overwrite anybody's work!
    - Requirement - The remote branch must be an ancestor of your branch.
    - Practice - You should only push to bare repositories.
    - We should get the remote stores branch into our local store using fetch, merge, and commit before pushing.
- More commands to use:
    - Git init - Initializes a repo on your local machine.
    - Git clone - Copies a remote store into a local store on your local machine.
    - Git status - Displays everything changed in your working tree and index.
    - Git log - Displays list of commits with hash values.
    - Git rm - Removes a file from index.
    - Git reset --hard HEAD~1: Resets the HEAD to the previous commit/node. Destroys node you are resetting from.

Ruby

- Ruby is an object oriented programming language, and it has classes, instances of objects, and inheritance. It is strongly typed, meaning that only specific operations can be done on specific classes.
- No parentheses or semicolons are needed.
- Ruby is interpreted, meaning that the programs do not compile. They are run with a program called an interpreter. Variables are dynamically typed, meaning that they do not have types, while objects do. So a string, int, or float have no types, but an array or hash does.
- EVERYTHING is an object.
- Since Ruby is interpreted, it can be slower than compiled languages like C++, and errors are usually only found at run time, not during compile time.
- Literals are numbers, strings, ranges, arrays and hashes in Ruby.
- A range in ruby determines an ordered list of objects. (..) Includes the start and end value, (…) does not include the end value.
- Comments start with #, multiline comments begin with =begin and end with =end. A convention is to use #=> to display the result of execution of a line of code.
- Operators:
    - + - * / %: Standard math, division rounds towards -infinity.
    - **: Squares a number.
    - ~ | & ^ << >>: Standard bitwise operators.
    - Comparison: < > <= >= <=> The last operator returns -1/0/1 is the LHS is smaller/equal/larger than the RHS.
    - && || ! And or not: Standard logical operators, and/or/not more rubylike.
- Special words:
    - The word self is an object that is the reciever of the current method. Think this in Java.
    - The word nil is an object like NULL in Java.
    - Booleans are true and false, 0 is *not false*.
    - __FILE__ and __LINE__ return the file and line names/number respectively.
- A variables name affects its semantics:

    - Local variables: Starts with lowercase letter or an underscore (_).
    - Global variables: Starts with $.
    - Instance variables: Start with @.
    - Class variables: Start with @@. A class variable is akin to a static Java variable.
    - Constants start with an uppercase like Size, but the Good™ way to do it is to make it ALLCAPS.
- Conditionals:

    Classical:

```
if(boolean)
…
elseif
…
else
…
end
```

    Contemporary:

```
if boolean
…
end
```

    Enlightened:

```
x = y if x > LIMIT
```

    Unless:

```
unless size >=10 #Means if not. So really means if size < 10.
…
end #Can also be done like "x=x+1 unless size >=10"
```

    Case:

```
variableName case expression

when nil
#if expression is null
when value
#if expression evaluates to value
when type
#executes when expression results in type
when /regexp/
#executes if expression matches regex
when min..max
#executes when expression is in range
else
#if all else fails…
end
```

- Conditionals do not create a nested lexical scope, so a variable created inside one will not be usable outside of the conditional.
- Iteration:

    Classic:

```
while boolean [do]
…
end
```

also

```
expression while boolean #WARNING: The expression will always
execute once.
```

also

```
until variableName > otherVariable
…
end
```

- In ruby, functions are called like foo(arg1, arg2), but can be called without the parentheses. In a function, the last statement is returned always. So it's Good to not use the return statement, or use parentheses (unless you are chaining functions in one statement, or passing more than one argument).
- To define a function:

```
def functionName(args)
    ...
end
```

- A Ruby object has an object ID, the value of its reference, and an object value, the actual info it holds. The == operator compares object values, while .equals? compares object IDs. This ID is a 4/8 byte number.
- Objects can have methods/functions called on them, using a dot operator.
- Aliases still exist. Numbers, true, false and symbols are immutable. But strings, lists, and other objects are mutable. No ++, -- operators exist.
- Parallel assignment can be done, like x, y = 10, 11.
- The method .to_a converts a range to an array. (0..100) makes an array with 101 indices. === tests if a number is in a range, like (0..10) === 5 is true. include? checks if a number is in a range, cover? checks if a subrange is in a range.
- You may have noticed, but a ? returns a boolean value, and a ! mutates an object directly. On ruby strings, empty? returns true if the string is empty, start_with? checks the first character, include? checks for a substring, length returns the string length, to_f converts it to a float, to_i converts it into an int, split converts it into an array based on a specified delimiter ("www.thisisalongstudyguide.com".split('.') => [www, thisisalongstudyguide, com])
- Arrays are instances of the Array class, and can be declared like x = Array.new 4. This will make a 4 index array. The Good way is to do x = [array_size]. Array.length will return the length, assigning an index outside the array size will dynamically resize it. Push will put something in the last index, pop will remove it. Unshift will put numbers at the start, and shift will return the first number. Concat will add another array onto the end, push can do the same but one index will be an entire array at the end. Sort will sort it.
- In Ruby, a block is a statement passed into a function as an argument. A block itself can have parameters, placed like |variableName|. An example is

```
5.times do { puts 'Hello!'} #Prints hello 5 times.
```

- When writing a function that takes a block as a parameter, the keyword yield will run the code within the block.
- For loops use blocks, like for str in strList {puts str}
- Array.each will perform code in the block on each element, a.each_index will do something with each index, fill will fill an array with values from the block, index will search the array, and sort will do a custom sort. Map will take every value and create a new array with the elements having the block enacted on them.
- A hash is a map of keys to values. Keys must be unique. Initialized with literal values like h = {'re'

=> 2} or Hash.new.
- To access a value, use h['re']
- Hash.size returns the size, delete removes a pair, each returns every pair, each_key/value returns those, merge combines 2 hashes, delete_if works on both arrays and hashes, deletes based on a conditional statement within the block.
- Keys are immutable.
- Symbols in Ruby are just immutable strings without all the methods. Defined like :symbolName. All uses of a symbol only refer to one object.
- Symbols are better to use as keys in hashes. In a literal definition of a hash, do h = {red: 'red'} To use a symbol instead of another value as a key.
- Ruby classes have methods and variables, remember that @ is an instance variable for Ruby and @@ is a class/static variable. Instance variables can be made private by designating that before the variable.
- To create a getter/setter method for an instance variable, use attr_accessor :variableName, where variableName is a symbol.
- There is no overloading in ruby, as methods are identified with symbols. You can have default arguments by specifying those in method parentheses.
- Classes are defined like:

```
class ClassName < Device
…
end
```

- This class has name ClassName. Class names are always first letter capitalized, and the class inherits from Device. Using the lowercase keyword super will call the parent methods class.

HTML: HyperText Markup Language

- This markup language is used to connect documents via links. It describes the content of the document, not the style.
- HTML is design by contract, which has strong insurance on what the language will do, but weak requirements for interpretation. This means browsers are permissive, and will display malformed HTML and make a mess of things. To ensure your HTML isn't garbage, use validator.w3.org, which will check for syntax problems. Helps if you are writing a website by hand.
- HTML is declared in a file by saving it as HTML filetype, and using the HTML 5 (latest version of HTML) declaration: <!DOCTYPE html>. Older versions of HTML have different, more detailed declarations.
- There are many element tags. They are opened like <elementName> and closed with </elementName>. Some examples are:
    - <head> - Info appearing in the header of the document.
        - <title> - The webpage title in the browser.
        - <meta charset> - The types of characters the document uses.
        - <script> Some code to run on the client side, like JS.
        - <link> Other documents to use, like a CSS file.
    - <body> - The main content portion of the document. Some block elements:
        - <p> - A paragraph of text.
            - Plain text within a <p> tag
            -
 A line break.
            - <h1> to <h6> - Large header text.
            - , <ul> - Numbered, bullet point list respectively.
                - - An element in the list
            - <table> A table.
            - <form> - A form with form elements.
            - - Generic container for content.
- An HTML entity is defined like: &#hhhh for a code point in hex.
- HTML follows a tree like structure. The root of the tree is always the HTML element. The two children are the head and the body, and the subtrees of these compose the content on the page.

The head will usually contain meta information about the document.

- The body will contain two types of elements:
    - Blocks, which are content that stands alone. They start new lines of text, and can contain more blocks or our next element, inlines.
    - Inlines, which are part of the surrounding context, do not interrupt the flow of the text, and can contain other inlines.
- The best way to think about it is that a block can contain a large portion of the page, like inlines, paragraphs, images and other things, and inlines are things to be used on one line of the document.
- An HTML5 document must have the following structure:

```
<html>
    <head>
        <title></title>
        <meta charset = "utf-8></meta>
    </head>
    <body>
    </body>
</html>
```

- There are also common inline elements:
    - <a> - A link, use URLs like <a href = "URL.com">
    - <img> - An image, uses URLS like a tags.
    - , - emphasis and strong emphasis.
    - <ins>, <del> - Inserted and deleted text.
    - generic inline container for content.
    - Elements within the Block <form> tags.
- Comments are done like <!-- comment --> and do not nest.
- To format a table, use these elements:
    - <tr> - A table row.
    - <td> - a Cell of data in the table.
    - <th> - A table header for a row or column.
    - <caption> - a caption

Networking Fundamentals

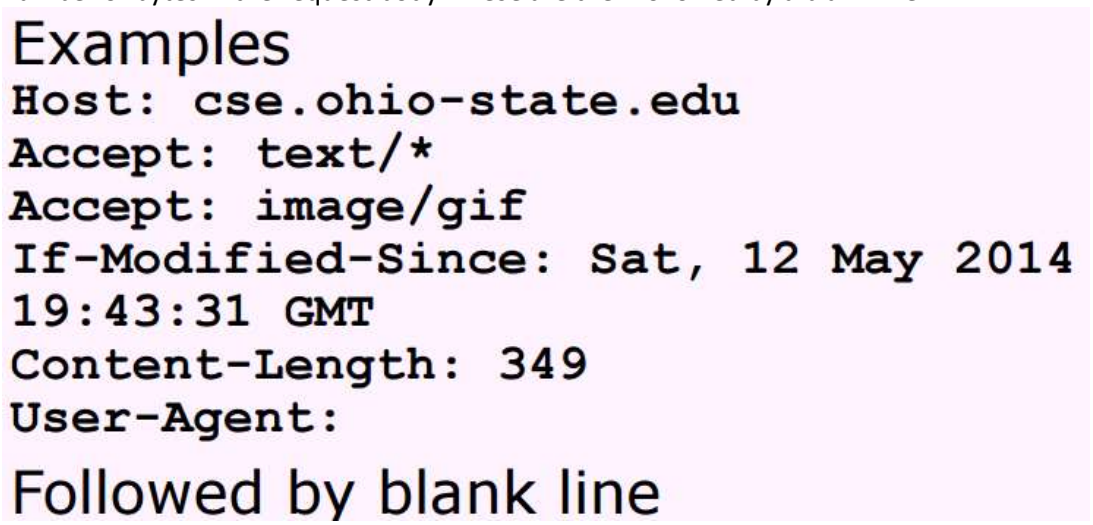
- An IP address is a unique 32 bit number, and is assigned to a device connected to the internet. Packets sent from end users through routers, ethernet switches, and the overall network core use this address at every router in the network core to determine where to send the packets next.
- Its written in dotted decimal notation, divided into 4 fields separated by periods. Each field is 8 bits.
- 127.0.0.1 is the localhost IP address your machine uses to refer to itself.
- All of the above described the IPv4 model of IP addresses. As 32 bit numbers have only 4 billion addresses, we cannot have a unique address for the 7 billion people on earth. So we invented IPv6.
- IPv6 is a 128 bit address, and has 10**40 addresses. We'll be fine for a bit.
- Each field in an IPv6 is divided by a colon, there are 8 fields, and each field is 16 bits or 4 hex digits.
- When representing these IPs, we omit the leading 0s in a field. We will also compress consecutive fields of 0's with :: , and we can only compress at most one set of these, the longest sequence we can find.
- Domain names are strings the match up or correspond to a specific IP address. These are case insensitive, and are a partial map, with lower case strings as keys and IP addresses as values. The tech that maps this is the DNS, or domain name service.
- Multiple strings can map to the same address. The domain name hierarchy is seperated by periods, and the top level domain is the right most field, like .com, .org.

- A FQDN, or fully qualified domain name, is the hostname and the domain name.
- A DNS Name Server is given a FQDN, and returns an IP address.
- A protocol is a specific ordering of messages, used to strictly define how two machines should communicate.
- An abstraction of network layering is the TCP/IP stack. It starts with the:
    - Application Layer – FTP, HTTP, SSH. Protocols used to create programs for use in everyday situations.
    - Transport Layer – TCP, UDP. Used to determine how two end users will pack and send data to each other.
    - Network Layer – IP. Determines how to get data through the network core/internet from one end user to another.
    - Link Layer – Uses protocols to send a packet from one network router/node to another.
    - Physical Layer – The actual cable or medium like WiFi used for transmission.
- Each protocol uses and assumes behavior from the layer below it. TCP will create a channel between two end users, and trust the network core to establish that connection, HTTP will assume that a channel has been opened, IP assumes packets get from one router to the next.
- A port is something a webserver or client is always listening to. Since HTTP is on port 80, a server will always be listening for requests on port 80 and send them back through port 80. A host/computer has many ports:
    - FTP – 20
    - HTTP – 80
    - SSH – 22
    - SMTP – 25
    - And many others...
- A URL is a Uniform Resource Locator. It is modeled like:
    - Scheme://FQDN:port/path?query#fragment
        - The scheme is like HTTP, FTP. It selects the protocol to use.
        - The FQDN is the string used by DNS to find the IP.
        - The port is the specific port to send the request to on the other machine. This is optional.
        - The path is the specific document within the folder path specified in the URL. If this path is not specified, the root of the site will be loaded. This root is designated by the person receiving requests.
        - The query portion specifies parameters that can be read from embedded languages on the webpage, or forms on the webpage.
        - The fragment is used by the webpage for multiple purposes. One of these is to remember the position on the webpage to navigate to upon page load.
- MIME stands for Multipurpose Internet Mail Extensions. It used to be used for mail attachments, but is now used to interpret a file, which is just a blob of bytes/bits. Some common MIME types include:
    - Plain/text
    - Plain/html
    - Image/gif
    - Image/jpg
    - Video/mpeg
- The sender/server of a document determines the MIME type of the document being sent. A reciever should get a file with MIME info, but if it does not it has to guess. This guessing can be done using the file extension, and some types are completely handled by the browser.

HTTP: HyperText Transfer Protocol

- HTTP is a stateless protocol, no info is saved between sending and receiving requests. The basic idea is that a client will send a request to a server, and that server responds. Recall we do this on default port 80.

- An HTTP request/response consists of:
    - The header, which holds meta information.
    - The body, which can be empty, that holds the resource we want from the server we requested from.
- The header consists of the method in the request, or the status in the response. It also has header fields and is finished with a blank line.
- The syntax of a request header's first line is: *verb path version.*
    - The verb can be GET, HEAD, POST, PUT, DELETE or others. It tells the server what to do.
    - The path is the path to the document, the path part of the URL.
    - The version is what version of the HTTP protocol to use.
- The header fields in the request each have their own line. The specify a key and a value to send to the server. These can be things like host, the FQDN without the port and other stuff at the end, multiple Accepts, specifying what filetypes will be accepted by the client, and If-Modified, which reports the latest version of the document being requested that is sitting in the browsers cache at the moment of the HTTP request. Content –Length is another header used, which reports the number of bytes in the request body. These are then followed by a blank line.



- The host is the only required field, as it allows the DNS lookup for the correct IP to send the request to over HTTP. Accept describes the MIME types the browser will accept. The if modified is used to check to see if we use the browser cache or not. The content length is the number of the bytes in the response body.
- The syntax of a response header is *http-version status-code text*
    - Http-version is the version of http the server is using to send the response.
    - The status code is what the server is doing in response to the request. Some common server codes are:
        - 200 OK – Everything is okay, response body is the requested document.
        - 301 Moved Permanently – Requested resource is somewhere else.
        - 304 Not Modified – Document has not changed from the date/time in the If-Modified-Since header field in the request. The cached resource in the browser is still the latest version of that resource.
        - 404 Not found – Server could not satisfy the request, blame falls on the client.
        - 500 Internal Server Error – Server could not satisfy the request, blame falls on the server.
    - Generally, 2xx codes mean success, 3xx codes mean redirection, 4xx codes mean client error, and 5xx codes mean server error.
    - Text is just the text from the above codes.
- A response HTTP message sends back the status code, header fields, a blank line, and then the body, which is usually the payload. The header fields are usually the date, server type, MIME type of the body, the characterset the body uses, and the length of the body.

- Multiple request methods exist. Some that exist are:
    - GET requests a specific resource from the server. When passing arguments, this is done using the URL with encoded key value pairs. Each key is seperated from its value using an =, each space is a +, and each key/value pair is separated using a & or ;.
    - PUT updates or creates a resource on the server. This needs to be idempotent.
    - DELETE removes a resource. This needs to be idempotent.
    - POST creates or updates something on the server. It changes the server state. It is more robust and dangerous than PUT, and is therefore not idempotent. The arguments in POST are actually passed in the body, which allows arbitrary length, and arguments are not saved as browser history. Plus, you can pass in XML and JSON as well.
    - Idempotency is the idea that a HTTP method can be called many times without different outcomes.

Regular Expressions

- A language is a set of strings. Each language has a cardinality, or a number of words, and can be defined statically or dynamically.
- A regular expression is a formal mechanism for defining a language. In computer apps, there is not a clear understanding when dealing with distinctions between characters or strings in different contexts. Regular Expressions help with this.
- A literal is a character from the alphabet. These can be regular alphanumeric characters, or things like backslash escaped whitespace.
- In Regex, () is used for grouping, and | is used for choices. So the language defined by something like cat|dog|fish is cat, dog, fish. The language defined/selected by Regex 'R' is all instances of the letter R in a file. (H|h)ello is Hello and hello. We use backslashes to represent escaped characters like parentheses and bars.
- You can also use ranges. So to do a grouping of any character, you can just do [0123456789], or ever [0-9].
- You can negate with Regex, so ^ will NOT select anything after the carat. EX: [^a-z] is any character that is a capital letter.
- Shorthands for character classes exist:
    - \d for digits [0-9]
    - \s for whitespace
    - \w for a word
    - Capitals of all of the above for negations.
- A period (.) matches any character except for newline.
- Repetition is handled in a couple of ways:
    - ? Searches for the preceding character or grouping repeating 0 or 1 times.
    - * searches for the preceding character or grouping repeating 0 or more times.
    - + Is the same as * but searches repeating 1 or more times.
    - {k} searches for the preceding character repeating k times.
    - {a,b} searches for the preceding character repeating a <= k <= b times.
- An FSA is a finite state automata. It is described as an "accepting rule", with a finite set of states, and a transition function based on the next character in the string. The start state is S0, and the set of accepting states is things the string can end on. A string is accepted if you can start on a character in S0, and end in an accepting state.

Additional Ruby Method Slide Paranoia Documentation Because Charlie Dropped Hints And I'm Not One To Take Chances

Splat and The Weird Thing .last() does on a Ruby Range

- The Splat operator split/gather arrays/elements. It is not really an operator.
- If used on the right size of an assignment, it splits up an array. So if you were in the weird situation where you had a number literal, and an array, you could say a, b, c = 1, *[2,3]. I tested this, and if

you do not use splat, b just gets assigned to the array. With splat, b gets assigned to 2, and c gets assigned to 3. A always gets assigned to 1.
- On the left size, it will collect everything including and after the object the variable it would ordinarily be assigned to. So a, b, *r = 1,2,3,4,5,6,7 would normally, without Splat, assign to 1,2,3. With Splat, it assigns to 1,2, [3,4,5,6,7]. Neat.
- If you use splat in a function definition as a parameter, it will gather up as many arguments passed into the function and store them in an array for use in the function. Cool.
- If you pass a Splat variable in as an argument, like a splatted array, it will pass every object in the indices of the array into the function. Tubular.
- This is in the entirely wrong place, but using the .last method on a range excluding the last value in the range still returns that last value that was excluded. But passing a number into that method won't return the excluded value. What the heck, Matz?

Map
- .map is a method used on an array, and it takes a block. This block should define an operation to take on each index of the array. What happens is this: It creates a new array and takes every element from the old array, performs the block action on it, and then tosses that new value into the new array.

Reduce
- Reduce will turn an array into a single value, incorporating one element at a time. It uses a block with 2 arguments, the current object conglomerate/total and the next array element. The value returned by the block should be the next total. If the initial index is not provided in the method, a[0] is the first total.

CSS:
- CSS has gone through multiple iterations, with its MIME type always remaining to be text/css.
- The key idea in CSS is to separate content and style. HTML is the content, CSS is the style. It allows for a single point of control over change.
- It is a declarative language, meaning it describes something, not how to do computation on something. It is a list of rules, a location to find and a style to use there.
- This rule is formatted like so

```
Selector1, selector2{
        Property1: style1;
        Property2: style2;
)
```

- There are many available properties, like:
    - Backgrounds
    - Text and font
    - Borders, margins, and paddings
    - Positioning
    - Dimensions
    - Lists, Tables
    - Other fancy things
- You can embed CSS directly in the HTML, place it in the style element in the <head> section of your html, or use the <link type="text/css" rel="stylesheet" href="filename.css" /> link tag to link to it in your HTML. The last method is best.
- The box model has a couple of separate parts:
    - The block and inlines
    - The border appearance, marked with border
    - The margins and padding, these are transparent and have 4 independent sides. The padding is part of the actual content with the background showing and spaces the content away

from the border, and the margins give space between elements.

- Be careful when spacing margins. Overlapping elements without spacing between the paddings can collapse margins and make things look weird.
- CSS element selectors have inheritance. A child will inherit many properties from a parent by default, but generally, the text related properties are inherited, while the box model related ones are not. If we want to set a global style, we can do it in the root of the tree in the DOM we want to sweep over.
- To identify many elements in the page, use CSS classes. These allow you to give a bunch of elements all of the same style. To ID these in your CSS, use elementName.className {} to have a class based on element type, or .className to select different elements in your DOM with the same class type. To set class type in HTML, have the <class="className"> property in your tag.
- If we have multiple block elements that need to be styled together, we have to either define styles in HTML for every block element in the tags, or use a class on the parent DIV tag that contains those block elements. It is better to use a DIV for this.
- The Div lets you use a logical block element, and can be styled just like any other block element can, while also having different blocks as children with their own styles.
- The Span is the same kind of thing but for inline elements. You could put a span tag with a class around some text in a paragraph, and it would style ONLY that text.
- You can also use ancestry to select elements, using ancestor1 ancestor2 … ancestorN elementName{}.
- There are also weird selectors:
    - To select the child of a class, use .className > childName
    - To select an adjacent sibling of an element, use elementName + elementNameNextTofirstName
    - To select all siblings of an element, use elementName ~ siblingElements
    - To select based on attributes, use elementName.[attributeName="example"]
- The ID is a unique class. There can only be one element with a specific ID name. To set this in HTML use <tag id="idName">. To select this in CSS, use elementName#idName{}.
- Overwrites occur in CSS when selectors match children of wide sweeping parent selectors.
- Multiple selectors can match an element in many ways, with multiple rules with the same selector, an element being part of two different classes, two different ancestors match, or different sources of CSS stylesheets exist.
- Generally, text styles are inherited, and can be overridden by selectors that match the children.
- Different rules change how HTML is actually styled with CSS. These rules are location, importance, specificity, and declaration order.
- There are three sources of CSS rules:
    - Author of document – Directs a style attribute on the element, or in a CSS style sheet.
    - Users – Uses their own CSS.
    - Browsers – Edits CSS to hide stuff, style links automatically.
- The priority for these is Author, then User, then Browser. Preference is usually given to document author, but sometimes users really need control. The !important modifier will allow the user to have the highest priority.
- Within a given category, the most specific rule has the highest priority. The specificity of an selector is a triple (x,y,z): x= # of IDS, y = # of classes, z = # of elements. You compare the x's of both selectors, and choose the selector with the higher number of x. If they are equal, you move onto y, and do the same. The process repeats for z.
- If everything above is equal, then the selector that comes second has a higher priority.
- To specify attributes you do not want changed by a selector and want to retain the value their ancestor has, use inherit as the value and it will use the value from the ancestor selector if it exists.
- To specify you want an element to inherit a property, use the inherit value for that property.

Graphics and Fonts

- Colors are used in fonts images and everything else.
- You can value color in many different ways:
    - Rgb (255, 0, 0) specifies the level of red, green and blue to use.
    - HSL is Hue, Saturation, and Light, and is valued like so: hsl (0-360 (angle on color wheel) ,100%, 50%).
    - Adding a decimal number between 0 and 1 inclusive to the end of these definitions will allow for an alpha channel, or transparency. 0 is invisible, 1 is opaque.
- Color can be thought as a Cube, with black at the origin, and in three different directions determining the color.
- It can also be a wheel, with hue being a degree on the circle, and lightness being the distance from the radius.
- Saturation is how much of that color actually shows up. 50 percent saturation is where you get a pure color.
- Color depth is the number of bits in the representation. 8 bits allow for 256 colors, 24 bits allow for 16,777,216 different colors, which is where we're at now.
- Alpha shouldn't be included in color depth, but represents a 4D space for color, it is the transparency of the color.
- If the image color depth > display color depth, we have two methods:
    - Quantization – Every pixel will get its closest visible color. Allows for banding, or the "I have reposted this from 16,000 other grandmothers on Facebook" Effect.
    - Dithered – Will add noise, junk pixels to help blend quantized pixels.
- An <img> tag has a couple of attributes:
    - Src: location of file
    - Width/height: Area in window to reserve for image
        - Image is scaled to those dimensions
        - These attributes will affect browser flow even if image is hidden.
    - Alt: Allows for display text when img is missing
    - Title: Text that augments the displayed graphic.

- Raster images are stored pixel by pixel, and vectors are stored by a math description of the image. Raster images can be stored as lossy or lossless. Lossy has better compression but lower quality, and lossless has the largest file size and the best quality.
- JPEG is a raster lossy, 24 bit image file good for photos.
- PNG is a raster lossless with variable depth and are good for icons
- GIF is a raster lossy image, 8 bit and sucks. You can use it for frame based animation, and anyone who tells you to avoid it has never visited the splendor that is YTMND.com.
- SVG is a lossless vector. It is a relatively new way to display images.
- Vector graphics scale perfectly, and raster images should be pre-scaled, with width and height attributes of the image tag should match the actual width and height of the image file.
- If you use clear, it requires that side of the element to be clear of floats.
- CSS has units for size. Absolute units are in,cm,mm, pt (point, 1/72 inch)
- A given resolution size can be px (pixels)
- A relative size to a current font can be used for size, with e(letter to size to). You can also scale relative to the parent size using %.
- Fonts are a key part of visual design, with font families/typefaces controlling the visuals. Serif fonts have serifs, sans-serifs do not. Variable space fonts look pretty with different spacing for each character, monospace font have the same space between each letter.
- Common opinions exist for fonts:
    - Less is more, use fewer fonts/sizes. You should usually only use one font, two at most for decoration.
    - Helvetica and Arial are clean but ubiquitous, they are everywhere and boring.
    - Times is hard to read on a monitor, and is good for printing.

    - DO NOT USE F**UCKIN**G COMIC SANS YOU MEME **FUCK**
- If you are not sure if an OS will have your font, in CSS list the alternatives in font-family in decreasing order of preference. Always end with one of the 5 generic fonts, which are
    - Sans-serif
    - Serif
    - Monospace
    - Cursive
    - Fantasy
- If you want to use a font family, use the @font-face{
    Font-family: HandWriting;
    Src: url('FONTNAME.ttf');
  }
- The user agent will dynamically download the font. Beware of copyright info.
- Middleman is a ruby gem, you can initialize a project using middleman init myproj for a project named myproj.
- Why are we doing this? It allows for code reuse and single point of control. We can have common headers and footers.
-

Testing

- **Testing is a process where we increase our confidence in an implementation by observing its behavior. Testing detects the presence of mistakes, not their absence. Large systems should ALWAYS be tested.**
- A test has 3 levels of abstraction in functionality, the idea, and an implementation of that idea. Testing compares this implementation against the **specification**, which defines the desired mapping from input to output. This means that the input goes through the spec, generating the expected output, and the implementation produces the actual output. When testing, we compare these outputs.
- Final values of parameters can change, and sometimes these values are based on the initial value of the parameter.
- There are multiple types of specifications. A functional specification maps each element in its domain to a single element in its range, and a relational spec maps each element in its domain to at least one element or more in its range. This means that functions have deterministic behavior, and relations have nondeterministic behavior.
- When writing a test, we test the specification and use the expected output to test the function.
- A function maps each element in its domain to a single element in its range, and a relation maps each element in its domain to at least one element in its range.
- Tests should be written to break a program, not to show how it works. When a test reveals an error, this is considered a success. A good approach to this is to have someone else test your code, but you can also test your own code.
- In a study from NASA, two groups tested code, the ones who wrote the code and people who had never seen the code. The ignorant testers found way more issues with the code in their tests.
- There are many ways to write good test inputs. We should test boundary conditions, like 0, empty arrays and empty strings. We should test different categories of input like positive, negative and 0 numbers. We should test different categories of behavior, like each menu option, each error message. We should test unexpected input, like last names with spaces and null/nil values. We should test representative normal input, like normal values.
- We can create expected output by hand, which is tedious. We can also do it with another program, which is error prone and might repeat the mistakes in the program you are testing. It is best to work backwards.
- We can also validate the output, like ensuring a sorted list has the same number of elements, and every element is less than the one in the next index. These can be easier to write than the thing we're testing.

- Some dangers with testing exist, like:
  - Expected output is wrong
  - Testing program is wrong
  - These create two dangers:
    - A test passes when it shouldn't, which is dangerous.
    - A test fails when it shouldn't, which isn't a huge deal but annoying.
  - Another danger is if the specification is wrong, which means that your own unit tests will not reveal the problem.

- Unit testing tests individual components in isolation, with UUT meaning unit under test. These often use a test fixture, with configuration, values, and objects being set up before running all the tests.
- There are 2 flavors of unit tests:
  - Black box: Testing based only on the specification, and we do not look at the code.
  - White box: Testing based on code structure. Meaning we can look at the code and write specialized tests to make sure all of it is covered.

- Integration tests have modules tested in combination in order to check the interfaces, and are best done incrementally upon adding new features. There are two types:
  - Bottom-Up: Starts with the most basic modules, and is easy to exercise all features. We would write driver code for higher level modules to test initial lower level features.
  - Top Down starts at the top, or where the entire application is initialized from. This tests interfaces early, and can use stubs for lower level features and modules. Stubs are programs that simulate the behaviors of software components.

- System tests verify that the system as a whole meets requirements and specifications. Alpha are tests done by developers, betas are done by friendly customers, and acceptance is testing done to see if you have an actual product or not.
- In dynamic languages, testing is even more important. An extreme position to take is to assume something doesn't work if it isn't tested.
- Test driven development has devs write tests first, writing tests and watching them fail, and then writing only enough code for tests to pass, and repeating.

JS

- JS has a MIME type of text/JavaScript
- It can be used in browsers, but is being used in settings outside of browsers like on the server side with Node.JS.
- To use a JS script on a webpage, we use the script tag with either <script type="text/JavaScript"> //code </script> tags of a src attribute in the same tag to the JS file we use.
- Common advice is to put scripts at the end of the body, as the browser will block when downloading the script. Don't inline JS.
- The execution environment in JS provides some objects, like the Document and Window objects in HTML files and browser programs.
- The syntax is real nice, with semicolons optional but used, comments like Java, statement and expressions contained within brackets and parentheses.
- All the operators you loved in Java are back with a vengeance.
- JS has primitives and Reference types, primitives being Booleans, numbers, strings, null and undefined. Reference types being arrays, objects, functions. Different from Ruby, where everything is a reference type to an object.
- Primitives are assignment by copy, so var a = 5; var b = a; b++; Still has a =5.
- Primitives are pass by copy to functions.
- You can use == to compare primitives, but not references like in Java.

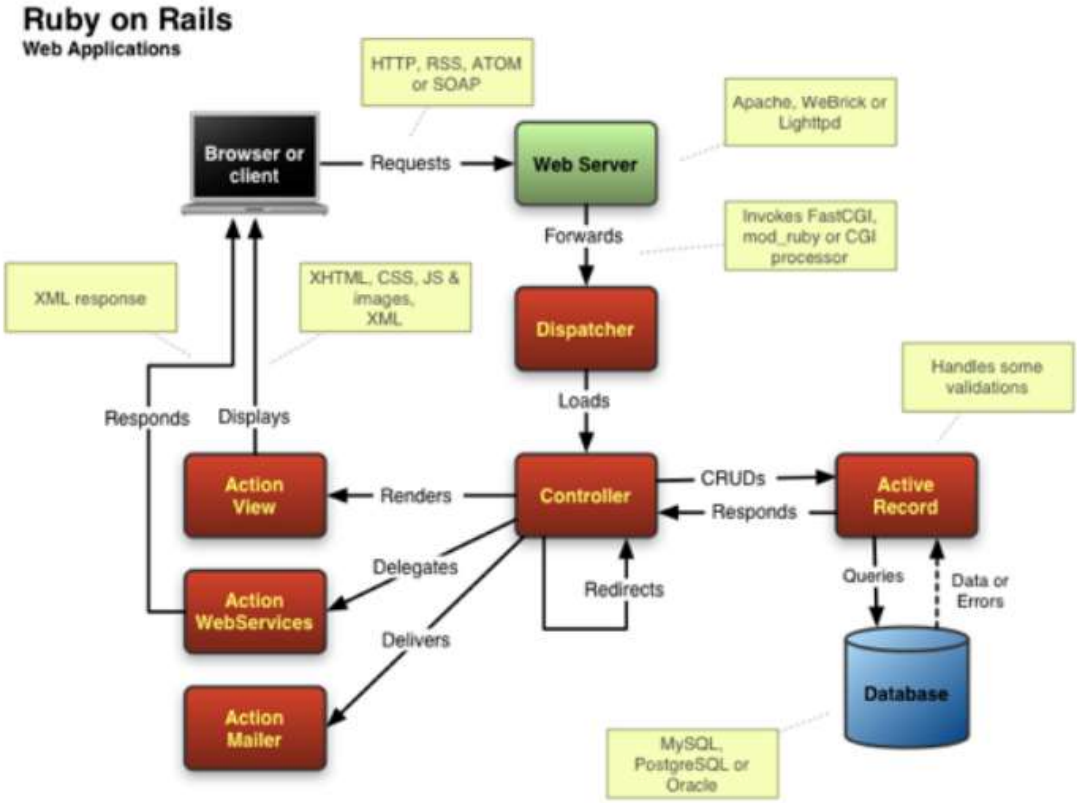
- References are assignment by alias. This means functions take references to the object in the arguments.
- == implicitly converts JS variables so the types match, so 3 == "3" turns the string into a number. === will not do this, and compare two different object types.
- Variables are dynamically typed, so their type is only known at run time.
- Functions are objects too, and are declared like function name(a, b){}, but can also be anonymous, meaning the name part is missing. This is used when we assign them to a variable.
- Due to this feature, we can pass functions in as arguments to another function.
- A closure is a function in JS that takes advantage of the fact that variables from previous function calls will be baked in to future function calls. So if my function returns a function, the returned function will use the data I passed to it when getting the function it returns.
- Arrays are dynamically sized. You can check length with .length.
- Access I index with array[i].
- Instantiate arrays with new, like var n = new Array(3);
- indexOf(thingToFind) and lastIndexOf(thingToFind) look for the first and last index of your element, and return -1 if it doesn't exist.
- .concat concatenates two arrays.
- .slice(n,m) will grab indexes n to m-1.
- .combine(seperator) will combine all entries into a string seperated by the separator argument.
- .push puts things into the end, .pop takes the last value out, .unshift puts things into the start, .shift takes the first value out.
- .splice(index to remove from and insert into, how many to remove forwards, [element list]) will put items into a specific index in an array, and remove the number specified in the second argument.
- .reverse reverses entries. .sort orders the elements. .sort by default sorts by string lex order, so if you pass it a comparator function, it will know what to do.
- Iterative functions are tricky. They usually take a function themselves as arguments. .every will take a function with arguments element, index, and array that performs a logical check on every element. .some does the same, but will return true if 1 or more pass the function you wrote.
- .filter will take a function as above, and return a new array containing every element from the original array that passes your function in the same order.
- .map will create a new array with every element being mutated by your function.
- .reduce will perform the same thing as ruby.
- An object in JS is a partial map of properties, which are just key value pairs. To make an object, just do var name = { property: value, property: value };
- A property is accessed like name.property.
- To add a property, just do name.newProp = true or some other value, to remove a property do delete name.Property;
- Property in name will return true if the object has that property. To iterate over properties, just do for(property in object){ object[property];}
- To assign a method to an object, just use an anonymous function as a property value. You can use this in these functions to refer to the object the method belongs to.
- To write a constructor for an object, remember that any function can be a constructor for it. To do this, just write var nameOfObject = new constructorFunction(arguments); Always capitalize your constructor functions.
- A prototype is an object in JS that will be looked at when a property cannot be found in an object that inherits from it.
- To set a prototype, just do Object.prototype = OtherObject.
- There are three ways to register an event handler on an object:
    - Inline Registration: In the HTML tag, set the onclick attribute to the name of the JS function, like <p onclick="bob()">
    - Direct registration: Use node selection in JS to find the required HTML element in the DOM, and assign it to a variable. Then, set the onclick property of this variable to a function, like:

var e = document.getelementbyid("coolnavbar"); e.onclick = function(){};

- Chained registration: Find the element like in direct registration, and then use the functon .addEventListener("event",function,false); EX: e.addEventListener("hover",explodeGovernment,false);

MVC

- In MVC, the **model** is the data or state of the application. This part of the application ONLY has methods for accessing and modifying the state.
- The **view** renders the contents of the model for the user. If the model changes, the view has to be updated.
- The **controller** translates user actions into operations on the model, like clicks or menu selections.
- In a web application, the model is usually a database and classes that wrap database operations. The view is HTML, CSS and JS files rendered by the clients browser. Files are used by the server to generate these HTML files. The controller will get HTTP requests via a web server, and orchestrates activity on both the view and model.



- In Rails, we use convention over configuration. We use naming and location conventions to wire components together implicitly. We can explicitly route as well, using names and pattern matching. For example, a GET request for /say/hello gets routed to the SayController with method hello, and the SayController will render a response called /say/hello.html/erb. Another example is that the Model classes directly wire up to the database. The model class Order maps to the database table orders. The attributes of order map to columns of a table, as they can be changed and updated to better or worse define the model object, and instances of the Order class map to rows, as they are individual entities.

Rails Models

- When working with Rails models, you will usually look within the app/models folder for the model classes, the /config folder for database configuration, and the /db folder to view your database directly.
- Databases come in many flavors. SQLITE in Rails takes no setup, but MySQL and Postgres work

better.

- A database is a collection of tables, with table names being plural. Each has a list of columns with a name and a type. It also has a list of rows. Invariants on the table exist, like no null values or uniqueness constraints. The unique identifier for each row is called a primary key. A foreign key is how you associate tables.
- A schema is used to define a database. The way to do this with rails is to write create table/index statements in the schema.rb file:

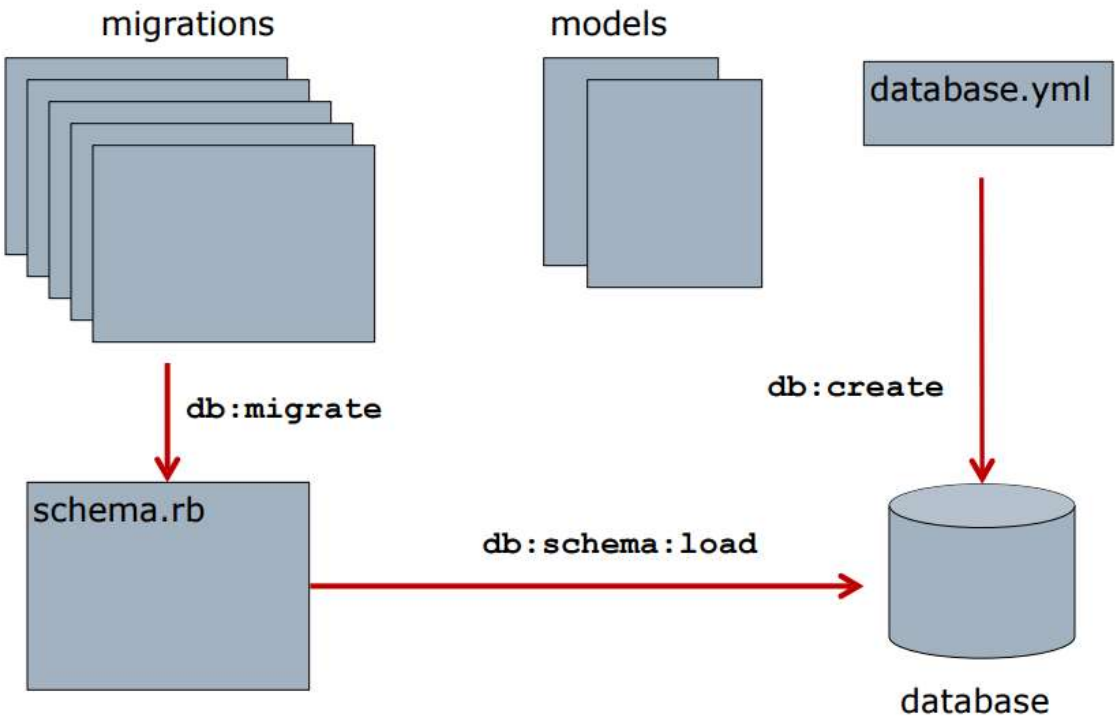
```
ActiveRecord::Schema.define(:version =>
                            20121025193013) d

  create_table "students", :force => true
                           do |t|
    t.string   "name"
    t.integer  "buckid"
    t.datetime "created_at", :null => false
    t.datetime "updated_at", :null => false
  end

end
```

- Instead of writing directly into schema.rb, you should write individual migrations instead. A migration is a class that represents a change in the schema. It can create new tables, modify existing tables, and drop tables.
- These are located in db/migrate. The filename has a timestamp of creation, and a class name that describes what it does. These run in order from oldest to newest. To load the changes into schema, do rake db:migrate.
- A change defined by a migration can be undone, and the schema is the result of applying them in order. We can use:
    - Rake db:create to make an empty db
    - Rake db:migrate to update the schema
    - Rake db:rollback to rollback schema.rb to an earlier point
    - Rake db:schema:load to load the schema

-



- Make sure to never edit schema.rb, and to commit both schema.rb and migrations to version control. Also, make new migrations to undo old migrations.
- Models are a programmatic way for apps to talk to the DB. They extend ActiveRecord::Base and include attributes that correspond to columns. Also, model naming is singular, as an instance of a model is a single row from the DB.
- To create a new instance, use Name.new, to create a new instance and save it to the DB, use Name.create(attribute: value). You can also retrieve a particular row from the table using name.find with the id number, or name.find_by_attribute. You can grab all rows with Name.all.
- Name.save saves the value to the DB, and you can edit attributes like normal in ruby. Name.destroy drops a row.

Rails Validations and Validation

- To keep your model and DB in sync, use the generate statement to make both automatically. This is done like rails generate model Student fname:string age:integer.
- You can also generate migrations using rails generate migration Add/Destroy(Index/Attribute) ToStudent. The name is important.
- To quickly load an attribute, use the add_index command in a migration, passing in the table name and attribute you want.
- To represent a N:1 relationship on a model class, use belongs_to :tableItBelongsTo. To set up a 1:N relationship on a model class, use has_many. This will create methods on the model to access that related info. The method will be the singular or plural of the foreign table.
- 1:1 uses has_one, and N:M uses has_many :through.
- Validations are invariants on the data in a table. The validations are done in the model. If a validation is failed, it will not allow the model to save.

```
class Post < ActiveRecord::Base
  attr_accessible :content, :name, :title

  validates :name,  :presence => true
  validates :title, :presence => true,
                    :length =>
            { :minimum => 5, :maximum => 50 }
end
```

- Uniqueness
  `:uniqueness => true`
  `:uniqueness => {:message => 'Username taken'}`
- Non-nullness (not the same as being true!)
  `:presence => {:message => 'Title needed'}`
- Truth of a boolean field
  `:acceptance => {:message => 'Accept the terms'}`
- Matching a regular expression
  `:format => {:with => /.*/, :message =>…}`
  `:format => /[A-Za-z0-9]+/`
- Being a number
  `:numericality => {:only_integer => true}`
- Having a length
  `:length => {:minimum => 5}`

Routes

- We need to be able to map an HTTP request to an application action. Rails will invoke a method based on an HTTP request and use the parameters from the request. It will then produce an HTTP response. These mappings are called **routes**.
- REST stands for Representational State Transfer, and is a style for developing webapps and mapping request types to HTTP. Basically, you have a bunch of resources that are bundles of server-side state that are identified by URLS. We use GET, POST, PUT, and DELETE to access these resources, and CRUD to change the state.
- A resource can be a collection of items or an individual member, each with their own kind of URL.
- A RESTful delete is done by getting a member either by itself or from a list, and issuing a delete for that member.
- A RESTful create is done by getting a form for a specific collection of items, and POSTING the info in that form.
- A RESTful update is done by GETting a populated form, editing the fields of the form, and PUTTING that info.
- A RESTful read is done by GETting information.

- For a resource like :students, the action pack includes
  - 1 controller (StudentController)
  - 7 routes (each with a method in controller)
  - 4 Views (list of students, show 1 student, new, edit)

HTTP Verb	URL	Resource	Method	Response (View)
GET	/students	Collection	index	list all
POST	/students	Collection	create	show one
GET	/students/new	Collection	new	blank form
GET	/students/3	Member	show	show one
GET	/students/3/edit	Member	edit	filled form
PUT	/students/3	Member	update	show one
DELETE	/students/3	Member	destroy	list all

- To allow routes to be created for a resource, you have to use resource :tableName in the config/routes.rb file.
  - To change which 7 routes are created

```
resources :students, :except =>
                        [:update, :destroy]
resources :grades, :only =>[:index, :show]
```

  - To specify a particular controller

```
resources :students, :controller => 'ugrads'
```

  - To rename certain actions

```
resources :students, :path_names =>
                     { :create => 'enroll' }
```

  - To add more routes to standard set
    - Add /students/:id/avatar (ie on member)
    - Add /students/search (ie on collection)

```
resources :students do
  get 'avatar', :on => :member
  get 'search', :on => :collection
end
```

- A URL request has parameters for a controller class, which can be accessed in the controller using the params[] hash.
- When thinking about routing, you have to think about Recognition vs Generation. We have to be able to recognize a URL, and generate a URL in response. Routes can do both, with helper methods giving relative paths like new_member_path or member_path(:id).
- The root route is the first page your application goes to.

Views and Controllers

- A controller is an ordinary Ruby class that extends ApplicationController. Actions are methods in

the controller class. A view is an HTML page that corresponds to that action. It has access to instance variables of the corresponding controller.

- You can generate a lot of stuff using rails g controller ControllerName action_name, like the controller_name_controller.rb controller class, the action_name method in that class, and the creation of a view corresponding to that action. It will also modify the routes file.
- In ERB:
  - <% code%> executes ruby code
  - <%= code %> returns the result of that code
- App/views/layouts/application.html.erb is for the overall structure of the HTML page, meaning that the HTML in this file will be shown in every page of your application.
- A controller action can create an HTTP response in three different ways:
  - The default response, meaning we try to render (for an example class called example with an action called blah) app/views/example/blah.html.erb. This is done because we did not try to render anything in the method blah.
  - The render response, which means the method called the render method on another action whose view should be rendered. The render method can return many things directly, like json or xml.
  - The redirect response, which sends the response of an HTTP redirect. This makes the client browser request to the indicated URL in the redirect method.
  -
```
class BooksController <
ApplicationController

def create
  @book = Book.new(book_params)
  if @book.save
    redirect_to :action => :show
  else
    render :new
  end
end
```

- A partial is a blob of Erb used in multiple views.
  - Include in a template (or layout) with:
    ```
    <%= render :partial => 'menu' %>
    <%= render 'users/icon' %>
    ```
  - Filename of partial has "_" prefix
    - Default location: app/views
      app/views/_menu.html.erb
    - Organize into subdirectories with good names
      app/views/users/_icon.html.erb

UNICODE

- Glyphs are the things our eyes interact with and our brains recognize.
- Characters can have many glyphs, like different fonts. One glyph can also represent multiple characters. One character could also be made up of different glyphs.
- Codepoints are the integers Unicode assigns to characters.
- Binary Encoding are the bytes that the codepoint compile down to.
- It's best to think of binary as the base tier, then the codepoint that describes a character that can be represented by different glyphs.

- A visual homograph is a security issue, a character that looks like another character. Something like paypal.com could have a different a and go to a malicious site.
- In Unicode, each character is assigned a unique codepoint. A codepoint is defined by an int value, and is also given a name:
    - 109 = LATIN SMALL LETTER M. 109 = 0x006D.
- The convention is to write codepoint 109 as U+006D.
- To break Unicode down into its binary encoding, take every character in the code point and get its binary value.
- Code points are grouped into categories, like basic Latin, Cyrillic, Arabic, currency, math and others.
- The Unicode standard allows for 17 x 2^16 codepoints. Each group of 2^16 is called a plane, with the same two hex characters meaning the same code plane.
    - U+xx1234 and U+xx1235 are in the same plane.
- Plane 0 is the basic multilingual plane and has everything you will probably ever use.
- UTF-8 is a way to write a Unicode character. It encodes a code point (integer) in a sequence of bytes (octets). Just because the 8 is in the name, does not mean that UTF 8 encodes Unicode in a single byte. It can be of variable length, with some having more than one octet. We cannot infer the number of characters from file size because of this, as characters can have different byte sizes. Basically, think of UTF-8 as the way to represent a codepoint as a binary integer.
- The encoding recipe for UTF is as follows:
    - If the Unicode character can be contained in one byte (U+0080), then the UTF-8 encoding is only one byte. The first bit will be 0. So the example earlier would be 00001000.
    - If the Unicode character is x bytes, we will take those x bytes, and consider the overhead from UTF-8 general encoding. The amount of bytes in the UTF-8 encoding will be equal to k, and we put k 1's at the start of the UFT-8 encoding, and after this a 0. After this, the start of each byte in the encoding starts with 10. So U+20AC (0010 0000 1010 1100) becomes 111x 10xxxxxx 10xxxxxx. These x's are then filled in with the binary representation of the Unicode character: 11100010 10000010 10101100.
    - To transform back, take away the first k 1's, and the zero, and then assemble half bytes out of the remaining bits that are not the 10's at the start of the encoding: 0010 0000 1010 1100 - 20AC.
- F0 A4 AD A2 is UTF-8, and the translation to Unicode follows these steps:
    - Binary - 11110000 10100100 10101101 10100010
    - Remove encoding bits: (11110)000 (10)100100 (10)101101 (10)100010
    - Reformat into half bytes (from right to left): 0 0010 0100 1011 0110 0010
    - Translate those bytes to hex to get the characters of the Unicode: U+24B62

- UTF-8 has 1-byte encodings, which encodes all of ASCII, an alternate to UNICODE. It must start with a 0. It has a range from 0 – 127, or U+0000 – U+007F.
- It also has 2 byte encodings, with the first byte starting with 110...
    - The 110 is there to say that the byte is encoded with 2 bytes.
    - The start of every byte has 10 except the first.
- The general encoding is that for an encoding of length k, the first byte starts with k 1s, then a 0 goes after those. The subsequent k-1 bytes each start with 10, and the remaining bits are the payload. This allows for mistakes in parsing to only lose one character.
- Not all encodings are allowed, with overlong encodings being illegal.
- Mojibake is the Japanese word for the garbage you see when you open a file and don't use the

right encoding.
- Unicode started as 2^16 codepoints, with the bottom 256 codepoints matched ISO-8859-1, and encoded every codepoint in 2 bytes. This was simple, but led to the bloat of ASCII characters, due to a bunch of 0s before every ASCII character in Unicode.
- A multibyte representation must distinguish between big and little endian, which is specified in the name, or require the byte order mark (BOM) at the start of the file. There is no U+FEFF codepoint, so we use these characters at the start to indicate byte order.

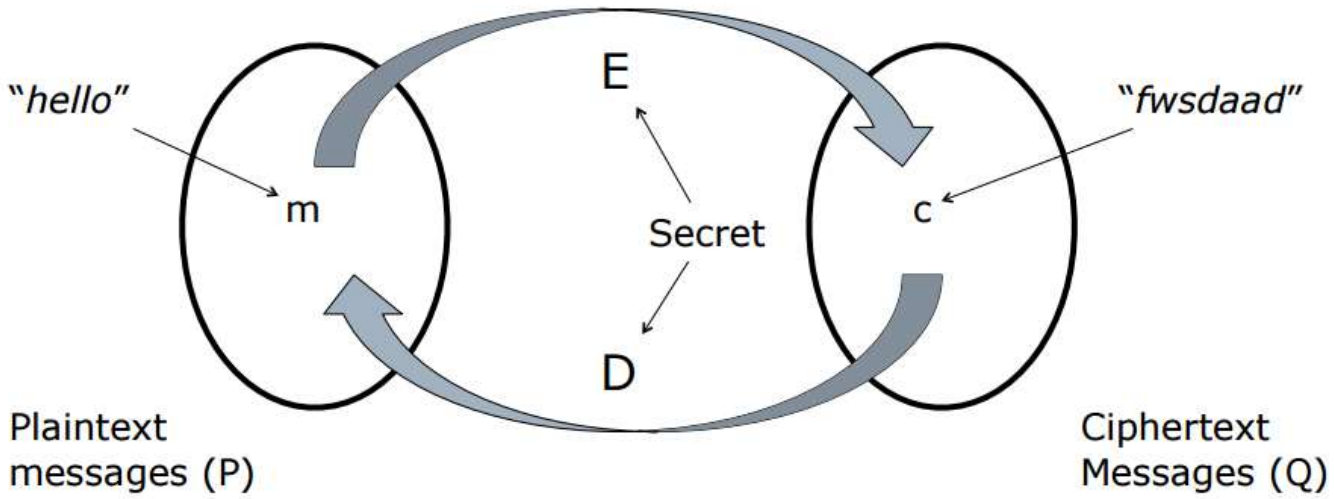
Technical Writing

- **Tech Writing** is writing we do as part of our jobs. It is used to inform, instruct, persuade and others.
- Effective tech writing engages a specific audience, uses plain an objective language, stresses presentation, and employs visual aids.
- We need to be able to communicate good ideas to keep society going, so this is why tech writing is important.
- We need to analyze our audience when doing technical writing, as few people read tech docs for fun. We have to ID a customer, and what their needs are. We need to state assumptions on the customer, like what skills they have. We can determine these through profiling them.
- A tech audience expects ALL info needed on a subject to be in the definitive piece of writing on it. They want function oriented organization, and will read it for a long period of time.
- A customer audience prefers task oriented organization. They just want the needed info and will spend as little time as possible.
- We need to initially identify the purpose of the document, as the golden rule of business comm is to begin with a clear statement of what you want. We need to figure out what info we want to teach.
- For our prewriting, we need to analyze the problem statement and decide upon a list of the cognitive tasks we will cover when writing about the problem. We take in what we already know, and compare that to what we will need to know to cover the cognitive task.
- For each cognitive task, we should make a quick list of the specific points for each task. Brainstorm here, quantity over quality.
- After this, choose a single point that will be in the final product and outline a section to develop that one point. Use other points if they relate, but focus on a specific point. Do research as you finish your outline.
- When outlining, get the planned structure down, don't forget key points, and check for the logical flow of your arguments and info.
- We call **components** as such because they are a section of a document with its own heading. Large components usually have smaller ones, and each element from the outline typically becomes a component.
- Components reduce the document to obvious milestones in a step by step process.
- To write one, you have to know how it relates to the purpose of the doc, have all the needed info, and have different strategies for writing it.
- A good set of rhetorical patterns to use in writing are:
    - General to Specific pattern, which starts with the most general statement and gradually gets more specific.
    - Classification Pattern, which organizes info by dividing it into categories, and presenting parallel info.
    - The compare/contrast pattern, which considers one aspect at a time and finds similarities and differences in the aspect with another aspect.
    - The definition pattern, which is short, simple, and explains in one shot what something is or does.
    - The chronological pattern, which is typical for task oriented instructions. It gives the order in which to do something.
    - The cause and effect pattern

- We put components together by adding headings and transitions between the components.
- The preliminary draft doesn't worry about spelling, grammar, or form, but just filling in the outline and spending effort on sound communication of major points.
- The middle draft builds on the prelim, refines the organization, ensures each point belongs where it does, and allows you to play with the text.
- The final draft is where you fix spelling, grammar, transitions, and word choice.

InfoSec

- Security has a lot of high level goals. Confidentiality prevents non authorized users from access to a system or data. Integrity validates the correctness of data. Availability ensures minimal down time or disruptions. Authenticity verifies that agents of your system are who they claim to be. Non repudiation ensures that a party to a transaction cannot deny that transaction.
- There are a couple of methods of attack:
  - Target people using social engineering. This is stuff like phishing and baiting.
  - Target software, like unpatched OSes, browsers and programs. Buffer overflow and other things like XSS.
  - Target the channel using a man in the middle attack.
- At the heart of security is cryptography. The basic problem is that two agents want to exchange private messages, but their basic channel between the two is not secure. The solution to this problem has other applications too, we can protect stored data, add signatures to deny non repudiation, among other things.
- The core idea is the secret, something that 2 agents share that cannot be the message itself, and is used to protect arbitrary messages. A crude analogy is a padlock, the copies of the key are the secret, and agents can lock and unlock the box. Real channels are bit streams, meaning we have to garble the message in some way.
- To protect a message, one agent encrypts the message, sends the encrypted text, and the other agent can decrypt the cipher text back into plain text.
- Two functions can be used in encryption, the encryption and decryption function. These functions have to be bijections, otherwise we may not get the same message.
- Below is a FSM that represents basic encryption:



$E: P \rightarrow Q$
$D: Q \rightarrow P$

$E(m) = c$
$D(c) = m$
ie $D = E^{-1}$

- Each pair of agents needs an E, an encrypting function, that is unique to them. Since E's are hard to invent, we parameterize one good E with a number.
- A Caesar cipher shifts a letter by x positions. In cryptography, a function is denoted as E, and the key passed into it is used as a subscript, like E3. The general is Ex or Ei, and the Caesar cipher shifts it by the subscript, and creates a new word.
- We can use frequency analysis to break these ciphers. We can use our knowledge of the language to analyze the encrypted text, and break it using these thoughts.
- The polyalphabetic cipher uses different Eis within the same message. This means that the two agents need to agree on the sequence of E's to use. This is perfectly secure, and called a one-time pad.
- The one time pad takes a message that is some sequence of bits, and the one time pad is a random bit sequence. E is a XOR operation on the bits. The cipher text is M0+x0 m1+x1 and on and on. The problems are is the pad is long and cumbersome, and the solution is a pseudo random sequence, generated from a seed (the key).
- Stream and block ciphers have differences. A stream encrypts bit by bit, but a block encrypts a fixed length sequence of bits.
- AES is the Advanced Encryption Standard, is a block cipher with 128 bits in 4x4 bytes. The key size is 128,192, or 256 bits, and AES is a multistep algorithm with many rounds.
- Symmetric key means knowing E is enough to figure out D (its inverse). If you can encrypt, you can decrypt too. The Caesar cipher, one time pad, and AES are symmetric key algorithms.
- For some functions, the inverse is hard to calculate. One direction is easy, but the opposite direction is hard/expensive/slow.
- Multiplying numbers is easy, but factoring a number is hard. We need 2^n steps to factor an n bit number.
- A hash function maps values to Zb=. Every message, regardless of its length, maps to a number in the range 0..B. Good hashes have uniform distribution, with a small difference in the message making a big difference in the individual hashed message in the list of hashed messages called the digest. Crypto hash functions are one way, meaning given a digest is it computationally infeasible to find a message that hashes to it. Collisions must still exist but are infeasible to find.
- MD5 is a hash function that sucks the big.
- SHA-1 is common but deprecated, Chrome, Windows and Firefox reject SHA-1.
- These algorithms can verify file integrity, and also be used for password protection. The server stores a hash of the users password, and check an entered passwords hash against the stored hash, if it matches, we allow them in. This way we do not store passwords that hackers can use if they download the database.
- You should always use a salt, which is used from user to user to add data onto the end of the string of the password.
- Public Key Encryption has asymmetric keys. The key for Bob's E is public, but the key for Bob's D is private. Anyone can encrypt message for Bob, but only Bob can decrypt these messages. Each agent only needs one public key, and no existing pre shared secret is needed. Bob can encrypt things using his private key, and only his public key can be used to decrypt it, letting the decrypter KNOW it was done by Bob.
- RSA is an example of a public key encryption algorithm. E and D are the same function, parameterized by a pair, which is the key.
- Symmetric keys are faster than public key algorithms. The optimization for RSA is to create a fresh symmetric key k, use a symmetric algorithm to encrypt m, and use the recipients public key to encrypt k.