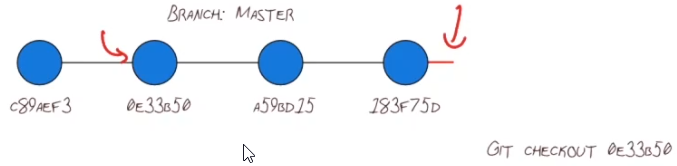
# Cisco DevNet Associate (200-901 DEVASC) – CBT Nuggets Training

Git - Version control

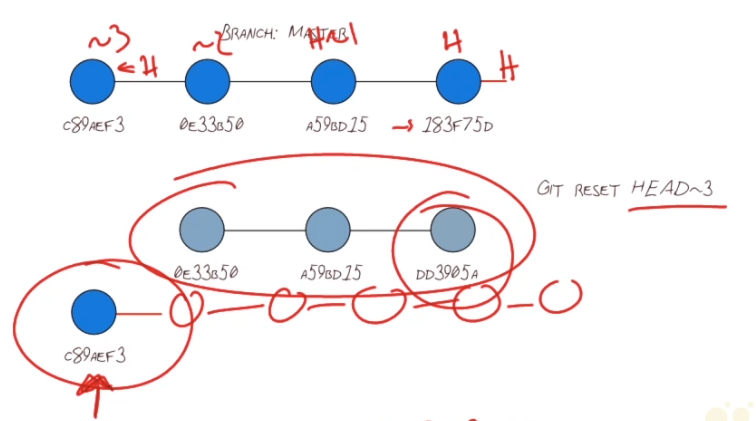
* There are 3-steps in the lifecycle of a file

1. Unmodified
   1. When the file is modified you are making a CHANGE
2. Modified
   1. After the file is modified you should move it to STAGING, with the command *git add .*
3. Staged
   1. Once the file is staged, this version is ready to be committed to the repository with *git commit -m “meaninful message”*

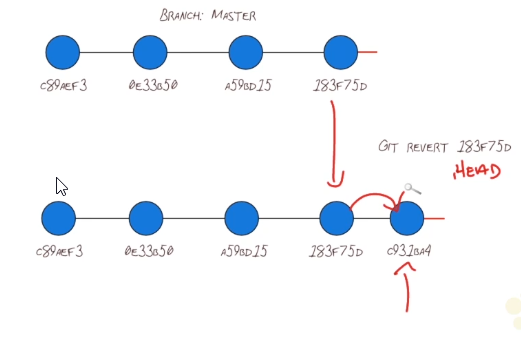
* GIT CHECK OUT
  + We can revert back to any previous commit in a brench.
  + With *git log*, you can see all your commits. And with *git checkout hash*.



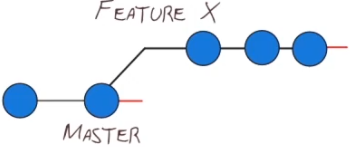
* + *Git checkout* is a tool to observe the code AS IT WAS back in time. And maybe even play with it a little.
  + It can also be used to check on some branch.
  + It should not be used to revert back and undo changes.
* GIT RESET
  + If you reset a branch, use *git reset head <head~3 / head hash>*



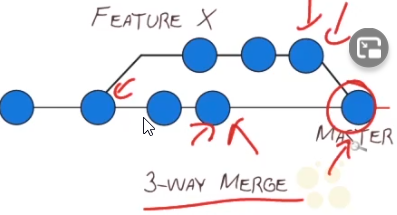
* + With git reset, you point the HEAD to a previous commit, deleting further commits from branch.
  + ATTENTION, because git reset moves the header to a previous commit, but it does not change your local code repository. You will still have all recent files (therefore, a bunch of these files will be unstagged relative to the HEADs new position).
* GIT REVERT
  + Git revert is used when you want to change back to previous commits, but without losing history.



* + Instead, the revert created another commit in the branch, with the reverted code.
* GIT RM and .gitignore
  + If you have committed files you don’t want, you can remove then from the repository with:
    - *git rm <file> --cached*
  + you can also create the *.gitignore* file and list, inside it, the files you don’t want to be staged and committed.
* GIT BRANCHES
  + Branches are parallel working environments.
  + The HEAD can only point to the tail of one branch, being that the master or any other branch.
  + You can create a branch with *git branch <featureA>.*
  + That will not point the HEAD to the branch. If you want, you can do it by *git checkout featureA*
* GIT MERGE
  + After work is done on branches, merges will consolidate the changes back to a single timeline (branch)
  + Target Branch: The branch where changes are being pulled FROM. Usually, branch *feature*.
  + Receiving Branch: The branch where changes will be pulled INTO. Usually, branch *master.*
  + There are two types of merges
    - Fast-Forward is when no changes where made to the master. So *featureX* can merge seamlessly



* + - 3-way merge is when changes were made both in the receiving and target branches.



* + To merge, just checkout the receiving branch and use *git merge feature*
  + If changes are made to different files, merge will be automatic
  + If changes are made to the same files but different lines, git will auto-merge using ‘recusive strategy’
  + If changes are made to the same line of files, conflicts will have to be solved manually.

XML, JSON and YAML

* Plaint-Text data formats are better for humans and computers to read
  + Structured for computers
  + Annotated for humans
  + Open and extensible
  + Self-Destructing
  + Platform Agnostic / good lifespan
* XML – Extensible Markup Language
  + It’s a legacy data format and it is used on SOAP.
  + Uses markups, like html:

<people> #this is called a root tag. All xml must have

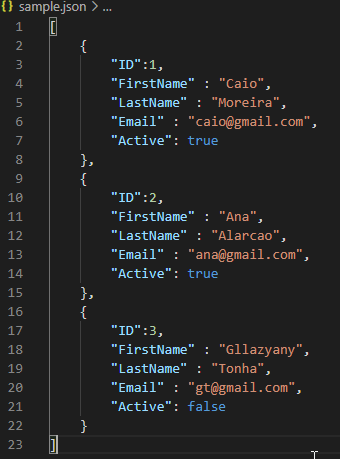
<person id=”1”> #this is called a TAG, and the id is called and attribute

<name>Caio</name>

<age>27</age>

</person>

<people>

* + The structure of the ‘person’ is called and ELEMENT, with elements inside it.
    - Name and age are also elements, but with values ony, no elements.
  + A lot of overhead.
  + DOM – Document Object Module. Is a tree view of the xml.
    - It’s important to understand the relationships of the xmlk data.
    - With the parent, the children and the cousins.
* JSON – JavaScript Object Notation
  + Popular, lightweight notation, native to JS.
  + It uses brackets {}, so indentation is irrelevant, and data can be compressed easily.
  + KEY-VALUE pair is the heart and soul of JSON
    - “FirstName”: “Caio”
    - All keys must use “”, but not all values will use it, depending on format.
    - Everything inside {} is a objetct:
    - There are 3 Objects in this JSON
    - Everything inside [] is an array
    - This JSON is 1 big array
    - Inside objects, you have key pairs
    - Inside arrays, you have objects.
    - But objects can also have array of key pairs:

{“Countries:

[

{“Name” : ”USA”,

“Population” : ”330 million”},

{“Name” : “Brasil”,

“Population” : “200 million”}

]

}

* + - In JS, you reference arrays with []: myObj[1] will get all the object ‘Ana’, in the example.
    - In JS, you reference objects as: myObj[0].LastName will get the string “Moreira”.
* YAML – YAML ain’t markup language
  + Highly legible, compact and succinct.
  + Ideal for configuration files.
  + Indentation is importante for YAML
  + YAML also has a key-pair with a key and a value
  + Values are associated with a data type automatically:
    - Strings, Integer, floats and Booleans.
    - Strings may or may not be inside of quotes “”.
    - Strings with quotes are interpreted with breaks, like \n.
    - Strings without quotes are literal (yes,yaml is contra intuitive)
    - Booleans can be written in capital or lower case.

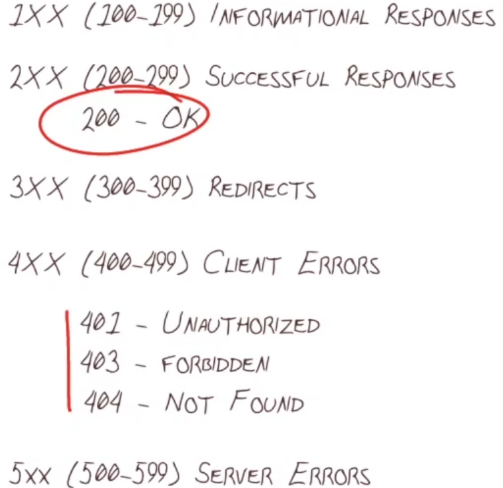
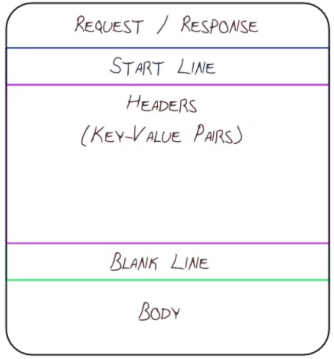
Parsing Data Formats with Python

* Creating data formats with programs is called: Serialization.
* Reading data formats with programs is called: Deserialization.
* For JSON
  + JSON Objects becomes dict (dictionary)
  + JSON Arrays becomes lists

RESTful API

* API – Application Programing Interface. REST – Representational State Transfer
* For the API to be restful, it must follow a set of constrains. 3 most important are:
  + Uniform Interface: The API address for calls must me consistent.
  + Client-server: An API must be consumable by a piece of client software somewhere else.
  + Stateless: Must not have state, that is, remember last calls. Every new call is independent.
* REST vs SOAP
  + Rest is a guideline for the structure of an API. It can use json, xml, basically anything, really.
  + SOAP is a protocol, with specifications at various layers and more comprehensive. Also it uses XML ONLY!

API Requests and Responses

* All HTTP communication generates a request and receives a response.
* HTTP/1.1 communications are plain-text formatted:
* A good API will respond with links on how to navigate to other endpoints. This is called: *Hypertext As The Engine Of Application State*

Authentication with HTTP and REST

* Base64 Encoding vs Encryption
  + Base64 encoding is reversible (by design) and is not ment to be secure
  + Encoding is NOT encryption
    - I have a very good life == SSBoYXZlIGEgdmVyeSBnb29kIGxpZmUu
  + HTTP uses TLS to encrypt data in transit (TLS deprecated SSL) and this is known as HTTPS
* Basic HTTP Auth
  + Plain-text username and password transmitted in the request header
    - Authorization: Basic <user:pass>
  + The key-pair value is base64 encoded
* Api Key Authentication
  + A plain-text token is generated by the API and transmitted by you in the request header
    - Authorization: APIKEY <token>
* Token Authentication
  + Most popular auth
  + JavaScript Web Token (JWTs) is the most popular form of token
  + First you send authentication to server. Then you get a JAWT reply to perform further requests.