FREE COURSE LI NK: https://www.youtube.com/watch?v=ce434nD79nU

Introduction to Automation

What is Automation

- Turning a manual task into an automatic one
- Some tasks just aren't suited for automation, because they might require a degree of creativity or flexibility that automatic systems can't provide
- Configuration management tools provide an out-of-the-box abstraction through which you can manipulate high-level objects, like machines and services
- In IT, the automation of these tasks is commonly accomplished by encoding the logic to perform the task into a **program** or **script**
- What is programming
    - **Syntax -** the rules for how it is written
    - **Semantics** - the meaning of the statements
    - **Scripts** - programs with a short development cycle that can be created and deployed quickly
    - **Compiler** - translates computer code written in one programming language (the source language) into another computer language (the target language)
    - **Interpreter** - directly executes, or performs, instructions written in a programming or scripting language, without previously compiling them into a machine language program
- If (time_to_automate * task_frequency) < (time_to_perform * task_frequency), then automate the task!
- Benefits of automation
    - Automation allows IT infrastructure to **'scale'**, and keep pace with growth and demand
    - **Scalability** - when more work is thrown at it, a system can do that work and increase its output to growth
- Pitfalls of automation
    - Once a task is wrapped in automation, anyone can do it
    - **Pareto principle** - can be used to help determine what to automate
        - 20% of the system administration tasks you perform are responsible for 80% of your work
    - **Bit Rot** - the actual bits in the script don't decay, but its assumptions about the implicit signals it relies on do
        - Happens when automation is not updated alongside infrastructure updates

Scripting with Ruby

"Hello World"

- puts "Hello World"
- **puts** - a special method that Ruby has defined to print things to the screen
- When a programmer uses a **keyword** or **method** in a script, it means that he is making use of the computer language syntax to tell the computer what to do
- Puts will automatically start a new line after displaying a statement on the screen, while **print**, will continue printing output on the same line
- **String** - programming term for a collection of text. A string is what is known in programming terminology as a **Data type**.
- Data Types
    - 

    - 

- **TypeError** - no implicit conversion of string into integer

Expressions

- Expressions & numbers
  - **Expression** - a combination of numbers, symbols, and variables, that, when evaluated, produce a result
  - Comparing things
    - **Boolean** - represents one of two possible states, either TRUE or FALSE
  - Precedence
    - High precedence operators happen before lower ones

Variables

- A placeholder for a value, containers for data
- **Variables** are important in programming because they allow you to perform operations on data that may change
- Naming variables
  - Assignment to a variable is done by using the **=** sign, in the form of variable = value
  - Variable names must start with a letter or underscore, no special characters

Flow control with Branching

- **Nil** - a reserved word to represent the concept “nothing” in Ruby
- Branching with the Else statement

Object Orientation

- Instances of some class representing a concept
- Classes (apple example)
  - class Apple
    attr_accessor :color, :flavor
    end
- Everything is an object
  - Any method in Ruby that returns a boolean value after it is called has a name that ends in a question mark
- Inheritance
  - **<** symbol denotes inheritance relationship
- **Breakdown**:
  - In an OOP language like Ruby, real-world concepts are represented by **classes**
  - Instances of classes are called **objects**
  - Objects are organized by **inheritance**
  - Objects can have **attributes** which are used to store information about them
  - You can make objects do work by calling their methods using **dot notation**

Arrays

- ‘Lists’
- Iteration
  - **each** method
  - string interpolation

File Control with Loops

Ruby Hashes

- **Key**, **value** - in a dictionary the word would be the key, the definition would be the value
- The keys inside a hash must be **unique**

Methods

- Parentheses should be used when there are either parameters to wrap in a **method declaration**, or when there are arguments to wrap in a **method call**
- Instance variables are denoted by @ symbol

Style

- Code should be self-documenting (**Refactoring**)
- Utilize comments for explanation, in Ruby comments are denoted by #
- Have consistency standards / **style guide**
- **Code Review** : after you write some code, you send it to a coworker to proofread it, checking for errors and ensuring it follow the tenets laid out in the style guide

System Programming

Libraries

- Code imported with the **load** method will be reprocessed each time load is used, while **require** keeps track of what it has imported and makes sure it's done only once
- Modules are not classes
- Namespaces
- Mixins
- **Library** - a collection of code, likely organized into multiple modules, that can be required by any program that imports it
- Packages, Ruby uses Gems > look at Gem website

Manipulating Files & Directories

- **Race Conditions** - this will occur when multiple processes try to modify and read one resource at the same time
- **Mode** - kind of like a file permission, which governs what you can do with the file you've just opened
- **Entries** - this method will return an array, each element of which will be a file or subdirectory within the current directory
- **Exists?** Can be used for all kinds of things, like verifying a log file is present before trying to write to it, or checking that a given file hasn't been created yet so you don't overwrite it

Scripting Pipelines

- **I/O Streams** - the basic mechanism for performing input and output operations in Ruby scripts
- Piping **|**

Subprocesses

- Any change to the system or the external commands your scripts utilize is an opportunity for **breakage**, which can sometimes be subtle and hard to detect
- If you want to run a system command an only need to know whether or not it succeeded, use the **system method**
- If you want to quickly run a command and capture its standard output, use **backticks** -- as long as you're ok with the script failing if the command doesn't succeed
- If you need to capture both standard output, standard error, and the return status, import the open3 module and use the **capture3 method**

Writing a Script from the Ground Up

- Problem statement - determine active connections (ping all connections on network)
    - Other options might include viewing the active DHCP leases on the network (like by looking at the **/var/lib/dhcp/dhcpd.leases** file in a Unix-like system, or by running the **netsh** command on Windows, or can use NMAP
- How big is the network?
    - The IPv4 address of our local computer is 192.168.1.10 and its subnet mask is 255.255.255.0

    - In Gems, **IP Addr class** - its purpose is to provide a set of methods to manipulate IP addresses
- Planning
    - First, we know that we'll need to **compute** the range of addresses in the network in order to make sure we can scan them all
    - We also know that we'll need to use the net-pinglibrary to **test** each of these addresses in order to see if they are up
    - Finally, we'll want a way to get this information out of the script. Printing it to the screen would be one way, but since a list of addresses was requested it would be even better if we could **save** them to a file that could be emailed or printed
- Writing the script

```ruby
require "ipaddr"
require "net/ping"

def calculate_network_range(ip, mask)
  address = "#{ip}/#{mask}"
  # transform values with map
  ip_range = IPAddr.new(address).to_range.map { |ip_object| ip_object.to_s }
end

def scan_range(ip_range)
  # filter values with select
  active_ips = ip_range.select { |ip| Net::Ping::External.new(ip).ping? }
end

def write_to_file(active_ips)
  File.open("active_ips.txt", "w") do |file|
    active_ips.each do |ip_address|
      file.puts ip_address
    end
  end
end

range = calculate_network_range("192.168.1.10", "255.255.255.0")
active_ips = scan_range(range)
write_to_file(active_ips)
```

Regex & Test Processing

Working with Strings in Ruby

- **/r** - indicates a carriage return, which means that the cursor should be moved to the beginning of the line
- **Strip** - will remove special characters like \r and \n, horizontal and vertical tabs, and null characters
- indexing, strip, gsub methods

Regular Expressions

- **Regex** or **Regexp** - regular expressions allow you to answer questions like, "What are all of the four-letter words in a file?" or "Give me all of the email addresses in a request log"
- Regex has its own syntax in each language
    - The ^ and $ specifically match the start and end of a line, not a string. If you have a string that spans multiple lines, you can use the \A and \Z anchors to match the start and end of the whole string, not just the line

    - Repetition symbols include both the + character, which means “match one or more occurrences of the preceding character,” and the * symbol, which means “match zero or more occurrences of the preceding character.”
- **Character classes** - allow you to match against the set of characters contained within them
    - Ruby also offers some specialized metacharacters that behave like character classes. These can be used as shortcuts to match specific types of text
- You can also iterate through a string using regular expressions and the scan method, which accepts a regexp as an argument
- You can use **split** to chop up a string into an array, using either a character or regular expression to tell Ruby where to do the splitting

Log File Processing

- Along with saving scripts to a file, Ruby also offers system admins the ability to **process text from the command line** itself, without having to go through the trouble of writing and saving a new script file
- **-e** , this flag lets us execute code we pass directly into Ruby on the command line, without the need to save it to disk first
- **-n** , this flag tells Ruby that we want to apply our code to each line of STDIN until there’s no more

Other Data Formats

- **CSV** files are stores in plain text, and each line in a csv file generally represents a single data record
- **HTML** data format - describes and defines the content of a webpage
    - Nokogiri - popular Ruby Gem used to parse HTML
    - More often than not you’ll want to access HTML-formatted data from an external source rather than a string

Version Control & Testing

Version control review

- **VCS** (version control system) - unlike a regular file server which only saves the most recent version of a file, a **version control system** also stores previous versions of each file as you save your changes
- A VCS provides a mechanism where the author of the last change (or commit) can record why the change was made
- A VCS can function a lot like a time machine, giving you insight into the decisions of the past

Introduction to Git

- Unlike some VCS’s, which are centralized around a single server, Git has a **distributed architecture**
- **GitHub** is a web-based Git repository hosting service. It adds extra features to the version control functionality of Git like bug tracking, wikis, and task management
- You can think of the git directory as the **database** for your Git project, where all of the snapshots you’ve taken from your commits are stored
- When you **clone** a repository, this git directory is copied to your computer
- The working tree is a checked-out snapshot of your project
- The **staging area**, also called an **index**, is a file maintained by Git that has all of the information about what filed and changes are going to go into your next commit

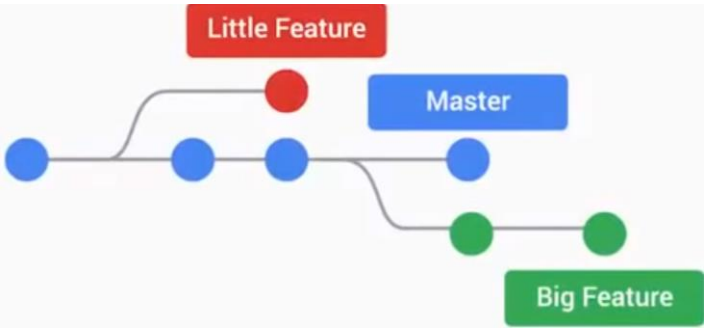
Git under the hood

- Typical workflow: track -> stage -> commit
- Using **amend** rewrites the git history, removing the previous commit and replacing it with the amended one
- Rollbacks
    - In Git, **revert** doesn’t just mean undo. Instead, it creates a commit that contains the inverse of all the changes made in the bad commit, to **cancel them out**.

- Identifying a commit = commit + hash
    - **SHA1** - takes a bunch of data as an input, then produce a 40-character string from that data as an output
- It's important to remember that Git is a tool, not a programming language

Branching & Merging

- **Branch** - a pointer to a particular commit
- **Master** - the default branch that Git creates for you when a new repository is initialized
- Branches make it easy to experiment with new ideas in your projects



- **HEAD** - points to the commit your current working directory is based on, which is usually the same as the branch that you're working on, aka the master
- You can use the git branch command to list, create, delete, or manipulate branches
- Merging
    - **Fast-forward merge** - this kind of merge happens when all of the commits in the checked-out branch are also in the branch that's being merged

Git remotes

- **Remote repo** - a copy of a project usually hosted on a different computer
- A locally hosted Git server can run on almost any platform, including Linux, Mac, or Windows, and has benefits like increased **privacy, control**, and **customization**
- If someone has updated the repo since the last time you copied it to your remote branch, Git will tell you that it's time to do an update. You'll pull down the code, fix and merge conflicts that might arise, and push upstream again
- When you clone a repo using the **git clone** command, Git automatically creates a shortcut to the remote repo called **origin** that points back to the repo you cloned
- Remote branches are **read-only**

Testing

- **Software testing** - the process of evaluating computer code to determine whether or not it does what you expect
- Formal software testing takes this a step further, codifying tests into its own software and code that can be run to verify that your program does what you expect
- **TDD** (test driven development) - the TDD cycle usually first involves writing a test, then running it to make sure it fails
    - Once you've verified that it fails, you write the code that will satisfy the test, and run the test again
    - If it passes, you can continue on to the next part of your program
    - If it fails, you debug it and run the tests again
- Black box vs White box
    - **White box testing** - sometimes called clear box or transparent testing, relies on the test creator's knowledge of the software being tested to construct the test cases
    - **Black box testing** - the tester doesn't know the internals of how the software works

    - White box tests are helpful because the test-writer can use their **knowledge of** the source code itself to create tests that cover most of the ways the program behaves
    - Black box tests are useful because they don't rely on knowledge of how the system works, which means their test cases are **less likely to be biased** by the code
- Unit tests
- Integration tests
- Regression tests
- Smoke tests (hardware testing)
- Taken together, a group of one or more tests is usually called a **test suite**
- Good tests help make any automation and scripts you write more robust, resilient, and less buggy
- When engineers submit their code, it's integrated into the main repository and tests are automatically run against it to spot bugs and errors in a process called **continuous integration**

Configuration & Automation at Scale

Introduction to Automation at Scale

- A scalable system is a flexible one
- IaC = infrastructure as code
- **Configuration Management (CM)** - a system used to configure and manage IT infrastructure, including the bare-metal, virtual, and cloud-based varieties

Configuration Management using Chef

- Recipes & Cookbooks
- In chef, **Node** - some machine managed by a Chef server, which could be anything from a web server to a network router
- **Domain-specific language (DSL)** - a specialized computer language that's used for a specific purpose, like declaring resources within a Chef recipe
- Chef built on Ruby
- **Resources** - statements of configuration that describe the desired state of an item of configuration, like a file, software package, or even a script
- **Cookbooks** - generally represent a set of configuration for a single infrastructure unit or scenario, like it's stated in the official Chef documentation

Monitoring

Alerting

- **Alert** - a machine-generated notification of a problem, meant for a human to read and then act on
- **Prober** - a piece of software that issues logical checks or probes against endpoints exposed by the monitored system

Alerting Example

- Prometheus - Open Source by Google
- Grafana (for visualizations) paired with Prometheus
- Configure alerts to email, pager, number, etc.