Parsing Text in Ansible Lab Guide

[**Goals**](#_1webf6sgacns) **3**

[**Overview of Tasks**](#_m0f5ifwxnaeo) **3**

[**Standards (naming and formatting)**](#_hmerqf6pnh26) **3**

[**Tasks**](#_4tlz2w9mp77y) **4**

[Task 1 - Learn about Regex](#_ps5jeeked896) 4

[RegEx Syntax](#_5ifyj56p1tod) 4

[Task 2 - Matching Strings](#_o70k1emrnjre) 5

[Task 3 - Named Capture Groups](#_vgofp6gl8lsh) 7

[Task 4 - Ansible Templates](#_xo5zifp3f1w6) 8

[Task 5 - Ansible Plays](#_2eidfctkwu5a) 9

[Task 6 - Capturing More than one piece of information](#_aui6b91arktv) 11

[One Solution](#_fb9xc3m0gdox) 12

[Things of note](#_k663wfzb94t6) 13

[Another Option](#_qw642ycquugx) 13

[Task 7 - Recurring Data](#_a9b2h5d5dv7a) 14

[Possible Solution](#_ov4xll5v7z86) 15

[**Additional Information**](#_i1i4g9fjdetv) **17**

# Goals

* Learn about RegEx and matching text
* Learn how to build matching templates
* See how to use the resulting data structure in further tasks

# Overview of Tasks

* Write some Regex to parse the output of some semi-structured text
* Create a template to parse the semi-structured text
* Write the ansible play to extract the data
* Use the data in future tasks

# Standards (naming and formatting)

1. If you need to type a command, the information will be given to you verbatim and will be formatted like this:

**<CODE>**

|  |
| --- |
| This would be the command you need to type |

1. If you need to insert a new name or variable special to your configuration it will be format like the command, but have “{{“ and “}}” around what needs to be replaced and have the “w/ VARIABLE” in the code block as shown in the following example. The variable will be named something descriptive and needs to be replaced. Variables are usually one work with no spaces.

**<CODE w/ VARIABLE>**

|  |
| --- |
| This is a {{ variable }} sudo mkdir {{ your\_project\_name }} |

1. If you need to copy and paste something, you will be shown in a screen shot, but this probably means that it’s individual to your setup and might look slightly different, but generally the same. This happens when copying and pasting keys or links to your code repository. I will not give you the exact command for these as they will not be the same.

# Tasks

## Task 1 - Learn about Regex

Regular Expressions is a language or syntax for matching text. This is very useful when using semi-structured data, such as network cli gear who displays data in a termina, but not in a structured format such as JSON or XML.

Regex is too complicated to teach in a single lab, but in this lab we will cover enough to allow you to get started and you’ll have links to resources to find more information if needed.

### RegEx Syntax

|  |  |  |  |
| --- | --- | --- | --- |
| **Pattern** | **Meaning** | **Example** | **Matching Characters** |
| Any character | Regex will match almost any character literally except for special characters | a | a |
| . | Any single character | cdef | c |
| [aeiou] | Match one of the characters | elf | e |
| [a-zA-Z] | Match a single character in the range of characters | hello | h |
| \s | Any whitespace character | I am fine | I am fine |
| o? | Zero or 1 of the preceding character | hello | o |
| o\* | Zero or more of the preceding character | hoop | oo |
| o+ | One or more of the preceding character | oops | oo |

There are many more combinations that you can read on your own, this gives a brief introduction.

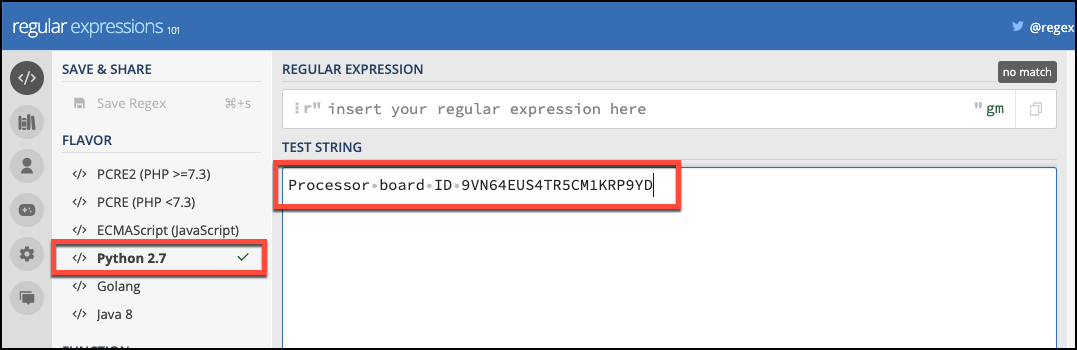
## Task 2 - Matching Strings

Now that we know a bit about RegEx, now let's try to match some text. Many websites are dedicated to teaching and allowing you to test Regex patterns quickly. Google searching for `regex tester` will yield many options. We’ll use regex101.com for this class.

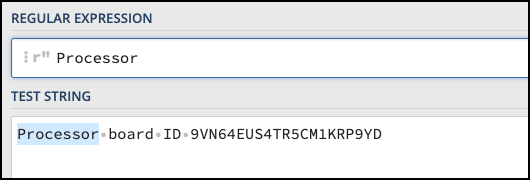
Let's take the following string from the **show version** of a cisco device.

|  |
| --- |
| Processor board ID 9VN64EUS4TR5CM1KRP9YD |

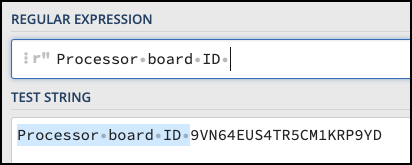
* Open a web browser to the website <https://regex101.com>
* Paste in the string above into the **test string** portion of the page and select **Python** for the flavor.



* In the box that says Regular Expression, type in the word **Processor** and notice how the word Processor is highlighted?

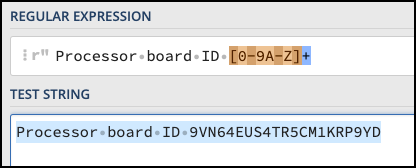


* Keep typing so the test string is matched up to the serial number, what do you think the matching string should be?
* This is what your regular expression should look like now.



* So now we need 2 things, the first is the regex to match the serial number. Up to this point, we’ve been able to literally match each character, now we need wild cards since not every serial number is going to be the same and this is only useful if we can match patterns across all the devices.
* There are several options that would work at this point. The following chart has several possible options and explanations of what they are doing. We’ll select the last one for our example, but feel free to try out the others on the Regex101 website. Are there any other options you can think of that would also match?

|  |  |
| --- | --- |
| Regex Matching String | Explanation |
| \S+ | One or more of any non-whitespace characters. This is good as we want to catch everything from here to the end of the line. |
| .+ | One or more of any character. This likely would be a bit too broad as it would capture anything on the line |
| [0-9A-Z]+ | This is a good match. Only matches characters from A-Z and 0-9. Since our matching string does not have lower case in it, it won’t match some other line that may be similar. |
| [0-9A-Z]{21} | This is an even more exact match as it will only capture the next 21 characters. If there were more after, it wouldn’t capture those. Depending on your needs, this may or may not be useful. |



## Task 3 - Named Capture Groups

Another concept that needs to be introduced is the idea of a Named Capture Group. Once you’ve found the information you want to extract, you can give it a name. In order to use the Ansible native text parsing module, you must use named capture groups.

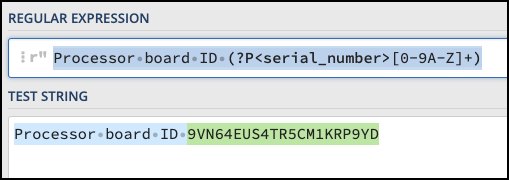
The syntax for a named capturing group is as follows:

|  |
| --- |
| (?P<name>...) |

The three dots **...** are where your Regex Matching pattern will go. The characters between the < and the > are the name of the capturing group.

* Lets rewrite our regex expression above so we have a named capturing group called **serial\_number**.

|  |
| --- |
| Processor board ID (?P<serial\_number>[0-9A-Z]+) |



* You’ll now see that the Regex101 website updates showing the color of the named capture group as a different color.

## Task 4 - Ansible Templates

Now that we have a matching regex, we need to build a template that Ansible can use.

So we could create a template file and point the **cli\_parse** module at that template, but Ansible has a method for auto selecting a template based on the connection type and the command.

* Create a **templates** folder in your working directory
* In that folder create a file called **ios\_show\_version.yaml**

That file name is specific for this task. The **ios** refers to the connection type we’re going to use. The **show\_version** refers to the commands, and the file must end with **.yaml**

In the **yaml** file paste the following data

|  |
| --- |
| - example: Processor board ID 9VN64EUS4TR5CM1KRP9YD  getval: "Processor board ID (?P<serial\_number>[0-9A-Z]+)"  result:  sn: "{{ serial\_number }}" |

* The first line in the **example**, is an example of the string we expect to match.
* The **getval** is the actual Regex we developed earlier in the Regex101 website.
* The result is the resulting data structure that will get returned. The final output must be a dictionary. In our result section, we’re saying when we find a match, we want the **sn** key to have the **serial\_number** variable from the capture group.

## 

## Task 5 - Ansible Plays

The module we’re going to use is the **ansible.netcommon.cli\_parse** module.

If it’s not already installed, run the following command at the command prompt.

|  |
| --- |
| ansible-galaxy collection install ansible.netcommon |

This will use the ansible-galaxy command to install the external module.

Once installed, we’ll need some additional files created in the root directory.

A file named **hosts**

|  |
| --- |
| [all:vars] ansible\_connection=ansible.netcommon.network\_cli  [Cisco] 192.168.255.10 192.168.255.11  [Cisco:vars] ansible\_network\_os=ios ansible\_user=ignw ansible\_password=ignw |

The hosts file has all the variables that need to be set for Ansible to be able to connect to the network devices in the lab.

A file name **ansible.cfg**

|  |
| --- |
| [defaults] host\_key\_checking = False |

You could SSH to each of the devices first, but if you don’t want to do that, you can set this to false and Ansible will connect to the devices without verifying the host keys first.

And finally a file named **parser.yml**

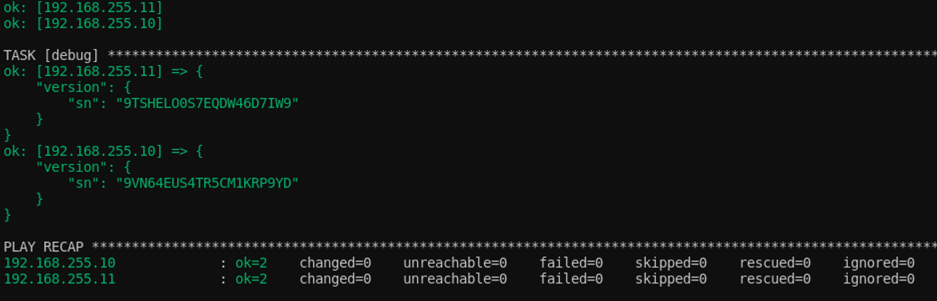
|  |
| --- |
| --- - hosts: Cisco  gather\_facts: false   tasks:  - name: Output device information  ansible.netcommon.cli\_parse:  command: show version  parser:  name: ansible.netcommon.native  set\_fact: version   - debug:  var: version |

This is the module that will connect to the device, run the **show version** command, then using the convention above, parse the output using the **ios\_show\_version.yaml** file in the templates directory.

Run the command with the following command

|  |
| --- |
| ansible-playbook -i hosts parser.yml |

The output looks like this.



## 

## Task 6 - Capturing more than one piece of information

So, what if you want to capture more than one piece of information from the **show version** command.

Most of the information on the **show version** tab can be found by the default **gather\_facts** feature of ansible. Lets say though, we wanted to capture the uptime hours and minutes as two values.

* Start with regex101.com and build a regex to match the uptime string, this time you’ll want two capturing groups, one for minutes, and one for hours.
* Add another set of values to our **ios\_show\_version.yaml** template
* Then update the **result** section with both pieces of data. Recommend using **minutes** and **hours** for the keys

Try to do this on your own using the examples above as guides. If you get stuck, there is an answer that works below.

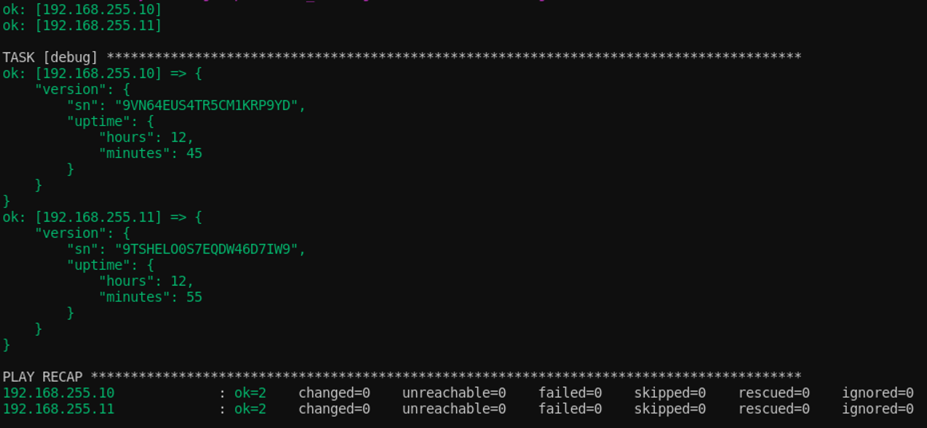
### 

### One Solution

There are several ways you could have done this, but below is an example of a capture template that meets the criteria above.

|  |
| --- |
| - example: Processor board ID 9VN64EUS4TR5CM1KRP9YD  getval: "Processor board ID (?P<serial\_number>[0-9A-Z]+)"  result:  sn: "{{ serial\_number }}"  - example: IOS-1 uptime is 11 hours, 25 minutes  getval: "\\S+ uptime is (?P<hrs>[0-9]{1,3}) hours, (?P<min>[0-9]{1,2}) minutes"  result:  uptime:  hours: "{{ hrs }}"  minutes: "{{ min }}" |

And below is our output for the above solution



### 

### Things of note

* In the **getval** command the double backslash **\\S+** If you need to use the backslash in your regex, when you put it in the yaml, you’ll need to escape it, using double backslash.
* The reason the **\S+** was used is because the hostname of the device is in that row, so every system, that string will be different. We could of also use **.+** to match the characters as well.
* An extra key was added called uptime, this can be done to make the data more usable or readable.
* This will only match the string if we have 1-3 numbers for hours and 1-2 numbers for minutes. What happens after 23 hours? Or 364 days? If the string is different once the counter rolls over, you may need additional captures rows in the template.

### Another Option

Maybe you want to know the total number of minutes, we could modify the above output with a new value called **total\_minutes**

|  |
| --- |
| - example: IOS-1 uptime is 11 hours, 25 minutes  getval: "\\S+ uptime is (?P<hrs>[0-9]{1,3}) hours, (?P<min>[0-9]{1,2}) minutes"  result:  uptime:  hours: "{{ hrs }}"  minutes: "{{ min }}"  total\_minutes: "{{ (hrs|int \* 60) + min|int }}" |

In this example, we use some of the math functions from python and Jinja2 filters to convert the string values to integers and do math with them.

## Task 7 - Recurring Data

The above example worked great for the **show\_version** because it was just a single form. What if you had repeating data? A good example of this would be the **show interfaces** command.

The first couple of output lines of the command are below and we want to capture the type of hardware for each interface.

|  |
| --- |
| GigabitEthernet0/0 is up, line protocol is up   Hardware is iGbE, address is 5254.0011.4522 (bia 5254.0011.4522)  Internet address is 192.168.255.10/24 |

* The first thing we need to make a new template in the **templates** folder
* Then we need to make a regex expression that will match each of the interface names and capture the name of the interface.
* Then we need to match the **Hardware is** line and capture the model
* Then we need to create a data structure to output the data.

Attempt a solution yourself. If you get stuck, you can ask for help. In the next section is a walkthrough of a working solution.

**TIP:** Don’t forget to escape your regex in your YAML files.

You can use the Ansible Cli Parse documentation to help you.

<https://docs.ansible.com/ansible/latest/collections/ansible/netcommon/cli_parse_module.html>

### Possible Solution

* Create a new file called **ios\_show\_interfaces.yaml**
* Now on one of the devices run the **show interfaces** command and copy the output into the regex101.com website under the test string section.
* We start by trying to match on the first line containing the interface name. Since the line can have multiple values, matching is a bit difficult. If there are not too many options, we can use the **OR** value

|  |
| --- |
| (?P<name>GigabitEthernet\S+|Loopback\S) is |

Now we need another regex to match the hardware type. In this case because we just want the next word after the string **Hardware is** we could use the regex [a-zA-Z0-9] but there’s another value called \w that will match any word character.

|  |
| --- |
| Hardware is (?P<type>\w+) |

Now our template looks as follows

|  |
| --- |
| - example: GigabitEthernet0/0 is up, line protocol is up   getval: "(?P<name>GigabitEthernet\\S+|Loopback\\S) is"  result:  "{{ name }}":  name: "{{ name }}"  shared: true  - example: Hardware is iGbE, address is   getval: " Hardware is (?P<type>\\w+)"  result:  "{{ name }}":  hardware\_type: "{{ type }}" |

And our final output is on the following page. Note the **shared** tag. This tag will allow values from that parser to be shared among all the other parsers in the template. It will always output the same value until it finds a match, then it will update to the new value.



# Additional Information

Regex101

<https://regex101.com/>

Parsing semi-structured text in Ansible (Ansible Documentation)

<https://docs.ansible.com/ansible/latest/network/user_guide/cli_parsing.html>

Ansible cli\_parse document (Ansible Documentation)

<https://docs.ansible.com/ansible/latest/collections/ansible/netcommon/cli_parse_module.html>