

How To Use This Deck

This deck relates to code examples found here: <https://github.com/GoKEV/module-creation>

This is intended to explain / show / hand off examples of custom module creation to those who have been unable to succeed, or are unsure where to start.

Stuff To Change:

The next obvious step is for this deck to describe converting this standalone content into a collection (we do not currently cover that yet in this deck)

The collection version of this walkthrough ~~should~~ will include plugins, playbooks, roles:

- Inventory plugin
- Parser plugin
- Playbook to launch two custom modules in sequence
- Role to perform specific functions using custom modules.
- Screenshots of file hierarchy
- Screenshots of collection in automation hub








Understanding Ansible Modules

And Creating Functions Within Them

Kevin Holmes
Principal Solution Architect

Red Hat
ISV Alliances Team

What Will This Course Provide?

-  Make you a better programmer
-  Teach you about python
-  Write better playbooks
-  Give a fundamental understanding of the
interaction of a playbook and a module
-  Inspire you to build a functional module

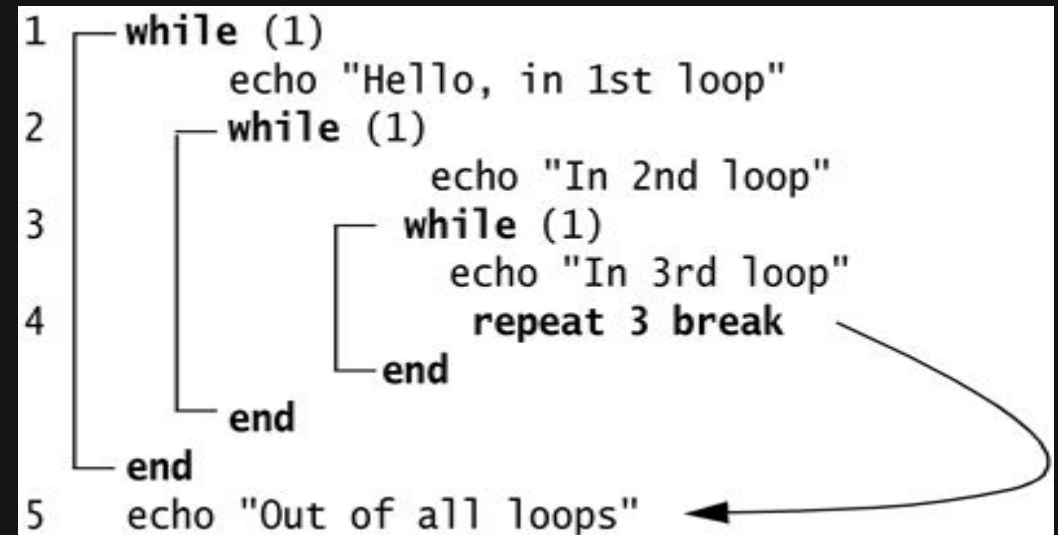
What / Why Is This?

The concept of **How To Create An Ansible Module** has always been a sort of mystery to the average person. Many DWABs ***almost*** make it copy-paste simple. (DWAB = "DUDE WITH A BLOG")

What / Why Is This?

The concept of **How To Create An Ansible Module** has always been a sort of mystery to the average person. Many DWABs *almost* make it copy-paste simple. (DWAB = "DUDE WITH A BLOG")

Ansible Isn't Always The Best Solution As a Standalone or could be made significantly easier with scripting behind the scenes of the YAML

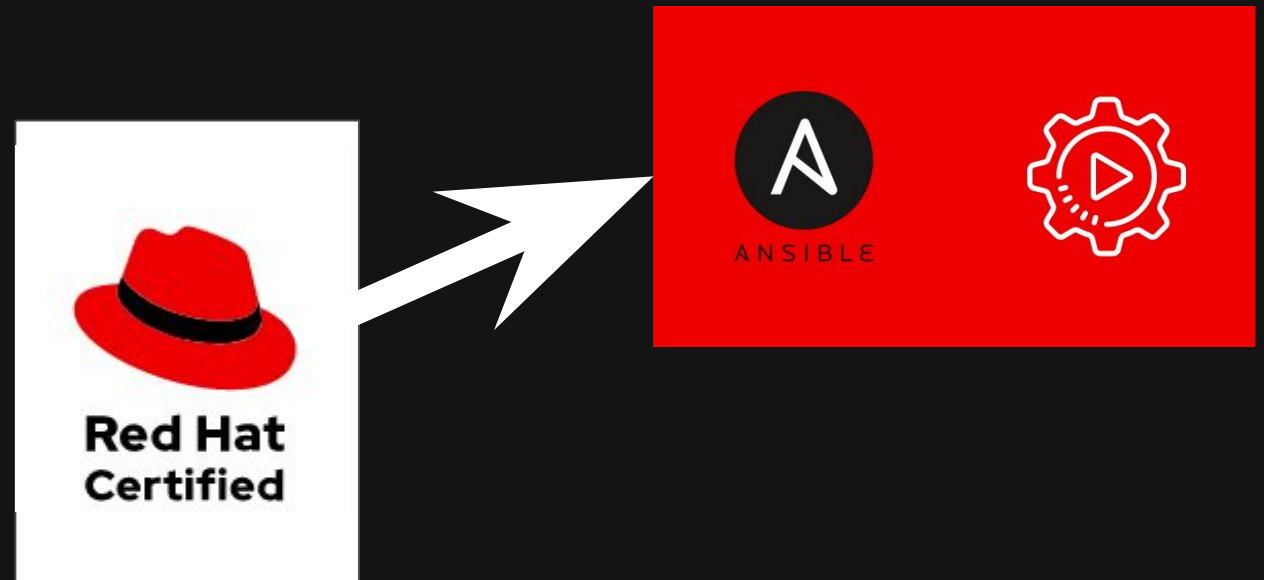


What / Why Is This?

The concept of **How To Create An Ansible Module** has always been a sort of mystery to the average person. Many DWABs ***almost*** make it copy-paste simple. (DWAB = "DUDE WITH A BLOG")

Ansible Isn't Always The Best Solution As a Standalone or could be made significantly easier with scripting behind the scenes of the YAML

Our Embedded and Alliance Partners
want to create their own certified content



What / Why Is This?

The concept of **How To Create An Ansible Module** has always been a sort of mystery to the average person. Many DWABs **almost** make it copy-paste simple. (DWAB = "DUDE WITH A BLOG")

Ansible Isn't Always The Best Solution As a Standalone or could be made significantly easier with scripting behind the scenes of the YAML

Our Embedded and Alliance Partners want to create their own certified content

Python Elitism Is A Thing. When a Python developer thinks about attaching anything non-Python... it's unnatural.

But other languages exist And are used daily for core functions of RHEL. There are some cases where an appliance natively uses a language besides Python.



How Deep Are We Going?

Eventually, We Could Discuss:

- Batteries Included... but "more power?"
- Environments With Proprietary Craft
- Certified Collections for ISV / Embedded Partners



Today, We Are Merely
Scratching The Surface:

The absolute basics of understanding modules.



Quick Ansible Refresher

Repeat after me:

- Simple
- Powerful
- Agentless

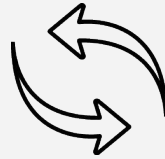
Why Ansible?



Simple

- Human readable automation
- No special coding skills needed
- Tasks executed in order
- Usable by every team

Get productive quickly



Powerful

- App deployment
- Configuration management
- Workflow orchestration
- Network automation

Orchestrate the app lifecycle

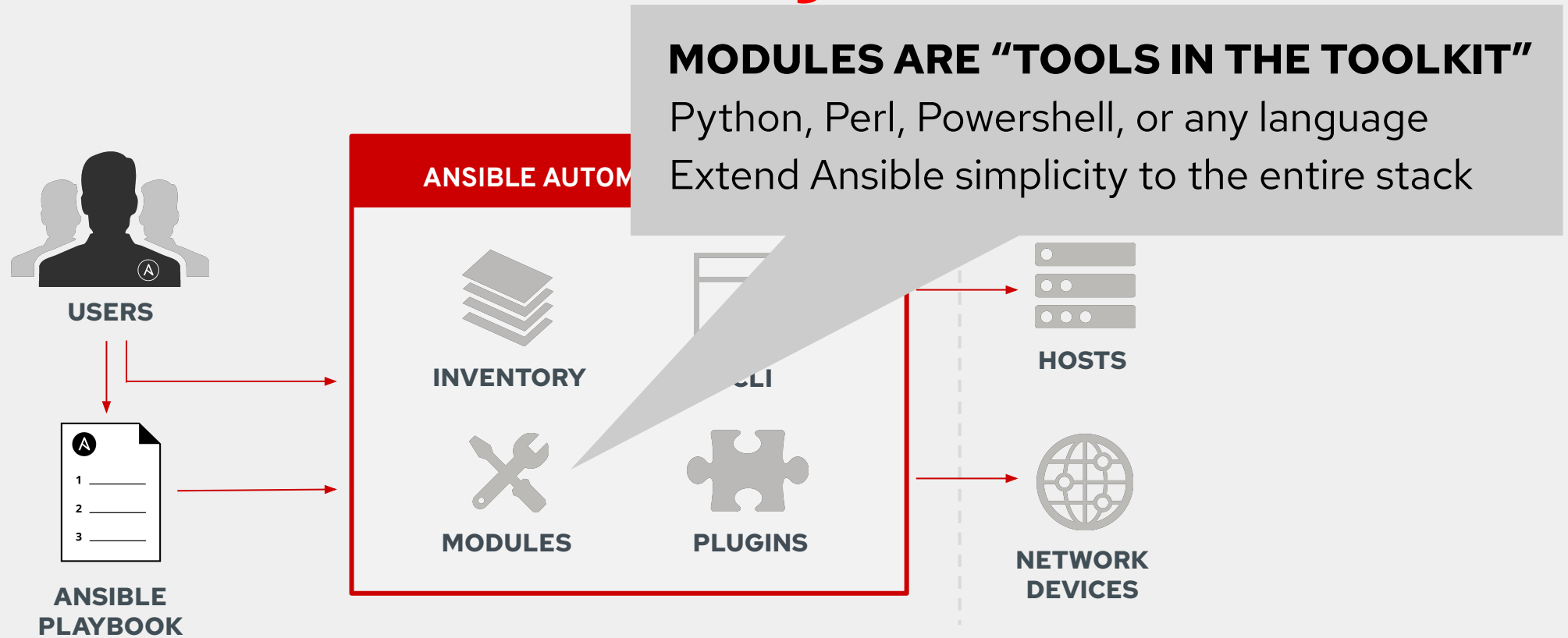


Agentless

- Agentless architecture
- Uses OpenSSH & WinRM
- No agents to exploit or update
- Get started immediately

More efficient & more secure

Ansible modules are typically written in Python, but not always...



Script

```
hosts_script.sh
```

```
#!/bin/bash  
echo $1 $2 >> /etc/hosts  
echo success.
```

```
[root@server01]# ./hosts_script.sh 127.0.0.1 someserver.tld  
success.  
[root@server01]# ./hosts_script.sh 127.0.0.1 someserver.tld  
success.  
[root@server01]# ./hosts_script.sh 127.0.0.1 someserver.tld  
success.
```

```
[root@server01]# tail /etc/hosts
```

```
127.0.0.1 someserver.tld  
127.0.0.1 someserver.tld  
127.0.0.1 someserver.tld  
127.0.0.1 someserver.tld  
127.0.0.1 someserver.tld  
127.0.0.1 someserver.tld
```

Module Wins.

```
- lineinfile:
  dest: /etc/hosts
  line: "{{ host_ip }}" "{{ host_name }}"
  state: present
  loop: "{{ play_hosts }}"
```

```
[root@server01]# ansible-playbook sethost.yml -e
"host_ip=127.0.0.1 host_name=someserver.tld"
```

What Are Ansible Modules?

- How they interact with Ansible playbooks
- Expected input / output

A Simple Ansible Playbook : Structure, Names and Logging

```
- hosts: webservers
  tasks:
    - name: Ensure httpd package is present
      yum:
        name: httpd
        status: installed

    - name: Ensure latest index.html file is present
      copy:
        content: Howdy
        dest: /var/www/html/index.html

    - name: Ensure httpd is running
      service:
        name: httpd
        state: started
```

Logging will show
these notations
Names of the
modules being used
Parameters Passed to the Modules

Running an Ansible Playbook: Logging Output

The many colors of Ansible

A task executed as expected, no change was made.

A task executed as expected, making a change

General text information and headers

A conditional task was skipped

A bug or deprecation warning

A task failed to execute successfully

Running an Ansible Playbook:

```
[user@ansible] $ ansible-playbook apache.yml
```

```
PLAY [webservers] *****
```

```
TASK [Gathering Facts] *****
```

```
ok: [web2]
```

```
ok: [web1]
```

```
ok: [web3]
```

The "Setup" module

```
TASK [Ensure httpd package is present] *****
```

```
ok: [web2]
```

```
ok: [web1]
```

```
ok: [web3]
```

The "yum" module

```
TASK [Ensure latest index.html file is present] *****
```

```
ok: [web2]
```

```
ok: [web1]
```

```
ok: [web3]
```

The "copy" module

```
TASK [Ensure httpd is running] *****
```

```
ok: [web2]
```

```
ok: [web1]
```

```
ok: [web3]
```

The "service" module

```
PLAY RECAP *****
```

```
webservers : ok=3 changed=3 unreachable=0 failed=0
```

But what the heck is that module doing?

How does the module see the variables I send?

What does it do with the data?

How does the module decide what to do on the target host?

Should I be worried about all this flying text?

Planning your Module; What can it do?

- System level configuration / file system changes
- Interrogative probing of services / conditions
- Detecting proprietary application conditions
- The same functions of existing modules, only in a flat and silo-ed way.

Beware Of This

Scripts Are Proprietary Plugs. Modules Are The Adapter Plugs

Each Module is a Connector To A Function

Each Ansible module executes code in an expected manner, returning a consistent behavior and output.

An Ansible module takes simple input variables and gives a consistent and predictable reply.

A module is a piece of code (a script, essentially) that can perform actions as simple or as complex as it's built to perform.



Dissecting The Communication

- What does Ansible say to the module?

Science Class: Dissecting a Living Playbook

Playbook Launch

```
[root@server01]# ansible-playbook some-playbook.yml
```


Science Class: Dissecting a Living Playbook

Playbook Launch

Ansible executes the module (script) with a single argument

```
[root@server01]# ansible-playbook some-playbook.yml
```

```
./path/to/module.py /some-temp-dir/ansible-args-file
```

Science Class: Dissecting a Living Playbook

Playbook Launch

Ansible executes the module (script) with a single argument
The code in our module (script) parses through the variables it is given and determines what it needs to know from Ansible

```
./path/to/module.py /some-temp-dir/ansible-args-file
```

```
#!/some-language  
  
&pull_vars(ansible-args-file);  
  
&evaluate($some_var);
```

Science Class: Dissecting a Living Playbook

Playbook Launch

Ansible executes the module (script) with a single argument
The code in our module (script) parses through the variables it is given and determines what it needs to know from Ansible

Our code decides what actions to take, what logic to consider, and how to reply to Ansible.

```
if (($some_var) == ($some_stuff)) {  
    &do_some_stuff;  
}else{  
    ! &do_some_stuff;  
}
```

```
&some_stuff{ do things ; be cool ; }
```

Science Class: Dissecting a Living Playbook

Playbook Launch

Ansible executes the module (script) with a single argument
The code in our module (script) parses through the variables it is given and determines what it needs to know from Ansible

Our code decides what actions to take, what logic to consider, and how to reply to Ansible.

The module (script) performs actions on the target system

```
! &do_some_stuff;
```

```
&some_stuff{ do things ; be cool ; }
```

Science Class: Dissecting a Living Playbook

Playbook Launch

Ansible executes the module (script) with a single argument
The code in our module (script) parses through the variables it is given and determines what it needs to know from Ansible

Our code decides what actions to take, what logic to consider, and how to reply to Ansible.

The module (script) performs actions on the target system

Reply back to Ansible with status and results.

```
[  
  {some_stuff: "was done", we_did: "be cool"},  
  {changed: "True"},  
  {failed: "False"}  
]
```

Science Class: Dissecting a Living Playbook

Playbook Launch

Ansible executes the module (script) with a single argument
The code in our module (script) parses through the variables it is given and determines what it needs to know from Ansible

Our code decides what actions to take, what logic to consider, and how to reply to Ansible.

The module (script) performs actions on the target system

Reply back to Ansible with status and results.

Close the connection to our module (script)

```
TASK [some stuff that happened]
*****
ok: [server02]
ok: [server01]

TASK [next stuff is about to happen]
*****
```



Science Class: Dissecting a Living Playbook

```
tasks:
  - name: some stuff that happened
    some_module:
      parameter: 'some parameter'
  - name: next stuff is about to happen
    next_module:
      parameter: 'some other parameter'
```

Close the connection to our module (script)

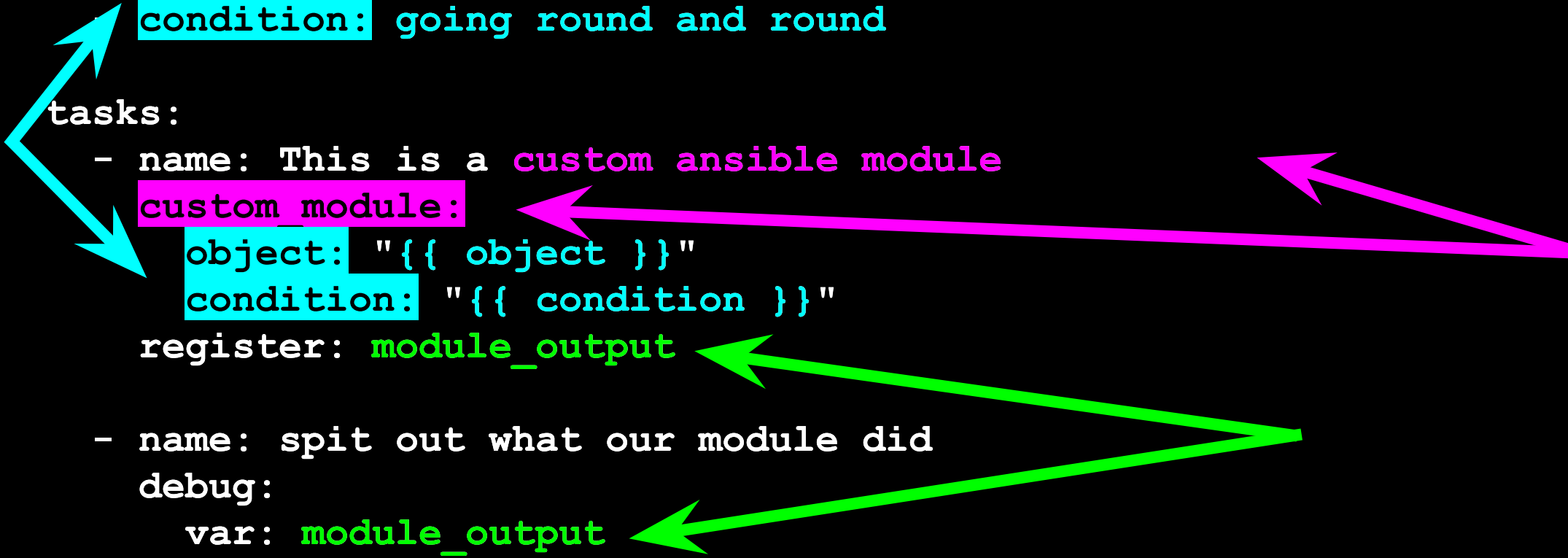
Next task in the playbook

```
TASK [next stuff is about to happen]
*****
```



An Ansible Playbook with a Simple Module

```
- hosts: localhost
gather_facts: no
vars:
  - object: Wheels on the bus
    condition: going round and round
tasks:
  - name: This is a custom ansible module
    custom_module:
      object: "{{ object }}"
      condition: "{{ condition }}"
      register: module_output
  - name: spit out what our module did
    debug:
      var: module_output
```



Running A Playbook with a Custom Module:

```
[module-creation]# ansible-playbook ./custom_module.yml

PLAY [localhost]
*****

TASK [This is a custom ansible module written in interpretive language scripting]
*****
changed: [localhost]

TASK [debug]
*****
ok: [localhost] => {
  "module_output": {
    "changed": "true",
    "failed": false,
    "msg": "The object is 'Wheels on the bus' and the condition is 'going round and
           round', but a vowel in the object marks it as CHANGED",
    "results": [
      "This is a line that goes into results",
      "And so is this",
      "a vowel in the object marks it as CHANGED",
      "no failure was found"
    ]
  }
}

PLAY RECAP
*****
localhost : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Running A Playbook with our Custom Module:

```
#####  
##### CHANGE LOGIC GOES HERE #####  
#####  
if ($object =~ /[aeiouyAEIOUY]/){  
    $is_changed++;  
    $thismsg = "a vowel in the object marks it as CHANGED";  
    $msg .= ", but $thismsg";  
    push(@resultsitems, $thismsg);  
}
```

```
##### FAIL LOGIC GOES HERE #####  
#####  
if ($condition =~ /[jzJZ]/){  
    $is_failed++;  
    $thismsg = "the characters j or z in status mark it as FAILED";  
    $msg .= ", failed due to inappropriate condition letters";  
    push(@resultsitems, $thismsg);  
}  
if ($condition =~ /grumpy|angry|snarky/i){  
    $is_failed++;  
    $thismsg = "grumpy, snarky, angry make this fail";  
    $msg .= ", failed due to a bad condition attitude";  
    push(@resultsitems, $thismsg);  
}  
}  
}  
PLAY RE  
*****  
localho
```

Science Class: Dissecting a Living Playbook

```
[root@module-creation]# ansible-playbook explain.yml

...

TASK [debug]
*****
ok: [localhost] => {
  "modoutput": {
    "changed": false,
    "failed": false,
    "msg": "",
    "results": [
      "Ansible sends the variables in the file as argument 0, which is",
      "/root/.ansible/tmp/ansible-tmp-1611123-14241-24380315/args"
      "We copied the vars file to
      /tmp/ansiblevars.txt
      so we can read it after the process ends",
      "We also formatted it nicely and copied the vars to
      /tmp/ansiblelines.txt"
    ]
  }
}
```

The Input Variables In a File

We Copy Them Here To View

Science Class: The Variable Contents

```
[root@module-creation]# cat /tmp/ansiblevars.txt

object='Pink Floyd' condition='comfortably numb' _ansible_check_mode=False
_ansible_no_log=False _ansible_debug=False _ansible_diff=False
_ansible_verbosity=0 _ansible_version=2.9.14
_ansible_module_name=explain_in_perl _ansible_syslog_facility=LOG_USER
_ansible_selinux_special_fs=['\"fuse\"', '\"nfs\"', '\"vboxsf\"',
 '\"ramfs\"', '\"9p\"', '\"vfat\"']
_ansible_string_conversion_action=warn _ansible_socket=None
_ansible_shell_executable=/bin/sh _ansible_keep_remote_files=False
_ansible_tmpdir=/root/.ansible/tmp/ansible-tmp-1610567227.1178262-8829-1237339
41222418/ _ansible_remote_tmp=~/.ansible/tmp'
```

Oh, the fun of space-delimited variables!
Let's clean this up a bit so that it's easier to read.

Science Class: The Variable Contents (Readable)

```
[root@module-creation]# cat /tmp/ansiblelines.txt
```

```
object='Pink Floyd'
condition='comfortably numb'
_ansible_check_mode=False
_ansible_no_log=False
_ansible_debug=False
_ansible_diff=False
_ansible_verbosity=0
_ansible_version=2.9.14
_ansible_module_name=explain_in_perl
_ansible_syslog_facility=LOG_USER
_ansible_selinux_special_fs=['''fuse''', '''nfs'''...(truncated line)]
_ansible_string_conversion_action=warn
_ansible_socket=None
_ansible_shell_executable=/bin/sh
_ansible_keep_remote_files=False
_ansible_tmpdir=/root/.ansible/tmp/ansible-tmp-1616727.1178262-8829-12373418/
_ansible_remote_tmp=~/.ansible/tmp'
```

Down With Slide Decks!

- Let's get our hands dirty with some code.

Next Steps

Where Do We Go?
Where Do We Go?



Where Do
We Go Now?

Next Steps with your Custom Content



Modules

The core function of Ansible, interacting with all of your devices, nodes, servers, appliances.

Playbooks

An organized series of tasks, using modules to perform consistent actions to your environment.

Roles

Playbooks, organized into orderly formats to distribute repeatable processes.

Collections

Distributed content including all of the above.

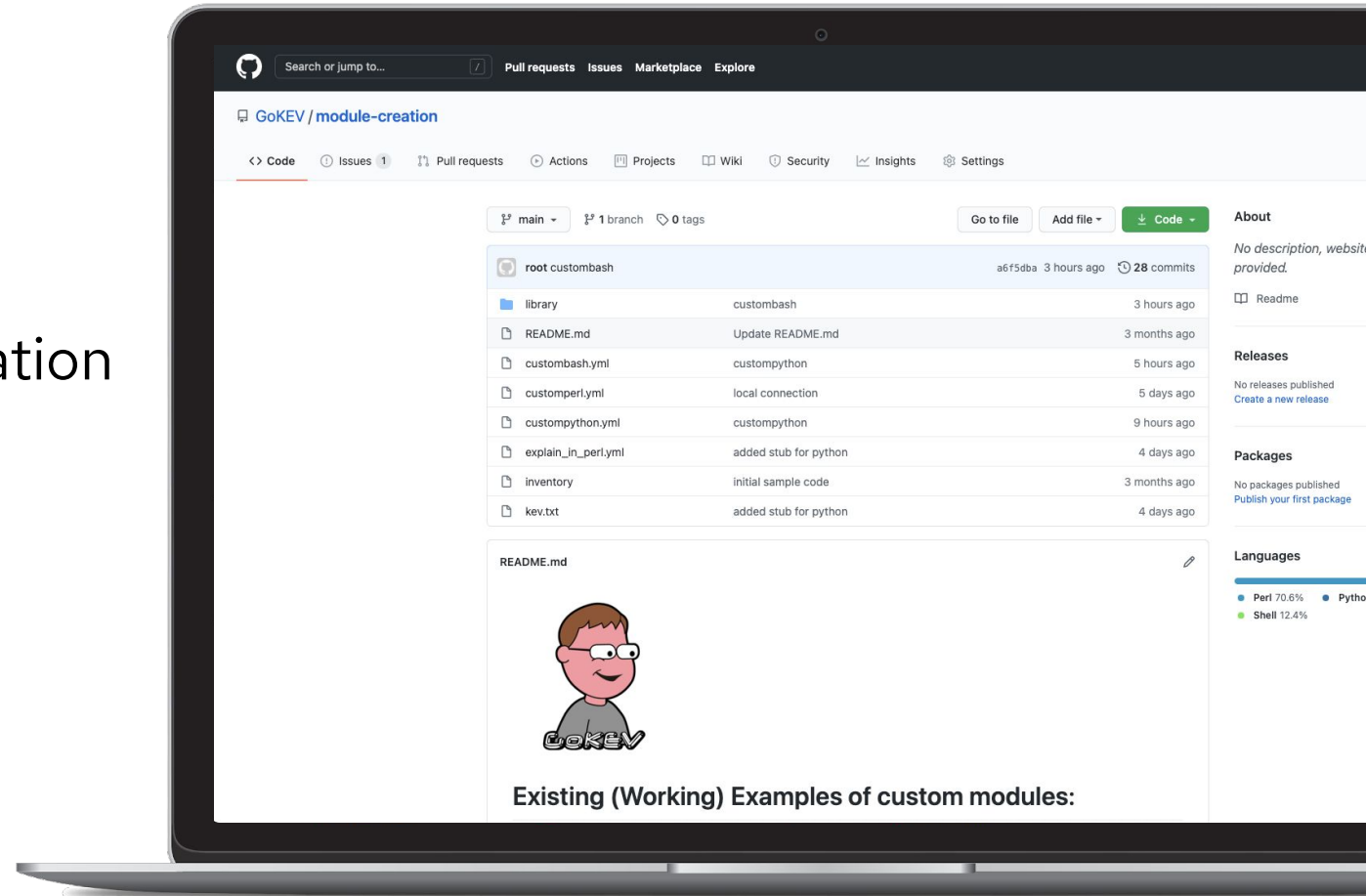
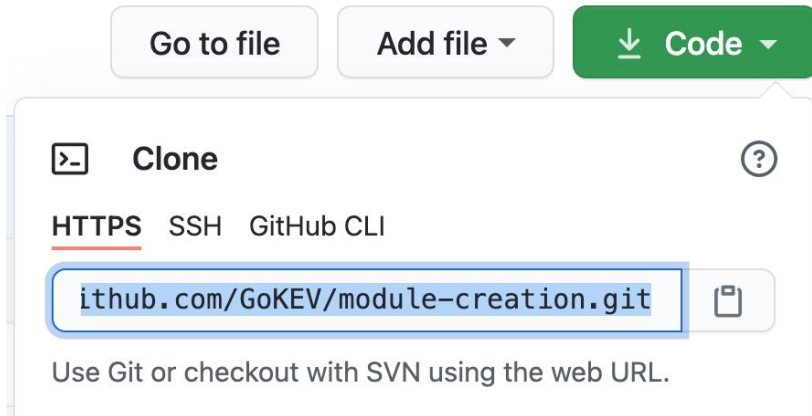
Where is the code?

Github

All code available in public Github.

(tested and verified to work on RHEL 8)

<https://github.com/GoKEV/module-creation>



```
[module-creation]# git clone https://github.com/GoKEV/module-creation.git
```

WINDOWS NOTES

- Windows new module development
- When creating a new module there are a few things to keep in mind:
- Module code is in Powershell (.ps1) files while the documentation is contained in Python (.py) files of the same name
- Avoid using Write-Host/Debug/Verbose/Error in the module and add what needs to be returned to the \$module.Result variable
- To fail a module, call \$module.FailJson("failure message here"), an Exception or ErrorRecord can be set to the second argument for a more descriptive error message
- You can pass in the exception or ErrorRecord as a second argument to FailJson("failure", \$_) to get a more detailed output
- Most new modules require check mode and integration tests before they are merged into the main Ansible codebase
- Avoid using try/catch statements over a large code block, rather use them for individual calls so the error message can be more descriptive
- Try and catch specific exceptions when using try/catch statements
- Avoid using PSCustomObjects unless necessary
- Look for common functions in ./lib/ansible/module_utils/powershell/ and use the code there instead of duplicating work. These can be imported by adding the line #Requires -Module * where * is the filename to import, and will be automatically included with the module code sent to the Windows target when run via Ansible
- As well as PowerShell module utils, C# module utils are stored in ./lib/ansible/module_utils/csharp/ and are automatically imported in a module execution if the line #AnsibleRequires -CSharpUtil * is present
- C# and PowerShell module utils achieve the same goal but C# allows a developer to implement low level tasks, such as calling the Win32 API, and can be faster in some cases
- Ensure the code runs under Powershell v3 and higher on Windows Server 2008 and higher; if higher minimum Powershell or OS versions are required, ensure the documentation reflects this clearly
- Ansible runs modules under strictmode version 2.0. Be sure to test with that enabled by putting Set-StrictMode -Version 2.0 at the top of your dev script
- Favor native Powershell cmdlets over executable calls if possible
- Use the full cmdlet name instead of aliases, for example Remove-Item over rm
- Use named parameters with cmdlets, for example Remove-Item -Path C:\temp over Remove-Item C:\temp