Ansible in 5 days

Denis Zuev, Artemii Kropachev, Dmitrii Mostovshchikov

December 7, 2017

0.1 Why Git

At the moment of writing this book, we have been looking for a tool that allows us to share this book solutions, and other related files with our readers. It has to be popular, easy to learn and use, and free to use. We have been using git for several years already, so the choice was very obvious. And for those who is unfamiliar with `Git`, we hope that you will learn and start using it on a daily basis.

0.1.1 About Git

`Git` is a distributed version control system developed as storage for source code and tracking changes. This is absolutely free software distributed under `General Public License (GPL)`. `Git` allows a number of people work on same projects independently.

`Git` is a very simple yet powerful tool that has been actively used by Developers, Engineering, DevOps and other communities for years. Since `Git` is not main topic of this book, we are not going to go use and explore `Git` full functionality, but rather use it to simply share playbooks, roles and other Ansible related files we use in this book.

0.1.2 Installing Git

Installing `Git` is quite easy. Below are 3 examples how to do it on 3 main Operating Systems you might working on.

`Git` is available on almost every Linux distributive and also on Solaris, FreeBSD, Windows, Mac OS X and many other systems. Depending on your current OS install Git using a proper way.

MacOS`$ brew install git` CentOS`$ sudo yum install git` Ubuntu`$ sudo apt install git`

`Verify that Git is installed on your system`$ git –version

0.1.3 Cloning a Github repository

Download this book repo from github.com. It includes all solutions for all the Chapters of this book. If you feel uncomfortable at any point while reading this book, you can refer to it. The folder structure is quite intuitive, so you should not have any problems finding the right solution.

```
$ mkdir repos
```

That is what you need from git for this book. We are going to work with git on a few more topics to give you an idea about how you can use it on your daily basis.

0.1.4 Creating an own Git repository

Get yourself registered on http://github.com. It should be fairly easy and straightforward process. Once you are done:

1. Press + and then New repository at the top-right corner of github webpage
2. Specify Repository Name, like my_project1
3. Leave the rest of the settings by default and then press Create Repository button.

For more information, follow the link `https://help.github.com/articles/create-a-repo/`

At this point you are ready to clone your own repository. Copy the URL that that should look like this `git clone https`

```
$ mkdir repos; cd repos
Cloning into my_project1...
```

0.1.5 Configuring Git

Before you start working with git repositories, we need to define our email and username. This minimal configuration is required to start working with `Git`:

0.2 Error handling

0.2.1 Errors in plays

Ansible normally has defaults that make sure to check the return codes of commands and modules and it fails fast forcing an error to be dealt with unless you decide otherwise. Sometimes a command that returns different than 0 isnt an error. Sometimes a command might not always need to report that it changed the remote system. This section describes how to change the default behavior of Ansible for certain tasks so output and error handling behavior is as desired.

0.2.2 Ignoring Failed Commands

By default, if a task fails, a play is aborted; however, this behavior can be overridden by skipping failed tasks. To do so, the `ignore_errors` keyword needs to be used in a task.

0.2.3 Force execution of handlers

By default, if a task that notifies a handler fails, the handler will be skipped as well. Administrators can override this behavior by using the `force_handlers` keyword in task. This forces the handler to be called even if the task fails. The following snippet shows hows to use the `force_handlers` keyword in a task to forcefully execute the handler even if the task fails.

0.2.4 Override the failed state

A task itself can succeed, yet administrators may want to mark the task as failed based on specific criteria. To do so, the `failed_when` keyword can be used with a task. This is often used with modules that execute remote commands and capture the output in a variable. For example, administrators can run a script that outputs an error message and use that message to define the failed state for the task. The following snippet shows how the `failed_when` keyword can be used in a task.

0.2.5 Override the changed state

When a task updates a managed host, it acquires the changed state; however, if a task does not perform any change on the managed node, handlers are skipped. The `changed_when` keyword can be used to override the default behavior of what triggers the changed state. For example, if administrators want to restart a service every time a playbook runs, the `changed_when` keyword can be added to the task. The following snippet shows how a handler can be triggered every time by forcing the changed state.

0.2.6 Error handling in blocks

In playbooks, blocks are clauses that enclose tasks. Blocks allow for logical grouping of tasks, and can be used to control how tasks are executed. For example, administrators can define a main set of tasks and a set of extra tasks that will only be executed if the first set fails. To do so, blocks in playbooks can be used with three keywords:

- `block`: Defines the main tasks to run
- `rescue`: Defines the tasks that will be run if the tasks defined in the block clause fails.
- `always`: Defines the tasks that will always run independently of the success or failure of tasks defined in the block and rescue clauses.

The following example shows how to implement a block in a playbook. Even if tasks defined in the block clause fail, tasks defined in the `rescue` and `always` clause will be executed.

0.2.7 Aborting the play

Sometimes its desirable to abort the entire play on failure, not just skip remaining tasks for a host. The `any_errors_fatal` play option will mark all hosts as failed if any fails, causing an immediate abort