Ansible in 10 minutes or less

I just remember a recent argument I had with someone about automation. It's unbelievable, how many things are still done manually on a widespread scale, not leveraging the possibilities at all. Especially with so many frameworks available to help out, sticking to "the old way" ain't just cool any more.

So let's quickly look at Ansible, and how we can be up and running for even simple task automation in 10 minutes or less.

While ansible has just recently be acquired by RedHat, it is definitely not a Linux-only tool and can be used as well to manage BSDs or even network devices.

One can use Ansible standalone or in combination with Ansible Tower, which just adds more bells and whistles, some APIs and inventory management.
But for simple tasks, the standalone will do just fine. All you need is a management station, from where Ansible can take off.

On FreeBSD, these packages are recommended to install:
pkg install sshpass python python27 ansible py27-paramiko redis py27-redis

Enable redis (a caching engine, somewhat similar to memcached):
echo 'redis_enable="YES"' >> /etc/rc.conf
service redis start

Add a directory which will hold your ansible scripts, so called playbooks:
mkdir -p /var/ansible/playbooks/roles

Edit **/usr/local/etc/ansible/ansible.cfg** and change the settings as shown below. Change the **roles_path** directory as needed.
transport = paramiko

fact_caching = redis
fact_caching_timeout = 3600
fact_caching_connection = localhost:6379:0

roles_path = /var/ansible/playbooks/roles

Now create a new file at **/usr/local/etc/ansible/hosts** and add some hosts.
You can start off with a very simple host group, where you just add some hosts.
[servers]
fbsdhost1.localdomain.local
rhelhost2.localdomain.local

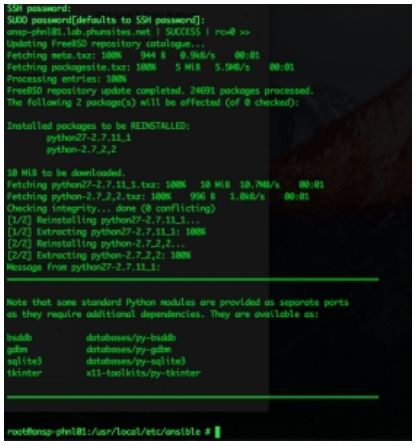
That's it. Now that the hosts are registered in the inventory, we're almost ready to get off the ground.
But first, there's one important thing: All hosts must be reachable and have SSH enabled, also, a remote-login using a (preferrably) unprivileged account with **sudo**-privileges must exist. For this example, a remote account called **sulogin** is assumed.

Another important thing is, that Python2.7 must be installed on the target host as well.
While a RedHat-based system typically has Python installed, FreeBSD has not.

Luckily Ansible allows to also send raw commands to a target host, this way it can be used to bootstrap itself like so:
ansible servers -l fbsdhost1.localdomain.local --user=sulogin -k --ask-become-pass -b -m raw -a "pkg install -y -f python27 python"

The command above will select the **servers** host group, but limit itself to given hostname. Additionally, it'll ask for the remote user's password, and the sudo password as well. The **raw** module is selected, scheduled to run the given **pkg** command.

The result of which looks somewhat like this:



As soon as Python is installed, you may us the **ping** module to see if everything is fine.

ansible all -m ping --user=sulogin -k

This will instruct Ansible to run a simple check on each target host, which result in something like this:

```
fbsdhost1.localdomain.local | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
rhelhost2.localdomain.local | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Now, let's put this to some good use.

The key to leverage Ansible is to use so-called playbooks, which implement the actual automation logic.
But first, some directories need to be created on the management station.

```
mkdir -p /var/ansible/playbooks/common
cd /var/ansible/playbooks/common
mkdir defaults files files/ssh handlers meta tasks templates
```

Now, let's look at a simple use-case: We want Ansible to take care for installing some ssh keys, disable remote root login, add an ssh banner, and make sure that the SSH service is actually restarted if a change occured.

In an ordinary world, this means creating some directories, copying files to their destination, set some permissions and change config files.

So, let's start by adding two ssh public key files to the new **/var/ansible/playbooks/common/files/ssh/** directory:

Also create a new ssh banner file:

```
echo "NO UNAUTHORIZED ACCESS TO THIS HOST PLEASE" > /var/ansible/playbooks/common/files/ssh/sshd-banner
```

The directory should now look somewhat like this:

```
ls -1 /var/ansible/playbooks/common/files/ssh/sshd-banner
id_rsa_ansible.pub
id_rsa_manager.pub
sshd-banner
```

Now go to **/var/ansible/playbooks/common/tasks** and create a new file called **sshdsetup.yml** with this contents:

```
---
- hosts: all
  remote_user: sulogin
  become: yes
  tasks:

  - name: install ssh keys
    authorized_key: user=sulogin key={{ item }} state=present manage_dir=yes
    with_file:
      - /var/ansible/playbooks/common/files/ssh/id_rsa_ansible.pub
      - /var/ansible/playbooks/common/files/ssh/id_rsa_manager.pub

  - name: copy ssh banner
    copy: src=/var/ansible/playbooks/common/files/ssh/sshd-banner dest=/etc/ssh/sshd-banner mode=0644 owner=root

  - name: enable ssh banner
    replace: dest=/etc/ssh/sshd_config regexp='#?Banner none' replace='Banner /etc/ssh/sshd-banner' backup=yes
    register: banner_enabled

  - name: root lockdown
    replace: dest=/etc/ssh/sshd_config regexp='#?PermitRootLogin yes' replace='PermitRootLogin no' backup=yes
    when: ansible_distribution == "CentOS"
    register: root_disabled
```

```
 - name: ssh restart
   service: name=sshd state=restarted
   when: banner_enabled.changed or root_disabled.changed
```

The file, a structured YAML file, is actually easy to read, and implements just the steps as outlined before.
Each section implements a single step. The first loops through the key files provided, and registers them with the ~/.ssh/authorized_keys file of user **sulogin**. The command will make sure that the directory is create if missing, and proper permissions are assigned. Also, the key will be registered just once.

The second step just copies the banner file to the new location. Ownership and permissions are provided, so no guesswork must be made.

Next is to actually enable the banner. The **replace** command knows how to apply dynamic rewriting using regurlar expressions to a text file, including creating a backup copy. The **register** keyword registers a variable if (and only if) the file has been changed.

The root lookdown will also do some on-the-fly changing of the sshd_config file. It will effectively disable RootLogin, which is enabled on RedHat/CentOS flavors. The **when** condition clearly indicates, that this steps is to be carried on a host running CentOS (RedHat users should use the appropriate distro name here, of course).
Again, a variable is registered if the change has been carried out.

Last but not least, the sshd daemon is restarted if one of the previously mentioned variables exists.

Such a playbook is run like this:

```
ansible-playbook -k --ask-become-pass sshsetup.yml -l servers
SSH password:
SUDO password[defaults to SSH password]:

PLAY ***************************************************************************

TASK [setup] *******************************************************************
ok: [fbsdhost1.localdomain.local]
ok: [rhelhost2.localdomain.local]

TASK [install ssh keys] ********************************************************
changed: [fbsdhost1.localdomain.local] => (item=ssh-rsa)
ok: [rhelhost2.localdomain.local] => (item=ssh-rsa)

TASK [copy ssh banner] *********************************************************
ok: [fbsdhost1.localdomain.local]
changed: [rhelhost2.localdomain.local]

TASK [enable ssh banner] *******************************************************
ok: [fbsdhost1.localdomain.local]
changed: [rhelhost2.localdomain.local]

TASK [root lockdown] ***********************************************************
skipping: [fbsdhost1.localdomain.local]
```

```
changed: [rhelhost2.localdomain.local]

TASK [ssh restart] *************************************************************
skipping: [fbsdhost1.localdomain.local]
changed: [rhelhost2.localdomain.local]

PLAY RECAP *********************************************************************
fbsdhost1.localdomain.local : ok=4    changed=1    unreachable=0    failed=0
rhelhost2.localdomain.local : ok=5    changed=4    unreachable=0    failed=0
```

As you can see, it's dead simple to to write a playbook which does just plain simple automation, and works fine cross-platform in different *nix flavours.
And the best: You don't need to bother too much about the scripting logic.

Proof: Up and running in 10 minutes. Now head out and look for more in the Ansible documentation.