# Introduction to Puppet Labs

## Lab 0: Setup

Ensure that you are using the Vagrant VM. See the `README.md` for setup. This setup is **required** for this class.

You should have Puppet 4 installed, and be able to run:

```
$ puppet --version
4.4.0
```

as well as:

```
$ sudo puppet --version
```

## Lab 1: Hello, Puppet World!

Ensure you can apply `hello.pp` using Puppet. When complete, you should see the following:

```
Notice: Compiled catalog for client.example.com in environment production in
0.03 seconds
Notice: Hello, World!
Notice: /Stage[main]/Main/Notify[greeting]/message: defined 'message' as 'Hello,
Puppet World!'
Notice: Applied catalog in 0.01 seconds
```

## Lab 2: Creating a User

Create a user for yourself by using a resource declaration inside a new manifest.

```
user { '<yourname>' :
    ensure => 'present',
    uid => '2000',
    gid => '1000',
    comment => 'This is my comment',
    managehome => true,
}
```

What you should see:

```
Notice: Compiled catalog for client.example.com in environment production in
0.03 seconds
Notice: /Stage[main]/Main/User[will]/ensure: created
Notice: Applied catalog in 0.02 seconds
```

You can also verify by using:

```
$ getent passwd <username>
<username>:x:2000:1000:This is my comment:/home/<username>:/bin/bash
```

How did you finally get it to work? What did you need to change in how you applied the manifest from Lab 1?

## Lab 3: Using Puppet Parser to Find Syntax Errors

Use the Puppet parser tool to fix the syntax of a manifest without ever needing to apply it.

Run:

```
$ puppet parser validate manifests/validate_me.pp
```

And debug what you find. What do you notice about the results from the parser?

**HINT**: `validate_me.pp:2:56` would mean line 2, character 56.

## Lab 4: `puppet resource` Practice

Write and test single line bash commands for the following:

1. Retrieve the resource block for the `root` user
2. Retrieve data for the `ssh` service
3. MD5 hash of the `~/.bash_profile` file
4. Calculate the number of users on your entire system

You should use `puppet resource ...` as a starting point for all of these. You may have to do some processing of the results using bash.

## Lab 5: Hosts in `/etc/hosts`

Often times when provisioning a server we'd like to be able to set up hosts information.

For this lab:

1. Look up the Puppet host resource documentation
2. Create a host resource for a host `foobar.com` at IP address "1.2.3.4" with aliases `fooserver` and `fooweb`
3. Also, add a comment with some userful information

Apply and ensure that the entry (along with the comment) has been applied to `/etc/hosts`.

## Lab 6: Oh no! Using Filebucket

Suppose we realize we made a mistake with our last change to `/etc/hosts`. What now?

Puppet provides a simple way to retrieve prior versions of files replaced with Puppet via the filebucket.

```
# show list of files we have recently changed
$ sudo puppet filebucket -l list

# find the hash matching to the old version you'd like to restore
# and then restore it
$ sudo puppet filebucket -l restore /etc/hosts d2c3b4ba5e2eed78d25516f153351d6e

# verify the file is back to normal
$ cat /etc/hosts
```

Pretty cool feature.

## Lab 7: Install & Use: `puppet-lint` Package

You will:

1. Define two package resource blocks - `rubygems` is the first, and the second is `puppet-lint`. Google to find the provider for `puppet-lint`.
2. Ensure that `puppet-lint` package depends on `rubygems`.

Run `puppet-lint <any-manifest-file>` to see how closely your manifest file matches Puppet best practices (don't worry, it's pretty picky).

Now try running it in fix mode by appending `--fix` to the command.

Check the file - if you like what you see, feel free to use this all the time. Some companies will make a `puppet-lint` with `--fix` run mandatory before merging code!

## Lab 8: Custom Classes & Modules

### Lab 8a: Create a Class

1. Create a manifest called `users.pp` which contains two user resources and a group resource, i.e., let's create two users `john` and `sally`, both of whom are in a group called 'mygroup'
2. Apply your manifest to be sure it works
3. Use `puppet resource` to remove `sally`.
4. Verify sally has been removed using `id sally`
5. Make your code into a class called `users`
6. Apply your manifest
7. Verify that it did nothing–why?

### Lab 8b: Create a Custom Module

Navigate to `/etc/puppetlabs/code/modules`. This is your global directory where you can store Puppet modules.

1. Make a folder for your new module: `/etc/puppetlabs/code/modules/users`
2. Make two more folders: `/etc/puppetlabs/code/modules/users/examples`, `/etc/puppetlabs/code/modules/users/manifests`
3. Take the `users.pp` from Lab 7a and copy it into this module, renaming it: `cp /path/to/users.pp /etc/puppetlabs/code/modules/users/manifests/init.pp`.
4. Create another file `/etc/puppetlabs/code/modules/users/examples/init.pp` and add a single line: `include users`
5. Try applying `puppet apply /etc/puppetlabs/code/modules/users/manifests/init.pp` and verify nothing happens
6. Now go back to your original `users.pp` and overwrite the contents of the file with: `include users`
7. Apply that file: `puppet apply /path/to/users.pp`
8. Verify that user who was deleted previously has now been created!

Congrats you've just created a custom module!

We created a module, documented how to use in in the `examples` subfolder, and we finally included it in a manifest we were working on, and in a single line of code, can harness the class we built to add the desired users.

## Lab 9: Dependency Cycle Visualizations

Apply the file: `manifests/depcycle.pp`.

What happens?

We can graph visually what's going on here using:

```
$ puppet apply /vagrant/manifests/depcycle.pp --graph  # to graph it
```

Then use any `graphviz` compatible visualizer or simply use an online one like this one by pasting the contents of the resulting `.dot` file.

## Lab 10: Package-File-Service: NTP

Create a new module (similar to Lab 7) that has a class that implements the "trifecta" for NTP. Remember this means that we flow from package installation and have a service that reloads whenever there are configuration file changes!

You should use the template in `files/ntp.conf` and add both `time.nist.gov` and `time.apple.com` as time servers. Google around to find the exact syntax as well as where this file should go on CentOS.

By the end of this lab, you should be able to create a `manifests/start_ntp.pp` file and include only the line `include ntp` and from this, you should be able to apply and run.

Finally, verify `/etc/ntp.conf` has correct content and that `ntp` service is running by running `$ ntpq -p`.

**HINT**: Our file resource block will need to pull in our `ntp.conf` file when it is applied. Use the `source` option of the file resource as described here. Read this very carefully as there's a trick here.

We'll also enforce random ordering to ensure you've done everything correctly in terms of specifying ordering dependencies. You can do this with any Puppet apply command:

```
$ puppet apply <some-manifest> --ordering=random
```

So your final goal of this lab is to run:

```
$ puppet apply /vagrant/manifests/test_ntp.pp --ordering=random
```

and then, to verify success, you should run:

```
$ ntpq -p
```

and see something like the following:

```
[vagrant@client]$ ntpq -p
     remote           refid      st t when poll reach   delay   offset  jitter
 ==============================================================================
*time.apple.com  .GPSs.           1 u    5   64    1  130.374   -2.814   0.000
 host-24-56-178- .ACTS.           1 u    4   64    1   84.778   -2.103   0.000
```