Save and close the editor, then run:

```
# puppet apply /root/examples/user-absent.pp
notice: /Stage[main]//User[katie]/ensure: removed
notice: Finished catalog run in 0.07 seconds
```

Now run it again:

```
# puppet apply /root/examples/user-absent.pp
notice: Finished catalog run in 0.03 seconds
```

Cool: You've just written and applied your first Puppet manifest.□

Manifests

Puppet programs are called "manifests," and they use the `.pp` file extension.□

The core of the Puppet language is the resource declaration. A resource declaration describes a desired state for one resource.

(Manifests can also use various kinds of logic: conditional statements, collections of resources, functions to generate text, etc. We'll get to these later.)

Puppet Apply

Like `resource` in the last chapter, `apply` is a Puppet subcommand. It takes the name of a manifest file as its argument, and enforces the desired state described in the manifest.□

We'll use it below to test small manifests, but it can be used for larger jobs too. In fact, it can do nearly everything an agent/master Puppet environment can do.

Resource Declarations

Let's start by looking at a single resource:

```
# /root/examples/file-1.pp

file {'testfile':
  path    => '/tmp/testfile',
  ensure  => present,
  mode    => 0640,
  content => "I'm a test file.",
}
```

The complete syntax and behavior of resource declarations are documented in the Puppet