A Gentle Introduction to Linux and Unix

Goals

A brief history of Unix and Linux

How people interact with Linux

Packages, services, and files

Tying it in with Puppet

Operating Systems

Windows, Mac OS, Linux, Unix: All offer a way for a user to interact with a computer

They provide an interface: a GUI you can use with a mouse or a command line (CLI) you can use with a keyboard

Most computers can run many kinds of operating systems

Free/Open Source Software ("FOSS")

Nutshell: "The source code for software should be freely available to all."

"Free Software" is the branch of "FOSS" more interested in ethical questions

"Open Source" was coined to make these ideas more business friendly

Puppet is open source software

In the Beginning Was UNIX

Created in the '70s

Reflects the technology of the time (plain text terminals, academic in flavor)

Tended to think of problems in terms of moving text around between small programs

Pipes and Filters

```
$ echo "The quick brown fox jumped over the lazy dog!"
```

```
The quick brown fox jumped over the lazy dog!
```

```
$ echo "The quick brown fox jumped over the lazy dog!" | sed s/foo/fox/
```

```
The quick brown foo jumped over the lazy dog!
```

```
$ ls
```

```
foo.txt
bar.txt
baz.txt
zoobar.txt
foobar.txt
barbaz.txt
```

```
$ ls | grep bar | sort
```

```
bar.txt
foobar.txt
zoobar.txt
```

UNIX Becomes Unix

A number of companies licensed UNIX, over time it fragmented into numerous, sometimes incompatible flavors

Legal battles created uncertainty about some of these versions

Despite this, Unix remained dominant in data centers and for networked applications

Along Comes Linux

1991: Linus Torvalds uses a small, academic Unix variant to scaffold his own version of Unix

Linux leverages existing software from the Free Software movement to provide a complete operating system

Software companies begin to add value to Linux by packaging easier-to-install versions of it and refining the user experience ("distributions" or "distros")

Linux Distributions

There are several major Linux distributors:

Red Hat (CentOS, Fedora)

Debian

Ubuntu

SUSE

Unix Versions

AIX (IBM)

HP/UX (Hewlett-Packard)

FreeBSD, OpenBSD, NetBSD (community)

Solaris (Oracle/Sun)

Mac OS (BSD Unix inside)

Unix/Linux Culture

The balance has shifted heavily toward "Linuxisms" as commercial Unix fades

Still text-centric

Traditionalist

Heavily informed by open source and free software movements

How a Linux Server Is Organized

Most administrators interact via a text interface

Servers provide services: email, web, file storage, time servers

Services are just programs

You install a package to run a service based on its configuration files

Packages

Packages are bundles of files and software needed to provide a service

Packages are installed and managed by a package manager

Packages provide all the needed parts of a given service

If a program needs other programs to run, those dependencies are satisfied by the package manager

Configuration Files

In the Unix world, even if there's a graphical tool to change the way a program works, it is probably using a text-based configuration file.

In the Windows world, most configuration is handled in a "Registry," which is not meant to be directly read by people.

In the Mac world, configuration is often handled in text files, but sometimes there are computers-only files, too.

Looking at a Service

Apache is a common web server we'll use as an example

We have to *install* an Apache package, which provides the Apache program and basic configuration files

We have to *configure* Apache to run the way we want it to run

We have to *start* the Apache service

Installing Apache

Depending on the version of Linux or Unix we're using, there are a number of ways to install a package

We're on a Debian system, so we'll use the `apt-get` command:

```
apt-get install apache2
```

The `apt-get` command downloaded the Apache package for us, checked to make sure there were no other *dependencies*, then it put the Apache program files where they need to be. 

Configuring Apache

Once the Apache package is installed, we have to configure it.

By convention, most configuration files for a Unix or Linux system live in a directory called `etc`

Most configuration files are plain text

Configuration files come in a variety of formats and styles 

Running Apache

Now that Apache is installed and configured, we have to start the Apache service.

Linux and Unix have a variety of tools for starting and stopping services

Unix systems have a collection of programs that help make sure a service is running correctly

Once a service is running, it's not usually very interactive. If you have to reconfigure it, you will have to stop it and restart it.

Connecting It All With Puppet

As we just saw, there's a lot to keep track of:

There are lots of configuration files

We have to keep track of the packages we need to provide services

We have to make sure a service is running

It doesn't get any easier when have dozens or hundreds of servers

What Puppet Does

Puppet provides a simple language (a "DSL") that makes it easy to express how a given computer should be configured

Puppet can install, remove and upgrade packages

Puppet can start and stop services

Puppet can set the contents of configuration files

How Puppet Works

A "puppet master" contains information about the configuration of all the computers it's responsible for

A "puppet agent" is a computer that depends on a puppet master for its configuration information

By default, a puppet agent checks in with its puppet master once every 30 minutes, the puppet master makes sure the agent's system is still configured correctly: packages installed, files configured correctly, services running

What Puppet Looks Like

```
package { 'apache2':
  ensure => installed,
}

file { '/etc/apache2/apache.config':
  ensure => present,
  content => "//SomeConfigurationText//",
  require => Package['apache2'],
}

service { 'apache':
  ensure  => running,
  enable  => true,
  require => File['/etc/apache2/apache.config'],
}
```

Why Puppet Works for People

One place to ensure systems are configured properly

Makes it possible to bring up new machines quickly and easily

Versionable configuration

Cuts through differences with a relatively simple configuration language

What's Next?

The Puppet Fundamentals course offers hands-on training where you get to see Puppet Enterprise in action, write your own Puppet code, and learn how to create reusable Puppet modules.