
# LCFG Basics

Kenneth MacDonald
<K.MacDonald@ed.ac.uk>

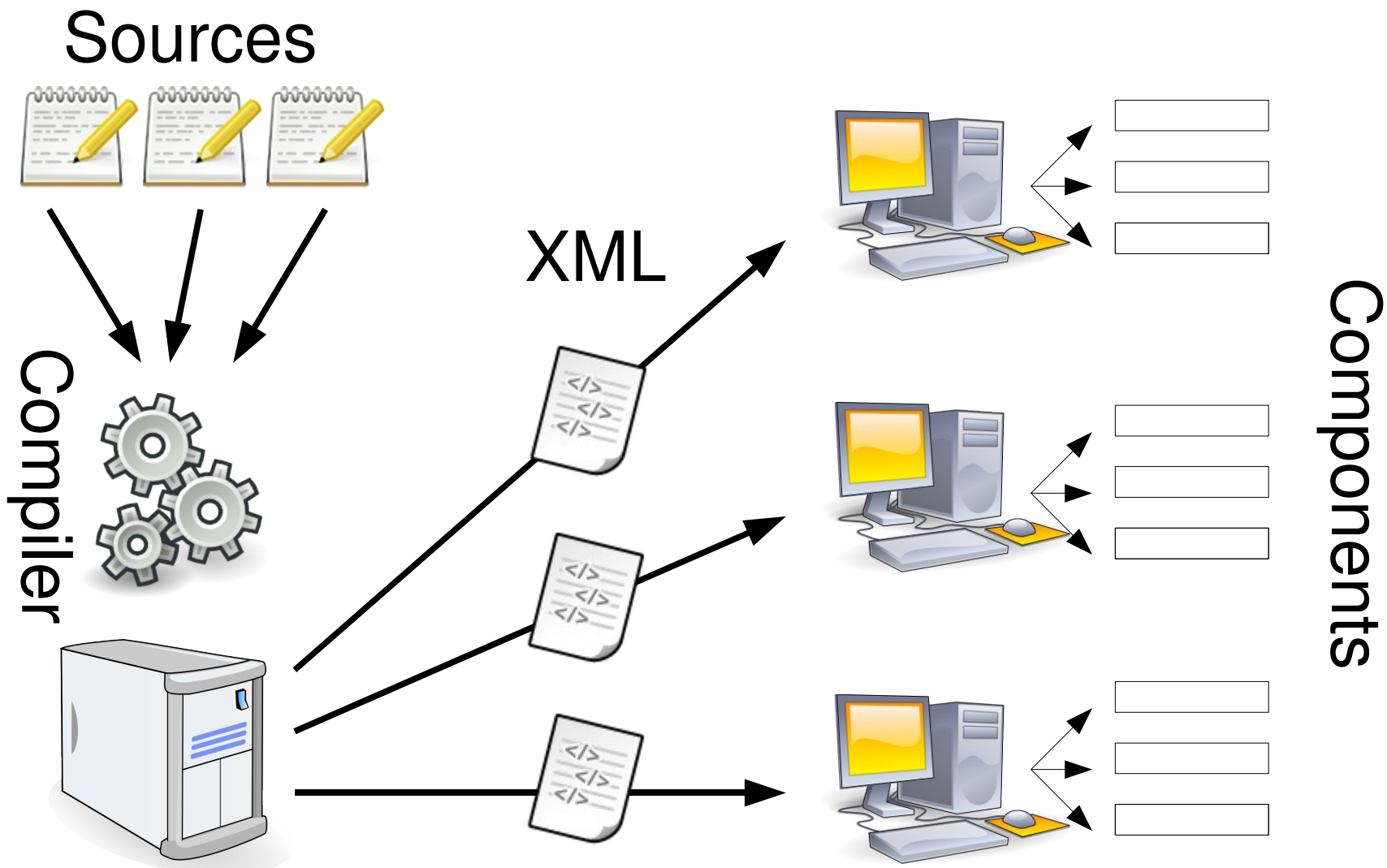
# Agenda

- History
- Architecture
- Uses
- Syntax
- Break
- Worked Examples

# History

- Started in 1994 (LISA 8 paper) in the Computer Science department of The University of Edinburgh.
- Production tool (1000+ machines) and research testbed.
- Capable of very comprehensive configuration but (traditionally) steep entry curve.
- See http://www.lcfg.org/

# Architecture



# LCFG “Component”

- A set of resources – basically key/value pairs.
- Optional templates for config files
- Optional control code, to manage daemons,
  - For example, stop/start/configure, based on a ...
  - framework provided by LCFG – shell or perl.

# The LCFG Compiler

- Makes the XML profile by processing simple text files containing the resource specifications.
- Handles “classing” to specify resources for groups (“include” files).
- Handles (some) conflicts between resource specifications for different groups.
- More complex things like relating resources between client and server machines (spanning maps).

- Notice that all resources for all hosts come from the central LCFG (web) server.
- This machine holds all the information about the configuration of all machines.
- Whenever things are changed on the server, the clients pick up the change as soon as possible (usually immediately).
- We can change the configuration of whole groups of machines by changing common files.

# The Client Component

- Watches for change notifications from a server.
- Fetches the resources from a web server (in a simple XML format called the “profile”).
- Tells the components whose resources have changed, so that they can “reconfigure”.

# “Heavyweight” LCFG

- LCFG is usually presented from the “top down” -
  - LCFG manages the whole site installation (including servers and relationships).
  - LCFG is the only thing which changes the configuration of any machine (prescriptive).
  - Clients run ~50 “components” ...

# But ...

- LCFG is capable of this “top down” approach, which has lots of benefits, but ...
  - This is very committing for people starting out.
  - It is difficult to learn the system starting in this way.

# “Lightweight” LCFG

- It is possible to use LCFG in a very “lightweight way” -
  - One or two components run on each client to manage specific configuration files.
  - Other parts of the configuration are managed in some other way.
  - This can gradually be extended to encompass more of the configuration as and when required and understood
  - Unlike simpler tools, this is possible without completely changing the tool.

The whole configuration task is now “reduced” to managing the configuration data on the server

# LCFG Syntax

- Setting resources
- Mutating resources
- Including other files
- Using the C Preprocessor

# Setting Resources

```
component.resource value
```

# Setting Resources

```
client.poll 10m+30s
```

# Mutating Resources

```
!component.resource MACRO(expression)
```

# Mutating Resources

```
!server.srcpath mADD(/tmp)
```

# Including Headers

```
#include <header.h>
```

# C Preprocessor

```
/* Linux clients need tweaks */
#ifdef LINUX
#include <linux.h>
#endif /* LINUX */
```

# Documentation

- The Complete Guide to LCFG
- Unix manual page for each installed component on a system
  - man lcfg-file
- Generated documentation for all components on the web site

# Hands On

- Start the VMWare Player image
- Log in as user lcfgfc6, password lcfgfc6
- Run startx
- Run sudo bash in one of the xterm windows
- Change directory to /var/lcfg/conf/server/source
- Our profile source is a text file called localhost

# Initial source profile

- The ./localhost file is our source profile
- Look at its contents
  - It is a plain text file
  - It has a C style comments
  - It includes several “header” files
- We will be changing this file in the coming examples ...

# Example 1-1

- We have been tasked to maintain a custom “message of the day” to all our clients
- This is done by placing the message in a file called /etc/motd
- What does this file currently contain?

# Example 1-2

- Copy /root/workshop/part1/example1 to ./localhost
- Check the contents of /etc/motd
  - Has it changed?
- Look at the localhost file contents
  - The top of the file is as before
- We used the file component to manage our file
- How did it actually happen?

# Example 1-3

- In the other window change directory to /var/lcfg/log
- Look at ./server
  - See the entry for the compilation of our profile
- Look at ./client
  - See the entry for our new profile and the file component’s configure method being run
- Look at ./file
  - See our file being managed

# Example 2

- Look at ls -l /etc/motd
  - Group lcfg?
- Copy example2 to ./localhost
- Check the three log files again
- Look at ls -l /etc/motd
  - That’s better!

# Example 3

- All computers have the same message
  - Users need to know which computer they logged on to!
- Copy example3 to ./localhost
- Check contents of /etc/motd
- The computer name has been merged into the message
- Any LCFG resource can be referenced by any other resource

# Example 4

- The file component can manage more than just plain text files
- Copy example4 to ./localhost
- Look at the source profile
  - Creates a directory called /etc/message_of_the_day
  - Creates a symbolic link inside the directory
- Check if the symbolic link is really there

# Example 5-1

- The file component can also use external templates to merge with resources to manage managed files
- Copy sshd_config.tmpl to /root/sshd_config.tmpl
- Look at the template, diff it with /etc/ssh/sshd_config
  - The extra lines control the file component
  - The port number is only managed if the resource is set

# Example 5-2

- Copy example5 to ./localhost
- Look at the source profile
- Look at /etc/ssh/sshd_config
  - Our LCFG comment is near the top
  - No managed port setting though
- We didn’t set the resource in the source profile!
- Try ssh’ing to localhost to check sshd is working

# Example 6-1

- Copy example6 to ./localhost
- Look at the source profile
- It tells sshd to listen on port 222 rather than the default 22
- Check /etc/ssh/sshd_config for the port setting
- Try ssh -p 222 localhost
- Why doesn’t it work?

# Example 6-2

- Run /etc/init.d/sshd reload
- Now try ssh -p 222 localhost
- Now try ssh localhost
- The file component only goes so far ...


# LCFG Basics

Kenneth MacDonald
<K.MacDonald@ed.ac.uk>

# Agenda

- History
- Architecture
- Uses
- Syntax
- Break
- Worked Examples

# History

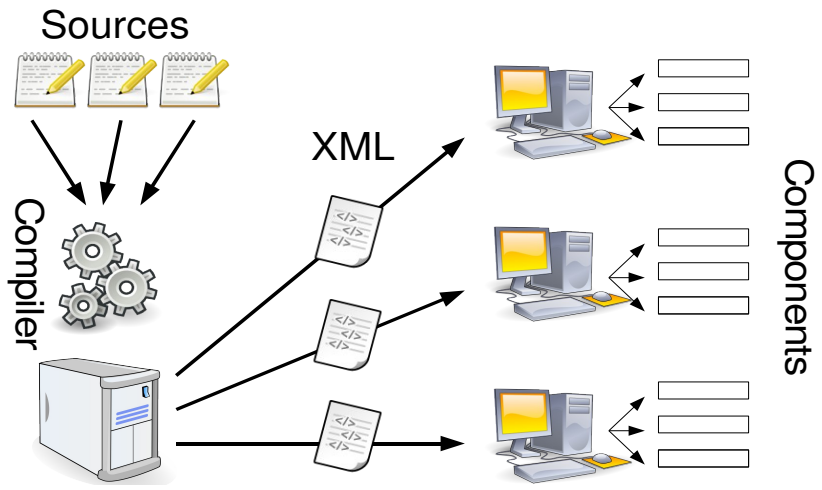
- Started in 1994 (LISA 8 paper) in the Computer Science department of The University of Edinburgh.
- Production tool (1000+ machines) and research testbed.
- Capable of very comprehensive configuration but (traditionally) steep entry curve.
- See http://www.lcfg.org/



Some external use but low profile - unlike some things, never promoted.

The University of Edinburgh is now deploying MacOS X and Linux desktops via LCFG across the University.

# Architecture




4

At least one source file per client named after its hostname. Other source files specify classes of clients. For example, all clients in one department share the same mail smart relay. Yet more source files configure aspects or options available to be included. For example, run an LCFG server or install a third party application.

The compiler generates an XML profile for each client, and notifies the client when it changes (single UDP packet on port 732).

Upon notification (and polling) the client retrieves the XML profile via HTTP and stores the contents in a local database. Each LCFG component is notified of resources which have changed.

The time from committing a change to a source file to the client being reconfigured is usually a matter of seconds.

# LCFG “Component”

- A set of resources – basically key/value pairs.
- Optional templates for config files
- Optional control code, to manage daemons,
  - For example, stop/start/configure, based on a ...
  - framework provided by LCFG – shell or perl.



We use the term resource rather than variable. Resources are owned by components and must be defined in their schema. Unknown resources throw errors in the compiler. The LCFG framework defines many standard resources which all components inherit.

A component’s executable code is stored in a file named the same as the component in the `/usr/lib/lcfg/components` directory. “Helper” programs are sometimes used to keep the shell or perl code simple.

Components define at least the **start**, **stop** and **configure** methods.

# The LCFG Compiler

- Makes the XML profile by processing simple text files containing the resource specifications.
- Handles “classing” to specify resources for groups (“include” files).
- Handles (some) conflicts between resource specifications for different groups.
- More complex things like relating resources between client and server machines (spanning maps).



If we wanted to, we could just list all the resources for each machine separately.

It’s more maintainable to build a library of options which are included where appropriate,

- Notice that all resources for all hosts come from the central LCFG (web) server.
- This machine holds all the information about the configuration of all machines.
- Whenever things are changed on the server, the clients pick up the change as soon as possible (usually immediately).
- We can change the configuration of whole groups of machines by changing common files.

# The Client Component

- Watches for change notifications from a server.
- Fetches the resources from a web server (in a simple XML format called the “profile”).
- Tells the components whose resources have changed, so that they can “reconfigure”.



When the server compiles a new XML profile it sends a UDP notification packet to port 732 on the client. If this packet doesn’t get through (firewalls, client offline, etc.) then the client will check for a new XML profile at start up and at an configurable polling interval.

The client processes the XML profile and stores component resources in a database which the LCFG framework automatically makes accessible to all components. The list of packages (if defined) is handled separately and is stored in a text file for whichever component (or other system service) is responsible for package management.

The client component runs the configure method of each component which owns changed resources, although the method called for each component is configurable like everything else!

# “Heavyweight” LCFG

- LCFG is usually presented from the “top down” -
  - LCFG manages the whole site installation (including servers and relationships).
  - LCFG is the only thing which changes the configuration of any machine (prescriptive).
  - Clients run ~50 “components” ...



Informatics at the University of Edinburgh runs LCFG like this. Not only all end user clients, but all servers are installed and configured by LCFG.

Benefits include guaranteed configurations (reliability), and less work (no manual changes).

You don’t have to do this - you can use it just to configure bits of your clients

# But ...

- LCFG is capable of this “top down” approach, which has lots of benefits, but ...
  - This is very committing for people starting out.
  - It is difficult to learn the system starting in this way.



A typical desktop client might have ~2500 resources, many of which could make the client unusable if they’re incorrectly set. New adopters quickly run into “cannot see the wood for the trees” syndrome.

# “Lightweight” LCFG

- It is possible to use LCFG in a very “lightweight way” -
  - One or two components run on each client to manage specific configuration files.
  - Other parts of the configuration are managed in some other way.
  - This can gradually be extended to encompass more of the configuration as and when required and understood
  - Unlike simpler tools, this is possible without completely changing the tool.



You can start out with less than a hundred resources, most of which are predefined by the LCFG framework. Most invalid values won’t disable your computer.

Your existing management framework can continue to coexist.

The whole configuration task is now “reduced” to managing the configuration data on the server

# LCFG Syntax

- Setting resources
- Mutating resources
- Including other files
- Using the C Preprocessor

# Setting Resources

```
component.resource value
```



The component name is separated from the resource name by a period, Therefore component and resource names cannot contain periods.

Everything after the space(s) following the resource name is assigned to the resource.

Some resources have can have multiple values keyed on “tags” allowing arrays to be constructed. The file component we will be looking at shortly uses tags. Tags are separated from resources by an underscore. For example,

```
component.resource_tag value
```

# Setting Resources

```
client.poll 10m+30s
```



This tells the client component to poll the server for changes to the XML profile every ten minutes with a 30 second randomisation.

The component schema files can optionally supply a validation macro for each resource. The compiler uses these to prevent invalid values getting into the XML profile. See pages 97 and 98 of the guide for more on validations.

# Mutating Resources

```
!component.resource MACRO(expression)
```



The LCFG compiler does not allow the same resource to be set twice. Instead its value must be mutated. The simplest mutation is to set a new value by using the mSET() macro..

A description of the built in mutation macros can be found on page 42 of the guide.

Advanced LCFG users can write their own mutation macros.

# Mutating Resources

```
!server.srcpath mADD(/tmp)
```



The mADD() macro appends something to a space separated list.

In this case, we’re telling the LCFG server component (compiler) that it should also look for client source files in the /tmp directory.

# Including Headers

```
#include <header.h>
```



“Header” files allow you to repeat the same code across multiple clients.

The LCFG syntax remains the same no matter the source file.

The compiler keeps track of the derivation of resources so you can see in which file it was set.

The compiler also records dependencies, so that if a header file changes, all clients which depend on it have their XML profiles updated.

# C Preprocessor

```
/* Linux clients need tweaks */
#ifdef LINUX
#include <linux.h>
#endif /* LINUX */
```



The source files are passed through the C preprocessor as part of the compilation process so you can make use of most of its features.

Comments are encouraged to document sources, but are not transferred to the clients.  C++ comments (//) are not supported.

You should use macros to guard header files from being included twice.

You can use #error to force a compilation to fail.

# Documentation

- The Complete Guide to LCFG
- Unix manual page for each installed component on a system
  - man lcfg-file
- Generated documentation for all components on the web site



The online copy of “The Complete Guide to LCFG” contains Unix manual pages for most components and supporting utilities.

The component manual page names have “lcfg-” prepended to the component name.

Full generated documentation for all core components can be found by following the [info] links on the release download pages on the website.

# Hands On

- Start the VMWare Player image
- Log in as user lcfgfc6, password lcfgfc6
- Run startx
- Run sudo bash in one of the xterm windows
- Change directory to /var/lcfg/conf/server/source
- Our profile source is a text file called localhost



You do not have to be running an X11 environment for this first hands on session, but you will for the next one.

The lcfgcf6 user has sudo access to everything.

The LCFG server is configured to look for client sources in /var/lcfg/conf/server/source which is the default location.

Ignore any files called install*. These are not used in this exercise, but are support files for installing a client from “bare iron”.

# Initial source profile

- The ./localhost file is our source profile
- Look at its contents
  - It is a plain text file
  - It has a C style comments
  - It includes several “header” files
- We will be changing this file in the coming examples ...



```
/* lcfg example host source profile */

#include <local/site.h>
#include <lcfg/os/minimal.h>
#include <lcfg/hw/vmware_ws5.h>
#include <lcfg/options/lcfg-server.h>
```

The initial source profile only includes some the header files. They contain resource settings which define a vanilla minimal LCFG client running in a VMWare virtual machine, but with the extra optional feature of running an LCFG server too.

We will not be looking inside any of these header files during this part of the workshop.

# Example 1-1

- We have been tasked to maintain a custom “message of the day” to all our clients
- This is done by placing the message in a file called /etc/motd
- What does this file currently contain?



Since we have only one client, we will be changing its individual profile. In the real world, we’d most likely be changing a header file which all of our clients include.

You should find that the `/etc/motd` exists, but is empty.

# Example 1-2

- Copy /root/workshop/part1/example1 to ./localhost
- Check the contents of /etc/motd
  - Has it changed?
- Look at the localhost file contents
  - The top of the file is as before
- We used the file component to manage our file
- How did it actually happen?

25

All the example files for this part of the workshop are stored in root's home in the `workshop/part1` directory. You need to be running as root to be able to write to the `/var/lcfg/conf/server/sources` directory.

`/etc/motd` should now contain "`Welcome to the LCFG tutorial.`".

Here's the relevant new lines in the `localhost` file ...

```
!file.files          mADD(example)
file.file_example    /etc/motd
file.type_example    literal
file.tmpl_example    Welcome to the LCFG tutorial.
```

We tag the file resources with "example" to differentiate from other files the file component may be managing.

First we tell the file component that we have another file to manage and tell it what we will be using as its tag. Then we can supply the full path to the file, the type of the file and its contents (since it's of type "literal").

# Example 1-3

- In the other window change directory to /var/lcfg/log
- Look at ./server
  - See the entry for the compilation of our profile
- Look at ./client
  - See the entry for our new profile and the file component's configure method being run
- Look at ./file
  - See our file being managed



Components by default (and convention) log to `/var/lcfg/log/<component>`.

The server log should have something like

```
05/06/07 12:34:56:   processing: localhost [1/1, pass 1]
05/06/07 12:34:56:               0 error(s), 0 warnings(0) (XML published)
```

showing the compilation was successful and the XML profile passed to the web server.

The client log should have something like

```
05/06/07 12:34:57:   new profile:
http://localhost.localdomain/profiles/localdomain/localhost/XML/profile.xml
05/06/07 12:34:57:     last modified Tue June  5 12:34:56 2007
05/06/07 12:34:57:   profile accepted: 1811efa86304e96a97716f769b97c7654
05/06/07 12:34:57:   reconfiguring component: file.configure
05/06/07 12:34:57:     [OK] file: configure
```

The client was notified of the new profile by the server (UDP:732), downloaded it and ran the configure method of the file component. It also stored an identifier for the particular version of the profile in case it received a later notification from another server.

The file log should have something like

```
05/06/07 12:34:57:   >> configure
05/06/07 12:34:57:      configuration changed: /etc/motd
```

# Example 2

- Look at ls -l /etc/motd
  - Group lcfg?
- Copy example2 to ./localhost
- Check the three log files again
- Look at ls -l /etc/motd
  - That's better!



Originally the file had root as its owner and group. Now the group is set to "lcfg".

Remember the source files are in `/root/workshop/part1`.

The example2 file adds the following lines

```
file.owner_example root
file.group_example root
file.mode_example  0644
```

Note that the mode entry must start with a leading zero.

# Example 3

- All computers have the same message
  - Users need to know which computer they logged on to!
- Copy example3 to ./localhost
- Check contents of /etc/motd
- The computer name has been merged into the message
- Any LCFG resource can be referenced by any other resource



The only change in this example is to the contents of the motd file.

```
file.tmpl_example    Welcome to the LCFG tutorial (running on <%profile.node%>).
```

The value of the referenced resource is used to customise the contents of the file. This process is done by the LCFG compiler on the server, not by the client.

See page 47 of the guide for more on referencing resources.

# Example 4

- The file component can manage more than just plain text files
- Copy example4 to ./localhost
- Look at the source profile
  - Creates a directory called /etc/message_of_the_day
  - Creates a symbolic link inside the directory
- Check if the symbolic link is really there



```
!file.files          mADD(example1)
file.file_example1   /etc/message_of_the_day
file.type_example1   dir
file.owner_example1 root
file.group_example1 root
file.mode_example1   0755
!file.files          mADD(example2)
file.file_example2   <%file.file_example1%>/motd
file.type_example2   link
file.tmpl_example2   /etc/motd
```

We chosen tags example1 and example2 for the directory and symbolic link. Their type resources are set to “dir” and “link”.

The directory does not need a “tmpl” resource. The symbolic link uses the “tmpl” resource to specify its target, and the directory’s path for its location by referencing the appropriate resource.

Even although the original `/etc/motd` file is not configured in this profile notice that it has not been put back to its original state. It still has our message in it – we call this “tattooing”. The file component does not keep track of objects it used to manage and undo the operations if it no longer manages them.

# Example 5-1

- The file component can also use external templates to merge with resources to manage managed files
- Copy sshd_config.tmpl to /root/sshd_config.tmpl
- Look at the template, diff it with /etc/ssh/sshd_config
  - The extra lines control the file component
  - The port number is only managed if the resource is set



Attempting to store larger file directly in the profile quickly becomes inefficient. To avoid this, we can supply a template of the file and instruct the file component to use this on the client. In this case resource referencing takes place on the client rather than in the compiler on the server.

To examine the differences between the original and template files do

```
diff -u /etc/ssh/sshd_config /root/sshd_config.tmpl
```

A comment has been added warning that this file is being managed by LCFG and shouldn't be edited. Some template control code has also been added

```
<%if: <%v_port%>%><%\%>
# Listening port configured by LCFG
Port <%v_port%>
<%end:%><%\%>
```

Only if the LCFG profile has a value for the "file.v_port" resource set will the comment and the Port directive be added to the file.

See page 84 onwards in the guide for a description of the LCFG template processor syntax.

You would normally distribute the template file as part of a package or some other method. In this case we copy it into place by hand to emphasise that the file component doesn't do this for you.

# Example 5-2

- Copy example5 to ./localhost
- Look at the source profile
- Look at /etc/ssh/sshd_config
  - Our LCFG comment is near the top
  - No managed port setting though
- We didn't set the resource in the source profile!
- Try ssh'ing to localhost to check sshd is working



```
!file.files          mADD(example)
file.file_example    /etc/ssh/sshd_config
file.type_example    template
file.tmpl_example    /root/sshd_config.tmpl
file.owner_example   root
file.group_example   root
file.mode_example    0600
!file.variables      mADD(port)
```

This time the "type" resource is set to "template" and the "tmpl" resource points to the location **on the client** of the template file.

We remember that sshd is sensitive to the permissions on its config file so set them correctly.

Finally, we add a our "port" tag to the file component's list of variables.

Check you can run `ssh localhost` and log on. We get our message of the day issued to us!

# Example 6-1

- Copy example6 to ./localhost
- Look at the source profile
- It tells sshd to listen on port 222 rather than the default 22
- Check /etc/ssh/sshd_config for the port setting
- Try ssh -p 222 localhost
- Why doesn't it work?



The only relevant change is the addition of

```
file.v_port        222
```

which sets our "port" variable, hopefully resulting in sshd listening on port 222 rather than the default 22.

The `/etc/ssh/sshd_config` file should now have the following lines

```
# Listening port configured by LCFG
Port 222
```

The "-p" option to ssh specifies the port that the remote sshd server is listening on.

We get no response because all that LCFG has done is change the contents of a configuration file. The file component has no concept of running processes so is completely unable to inform sshd of the configuration change.

# Example 6-2

- Run /etc/init.d/sshd reload
- Now try ssh -p 222 localhost
- Now try ssh localhost
- The file component only goes so far ...