RED HAT :: CHICAGO :: 2009
SUMMIT

RPM-ifying Third Party Software

Paul Waterman
Professional, IT Systems – Motorola, Inc.
September 3, 2009

presented by

Agenda

Introductory Items
Housekeeping items like this agenda, who I am, expectations, etc.

The Scenario
Non-RPM installs, why companies do this, and why RPM's are better.

The Basic Process
A quick overview of how to turn third party software into an RPM.

The Process in Detail
A detailed dive into the process of turning third party software into an RPM.

Advanced Techniques
Virtualization, LVM snapshots, separate configuration RPM's, and triggers

Who is Paul Waterman?

Motorolan

I've worked for Motorola, Inc. for 13 years, and am currently the Linux Technical Lead for Midrange Unix Operations in Broadband Mobility Solutions.

Computer Geek

My first computer was a TRS-80 Model I with 16k of RAM in 1980.

Unix Guy

I was first introduced to Unix (ULTRIX) on a DEC MicroVAX II in 1987.

Fun Fact:
Ever played Nethack?
I was the principle author of the WCST Nethack Spoilers.

Linux Fan

I started using Linux in 1997 to run Magic: The Gathering tournaments.

Red Hat Admin

I began using Red Hat personally with Red Hat 5.2, at Motorola with Red Hat 6.2, was certified RHCE in 2004, and certified RHCA in 2009.

Expectations

To get the most out of this presentation, you should ...

… have system administration skills.
(RHCE or equivalent skills will best help you grasp the material.)

… have some experience with third party applications.
(I use ClearCase for examples, but the material should be broadly applicable.)

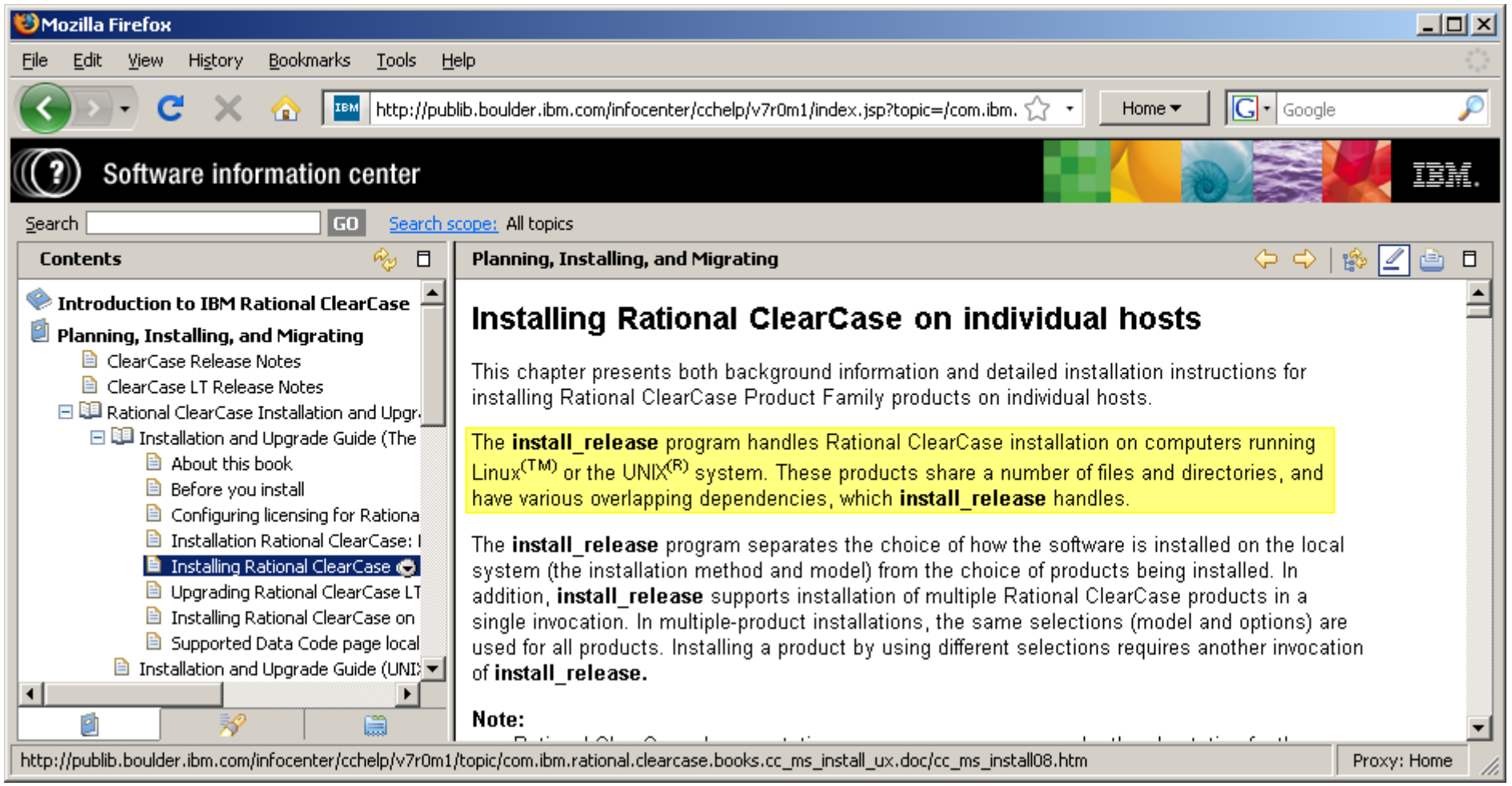
… be familiar with creating RPM's (spec files, etc.).
(This is not intended to be a general training session on creating RPM's.)

If you have questions, don't wait – ask them!

If the question will take too long to answer or take us too far off topic, we'll move on and you can talk to me afterward.

The Scenario

Do install instructions like this look familiar?



Why, why, why?

Why do companies use custom install programs instead of using the packaging mechanism used by the OS?

It may simplify their installation instructions by allowing the same instructions to be used on multiple platforms.

It may simplify their development by allowing them to re-use installation code on multiple platforms and by eliminating the need for familiarity with multiple packaging formats.

It allows them to do things that packaging mechanisms discourage or don't allow, such as forcing you to
read and acknowledge an EULA.

Did you know?
Some companies package their software as RPMs, but then embed the RPM in an install script!

Why RPM's?

So why not just use their install program?
Why are RPM's better?

You can install with one command, the same way you install everything else:

```
rpm -i some-package-1.0-1.x86_64.rpm
yum install some-package
```

You can easily see what is installed on the system:

```
rpm -q some-package
rpm -qa
```

You can see what package put a specific file on the system:

```
rpm -qf /path/to/some/file
```

You can see if package contents have been changed or removed:

```
rpm -V some-package
rpm -V
```

You can ensure that dependencies are properly handled – even *after* your third party software is installed.

The Process in a Nutshell

So how *do* you take a third party software installation and turn it into an RPM?

Here are the basic steps:

Set up a static, known system.

Install the third party software.

Figure out what was installed.

Create a spec file to package up what was installed.

Build the RPM.

1. Set up a static, known system

STEP 1

To start the process, you need a static, known system. What does that mean?

Static:

The system isn't changing – file system elements aren't being added, deleted, or modified (at least to the extent possible).

Known:

The system is in a known state – you know what's already on the system so you can tell if something has changed.

Static System

STEP 1

Make sure that as little as possible is being changed on the system.

Turn off all unnecessary services.
Really, truly, seriously, if a service is not absolutely necessary for what you're doing, turn it off.

Make sure nothing's connecting in from elsewhere and changing the system.

Static System – Services to look out for...

STEP 1

You would probably never normally turn these services off, but they regularly make changes to your system:

anacron / crond

Cron jobs may update indexes, pre-link binaries, rotate logs, etc.

```
# chkconfig off anacron
# chkconfig off crond
# service anacron stop
# service crond stop
```

syslog

System logging changes files in /var/log potentially multiple times per second.

```
# chkconfig off syslog
# service syslog stop
```

Static System - Example

STEP 1

My RHEL 4.8 static system:

Tip:
Some of these probably aren't necessary for me (e.g., readahead), and some may not be necessary for you (e.g., autofs).

```
# chkconfig --list | grep -w on | sort -b
acpid           0:off   1:off   2:off   3:on    4:on    5:on    6:off
autofs          0:off   1:off   2:off   3:on    4:on    5:on    6:off
iiim            0:off   1:off   2:on    3:on    4:on    5:on    6:off
irqbalance      0:off   1:off   2:off   3:on    4:on    5:on    6:off
lvm2-monitor    0:off   1:on    2:on    3:on    4:on    5:on    6:off
messagebus      0:off   1:off   2:off   3:on    4:on    5:on    6:off
microcode_ctl   0:off   1:off   2:on    3:on    4:on    5:on    6:off
network         0:off   1:off   2:on    3:on    4:on    5:on    6:off
nfslock         0:off   1:off   2:off   3:on    4:on    5:on    6:off
portmap         0:off   1:off   2:off   3:on    4:on    5:on    6:off
rawdevices      0:off   1:off   2:off   3:on    4:on    5:on    6:off
readahead       0:off   1:off   2:off   3:off   4:off   5:on    6:off
readahead_early 0:off   1:off   2:off   3:off   4:off   5:on    6:off
sshd            0:off   1:off   2:on    3:on    4:on    5:on    6:off
xfs             0:off   1:off   2:on    3:on    4:on    5:on    6:off
ypbind          0:off   1:off   2:off   3:on    4:on    5:on    6:off
```

Static System – Things to think about...

STEP 1

Make sure people aren't logging into the system to try to do work.

- Consider disabling logins.

If you have automated jobs that remotely connect to your systems (e.g., backups, monitoring, automated password updates, vulnerability scanning, etc.), make sure that your static system is not modified by those jobs.

- You may need to add the system to an exception list or otherwise disable the automated jobs.

Known System

STEP 1

Before you install the third party software on your system, you need to know everything that's on your system.

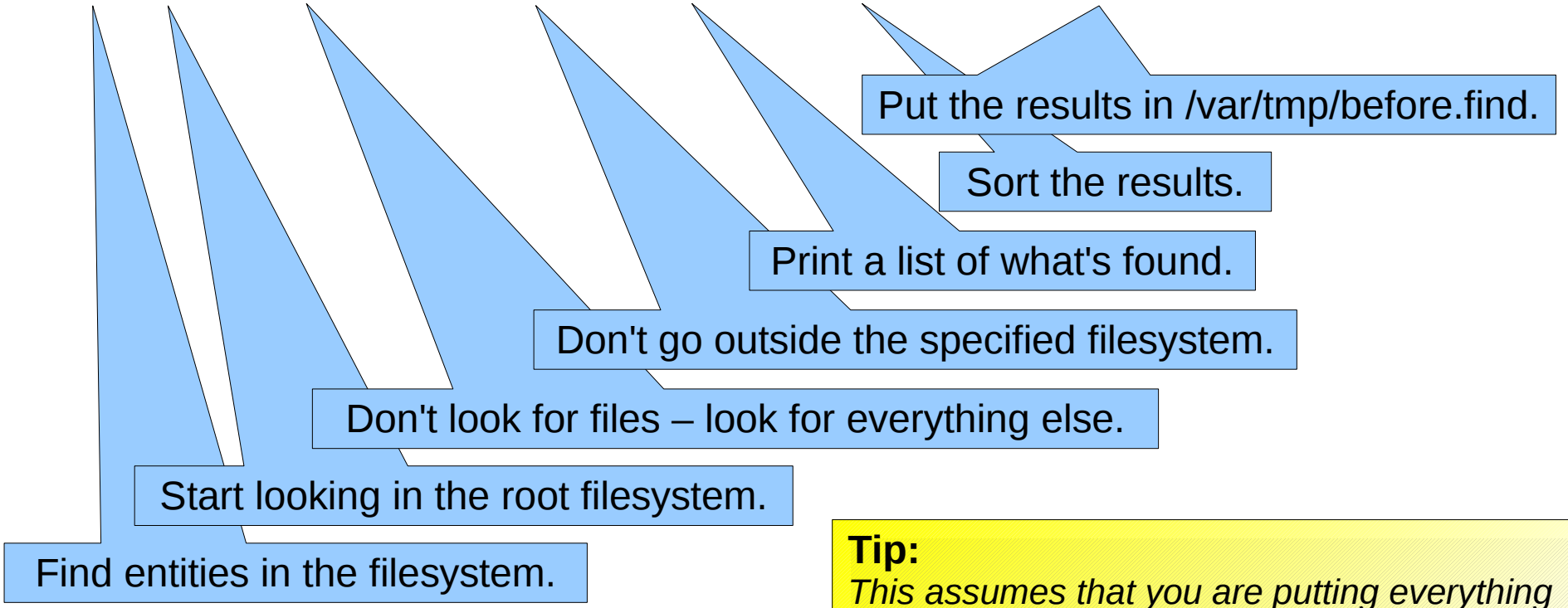
You need to know this so you can figure out what's changed after you install the third party software.

Known System – How To

STEP 1

First, get a list of all non-files on the system:

```
find / -not -type f -xdev -print | sort > /var/tmp/before.find
```



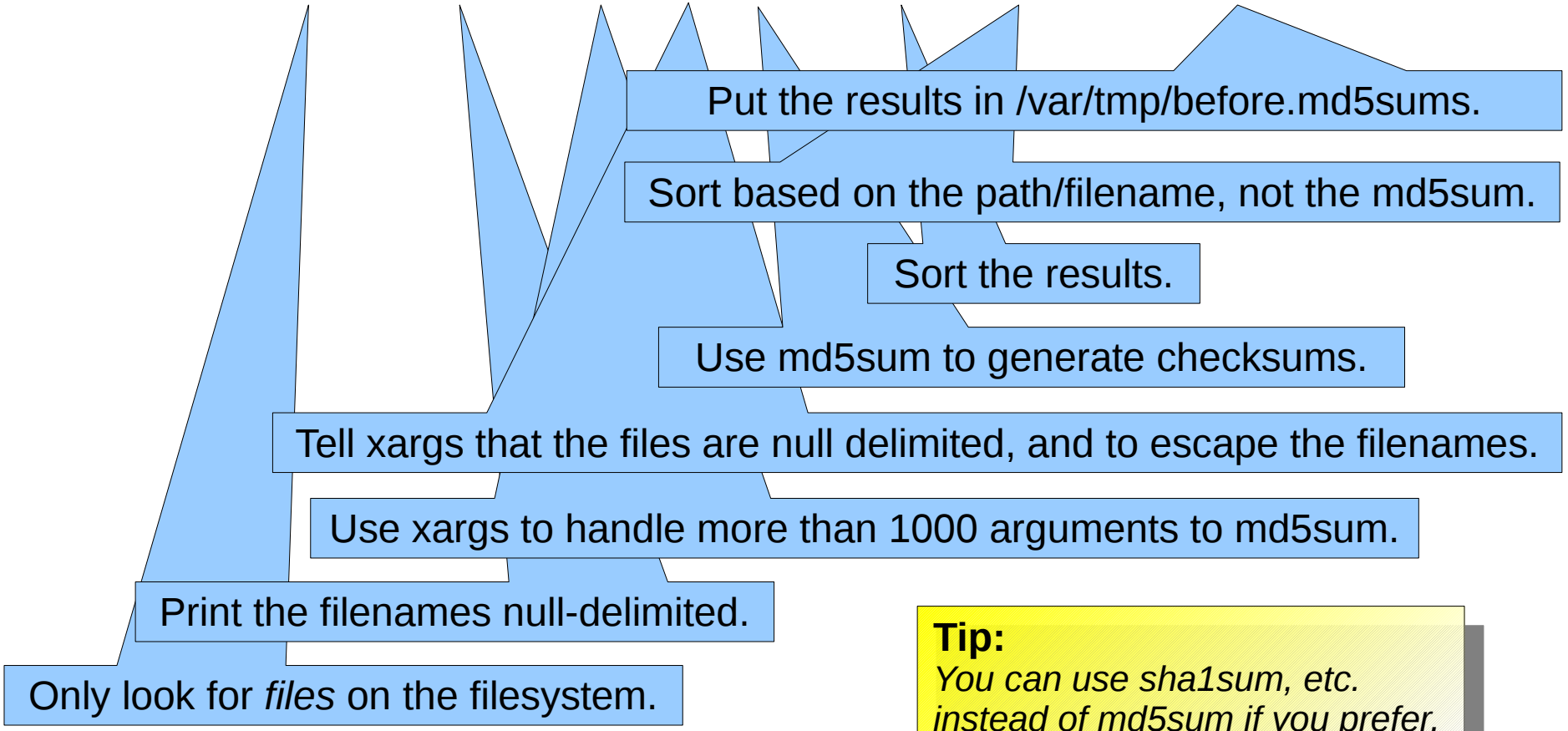
Tip:
This assumes that you are putting everything on the system in a single root filesystem. If you have split out /usr, /var, or other filesystems, you should also scan those.

Known System – How To

STEP 1

Next, generate a checksum of all files on the system:

```
find / -xdev -type f -print0 | xargs -0 md5sum | sort --key=2 > /var/tmp/before.md5sums
```



Tip:
You can use sha1sum, etc. instead of md5sum if you prefer.

2. Install the third party software.

STEP 2

Simply follow the installation instructions for the third party software. For example:

```
[root@hostname install]# ./install_release

Copyright IBM Rational Software. All Rights Reserved. This software
contains proprietary and confidential information of IBM Rational Software
and its suppliers.  Use, disclosure or reproduction is prohibited
without the prior express written consent of IBM Rational Software.
Subject to RESTRICTED RIGHTS for US Government users.
[. . .]
*****************************************************************
>> Begin component final customizations
*****************************************************************

FINAL: hostname        Errors:0        Warnings:0

         Log file for this session: /var/adm/rational/clearcase/log/Rationa
l_install.090809.09:43

[root@hostname install]#
```

3. Figure out what was installed.

STEP 3

Remember the commands you ran before to capture what was on the system? Run them again, putting the results into different files, and compare the results.

```
find / -not -type f -xdev -print | sort > /var/tmp/after.find
find / -xdev -type f -print0 | xargs -0 md5sum | sort --key=2
> /var/tmp/after.md5sums

diff /var/tmp/before.find /var/tmp/after.find
diff /var/tmp/before.md5sums /var/tmp/after.md5sums
```

Look for items that have been added or removed when comparing the find results.

Look for items that have been added, removed, or changed when comparing the checksum results.

Example Comparison

STEP 3

The following shows new directories added during an install of ClearCase:

```
[root@hostname /]# diff /var/tmp/before.find /var/tmp/after.find | more
24a25,34
> /cc
> /cc/vob
> /cc/vob/artp
> /cc/vob/cap_vob
> /cc/vob/masca_pdgcom2
> /cc/vob/mielb_agents
> /cc/vob/mssc_agentcom
> /cc/vob/nextsim_vob
> /cc/vob/pdgcom
> /cc/vob/sdu
1263a1274
> /etc/rc.d/rc0.d/K08clearcase
1339a1351
> /etc/rc.d/rc1.d/K08clearcase
1414a1427
> /etc/rc.d/rc2.d/K08clearcase
1548a1562
> /etc/rc.d/rc3.d/S77clearcase
--More--
```

Example Comparison

STEP 3

The following shows added and changed files during an install of ClearCase:

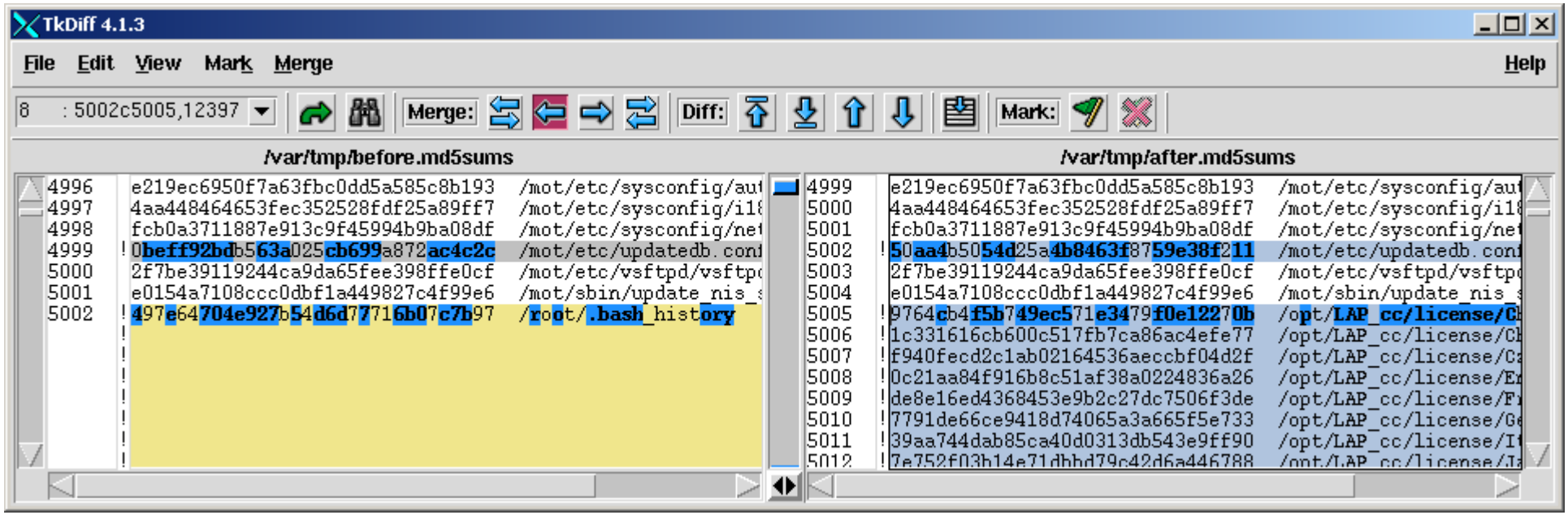
```
[root@hostname /]# diff /var/tmp/before.md5sums /var/tmp/after.md5sums | more
210c210
< 50c522a84740ac27f02d6d10ce0f5631  /etc/adjtime
---
> 3980633e477c5f6ef28f86f197cd9ed1  /etc/adjtime
342a343
> 4e548ca25458d6f4cb3dc2e8aa180db8  /etc/exports.mvfs
1787c1788
< f418638d44ea91cbb90ccce7ac882ccb  /etc/mtab
---
> 3c32bf46f2f641f8386d430fd9f5af63  /etc/mtab
2020a2022
> 6ec924d4e298f127bef972d8308edad3  /etc/rc.d/init.d/clearcase
4469a4472
>
8cadb286841e62094787aa5269a314cd  /lib/modules/2.6.9-89.ELsmp/kernel/fs/mvfs/mvfs.ko
4764c4767
< f5d9b1f8f67bf63a75b50bf546886b70  /lib/modules/2.6.9-89.ELsmp/modules.dep
---
> 1f48253f02de0d3b5bef18d2c6249e11  /lib/modules/2.6.9-89.ELsmp/modules.dep
--More--
```

Use a Visual Diff Tool

STEP 3

You may want to use a visual diff tool when comparing these lists – it makes it a lot easier.

I find `tkdiff` (not part of RHEL) quite useful.



RED HAT :: CHICAGO :: 2009 SUMMIT

New files, directories, etc.

STEP 3

Look for items that have been added to the filesystem.

If you're lucky, most of these will be clustered in a few directories that is “owned” by your software. In that case, you can just include the top level directory and have `rpmbuild` recursively include the rest when it builds the RPM.

For example, with ClearCase 7, the following directories contain almost everything:

```
/opt/LAP_cc
/opt/rational
/var/adm/rational
```

New items to ignore

STEP 3

Some new items should be ignored:

- Ignore anything that is created by the OS during normal operations.
  Some common examples:
  `/var/lock/subsys/*`
  `/var/run/*`
- Ignore anything in temporary directories.
- Ignore anything that will be created on the fly by the application. For example, in my ClearCase install, I know that the following are mount points that are automatically created by ClearCase:
  `/cc/*`
  `/usr/vob/*`
  `/vob/*`

Special handling required

STEP 3

Some new items may require special handling:

There may be new items on the system that should end up on the system, but will need to be handled in scriptlets within your RPM.

For example, in our ClearCase install, the following symbolic links are created on the system:

`/etc/rc.d/rc[01246].d/K08clearcase`
`/etc/rc.d/rc[35].d/S77clearcase`

Instead of including these symbolic links in the RPM, you should create them via `chkconfig` in the `%post` scriptlet.

Changed files, directories, etc.

STEP 3

Look for items that have changed on the filesystem.

Ignore changed items using the same rules as for new items.

Usually you will find that the majority of changed items are items that were changed by the OS as part of its normal operation, and can be ignored.

If changes made by the install need to be included in the RPM, you will need to make those changes in your RPM scriptlets.

Changed files, directories, etc.

STEP 3

For example, ClearCase updates the following file:
`/lib/modules/<kernel>/modules.dep`

Simply including this file in your RPM would be dangerous, as you might overwrite local changes to the `modules.dep` file on some systems.

The correct solution is to include lines in your RPM scriptlets to update the `modules.dep` file by running the `depmod` command.

4. Create a spec file...

STEP 4

Now you need to create a spec file. You'll use this spec file to build the RPM.

If you're not familiar with how RPM spec files and `rpmbuild` work, or would like to become more familiar, here are a couple of useful resources:

Maximum RPM (old and slightly out-of-date, but good):
http://www.redhat.com/docs/books/max-rpm/

Fedora Project RPM Guide:
http://docs.fedoraproject.org/drafts/rpm-guide-en/

The Preamble

STEP 4

Start with the preamble. For example:

```
#----------------------------------------------------------------------
# This spec file is Copyright 2009, Motorola, Inc.
#----------------------------------------------------------------------

Summary: IBM Rational ClearCase
Name: clearcase
Version: 7.0.1.4
Release: 1
Copyright: Copyright IBM Rational Software
Group: Motorola/ClearCase
Vendor: IBM Rational Software
Packager: Paul Waterman <my-email@motorola.com>
AutoReqProv: no
Requires: compat-db, coreutils, kernel-smp = 2.6.9-89.EL

%description
This package contains ClearCase 7.0.1 with patch 7.0.1.4.
```

Tip:
You may want to consult with your company's legal team regarding appropriate copyright notices and any other legal niceties with regard to repackaging commercial software for internal distribution.

Requirements

STEP 4

By default, `rpmbuild` will automatically detect what your package will require and add those to your RPM.

- It may miss some critical dependencies.
  - For instance, `rpmbuild` misses the fact that ClearCase installs a kernel module for the current kernel and thus won't work with other kernels.
- Commercial packages may include cruft that cause rpmbuild to incorrectly detect dependencies.
  - For instance, `rpmbuild` incorrectly thinks ClearCase needs `/bin/perl` and several other binaries referenced by unused scripts installed by ClearCase.
- You can eliminate this problem by turning off automatic requirements provisioning and specifying them manually:

```
AutoReqProv: no
Requires: compat-db, coreutils, kernel-smp = 2.6.9-89.EL
```

%prep Section

STEP 4

Normally, the %prep section is used to prepare the source code of an application for building – the source package is unpackaged, patches are applied, etc.

You should use the %prep section to make sure that your application is properly installed and ready to be packaged.

If your application normally runs a service or daemon on the system, it's a good idea to shut it down here in order to make sure that files aren't being changed by your application while they're being packaged.

%prep Section Example

STEP 4

Here's an example %prep section from my ClearCase spec file:

```
%prep
# Check to make sure ClearCase is installed
if [ ! -e /opt/rational ] ; then
  echo "ERROR: ClearCase is not installed."
  exit 1
fi

# If ClearCase is running, stop it
if [ -e /var/run/clearcase.pid ] ; then
  service clearcase stop
fi
```

Sections You Don't Need

STEP 4

There are a number of sections included in a typical spec file that you probably don't need:

`%build` – You're not building the application, so this section can be left out.

`%install` – Leave this section out, since you're doing the install on your own rather than via `rpmbuild`.

`%clean` – You're not building or installing anything via `rpmbuild`, so there's nothing to clean up.

Tip:
In certain circumstances (especially with advanced use cases) you may *want to use these sections to make sure that everything is in order for your package.*

Scriptlets

STEP 4

Create scriptlets as necessary to ensure that the RPM installs everything properly.

This is where you should handle items that you identified in Step 3 as needing special handling, such as files that were modified by the application install.

If your application involves a service, this is also where you should ensure that it's properly setting up the service.

Don't forget to write your scriptlets so that they handle uninstalls and upgrades as gracefully as they handle installs!

%pre Scriptlet Example

STEP 4

Here's an example %pre scriptlet from my ClearCase spec file:

```
%pre
# If ClearCase is running, stop it
if [ -e /var/run/clearcase.pid ] ; then
  service clearcase stop
fi
```

We do this to make sure that when upgrading ClearCase it will gracefully shut down before the upgrade.

%post Scriptlet Example

STEP 4

Here's an example %post scriptlet from my ClearCase spec file:

```
%post
# Update the module dependency list
depmod 2.6.9-89.ELsmp

# Add the ClearCase service
chkconfig --add clearcase

# Turn on the ClearCase service
chkconfig clearcase on

# If ClearCase isn't running, start it
if [ ! -e /var/run/clearcase.pid ] ; then
  service clearcase start
fi
```

%preun Scriptlet Example

STEP 4

Here's an example %preun scriptlet from my ClearCase spec file:

```
%preun
# If we're uninstalling the last copy of ClearCase...
if [ $1 -eq 0 ] ; then

  # If ClearCase is running, stop it
  if [ -e /var/run/clearcase.pid ] ; then
    service clearcase stop
  fi

  # Remove the ClearCase service
  chkconfig --del clearcase
fi
```

This is important, because when you upgrade a package, the new version gets installed before the old version is uninstalled.

%files Section

STEP 4

The %files section contains a list of what you want to go into your RPM. For example:

```
%files
# Directories to be recursively traversed
/opt/LAP_cc
/opt/rational
/var/adm/rational
# Directories not to be traversed
%dir /view
# Files
/etc/exports.mvfs
/etc/rc.d/init.d/clearcase
/lib/modules/2.6.9-89.ELsmp/kernel/fs/mvfs/mvfs.ko
# Symbolic links
/sbin/mount.mvfs
/usr/atria
/var/adm/atria
```

Build the RPM

STEP 5

Once you've done all of that, the actual RPM creation is relatively anticlimactic. Simply create the RPM using the `rpmbuild` command, specifying your spec file.
For example:

```
[root@hostname SPECS]# rpmbuild -bb clearcase-7.0.1.4-1.spec
Executing(%prep): /bin/sh -e /var/tmp/rpm-tmp.25962
+ umask 022
+ cd /usr/src/redhat/BUILD
+ LANG=C
+ export LANG
+ unset DISPLAY
+ '[' '!' -e /opt/rational ']'
+ '[' -e /var/run/clearcase.pid ']'
+ exit 0
Processing files: clearcase-7.0.1.4-1
Checking for unpackaged file(s): /usr/lib/rpm/check-files %{buildroot}
Wrote: /usr/src/redhat/RPMS/i386/clearcase-7.0.1.4-1.i386.rpm
```

Problem 1 – Strict Dependencies

The RPM that we've just created has very strict dependencies: It will only work on a system running one specific kernel.

ClearCase allows us to rebuild the module, so we can fix this by building the module in the `%post` scriptlet.

Problem 1 – Strict Dependencies

Example `%post` scriptlet update:

```
for k in `rpm -q --queryformat="%{VERSION}-%{RELEASE}\n" kernel`
do
  cd /var/adm/rational/clearcase/mvfs/mvfs_src
  echo "/lib/modules/${k}/build" | make clean
  echo "/lib/modules/${k}/build" | make all
  mkdir -p /lib/modules/${k}/kernel/fs/mvfs
  install --backup --suffix=.save mvfs.ko /lib/modules/$k}/kernel/fs/mvfs/mvfs.ko
  depmod ${k}
done

for k in `rpm -q --queryformat="%{VERSION}-%{RELEASE}\n" kernel-smp`
do
  cd /var/adm/rational/clearcase/mvfs/mvfs_src
  echo "/lib/modules/${k}smp/build" | make clean
  echo "/lib/modules/${k}smp/build" | make all
  mkdir -p /lib/modules/${k}smp/kernel/fs/mvfs
  install --backup --suffix=.save mvfs.ko /lib/modules/${k}smp/kernel/fs/mvfs/mvfs.ko
  depmod ${k}smp
done
```

Problem 1 – Strict Dependencies

We also need to make a few other adjustments...

Update the requirements list:

```
Requires: compat-db, coreutils, gcc, make
```

We run depmod for each kernel, so we can remove the original depmod line from the `%post` section:

```
# Update the module dependency list
depmod 2.6.9-89.ELsmp
```

We can remove the kernel module from `%files`:

```
/lib/modules/2.6.9-89.ELsmp/kernel/fs/mvfs/mvfs.ko
```

Problem 1 – Strict Dependencies

The RPM that we've created is also intolerant of upgrades on systems it's installed on. Using a new kernel on the system will break our application.

We can solve this by including triggers in our spec file.

These triggers will have code similar to the code that we added to our `%post` section to build the module for any new kernels that get installed.

In this case we can't rely on only the triggers to build the module, because triggers run after `%post` and we need the module there for our `%post` section.

Problem 1 – Strict Dependencies

Example triggers:

```
%triggerin – kernel
for k in `rpm -q --queryformat="%{VERSION}-%{RELEASE}\n" kernel`
do
  if [ ! -e /lib/modules/${k}/kernel/fs/mvfs/mvfs.ko ] ; then
    cd /var/adm/rational/clearcase/mvfs/mvfs_src
    echo "/lib/modules/${k}/build" | make clean
    echo "/lib/modules/${k}/build" | make all
    mkdir -p /lib/modules/${k}/kernel/fs/mvfs
    install --backup --suffix=.save mvfs.ko /lib/modules/${k}/kernel/fs/mvfs/mvfs.ko
    depmod ${k}
  fi
done

%triggerin – kernel-smp
for k in `rpm -q --queryformat="%{VERSION}-%{RELEASE}\n" kernel-smp`
do
  if [ ! -e /lib/modules/${k}smp/kernel/fs/mvfs/mvfs.ko ] ; then
    cd /var/adm/rational/clearcase/mvfs/mvfs_src
    echo "/lib/modules/${k}smp/build" | make clean
    echo "/lib/modules/${k}smp/build" | make all
    mkdir -p /lib/modules/${k}smp/kernel/fs/mvfs
    install --backup --suffix=.save mvfs.ko /lib/modules/${k}smp/kernel/fs/mvfs/mvfs.ko
    depmod ${k}smp
  fi
done
```

Problem 2 – Configuration

The RPM that we've just created contains the configuration and the application all in a single RPM.

If we create RPMs like this with multiple different configurations, we'll end up with several large RPMs and won't be able to quickly and easily change configurations on a system.

One solution is to split out the application and the configuration into two different RPMs.

Problem 2 - Configuration

In ClearCase, the configuration files are stored in the following directory:
`/var/adm/rational/clearcase/config`

If you recall, our `%files` section included the following:

```
%files
# Dirctories to be recursively traversed
/opt/LAP_cc
/opt/rational
/var/adm/rational
```

So we have a problem: If we want to leave the config directory out of the application RPM, we can't recursively include the top level directory in the `%files` section of our application RPM.

Problem 2 - Configuration

We can solve this by constructing a list of what goes into the `%files` section on the fly, pruning out what we don't want to include in the application RPM.

We can add this to the `%prep` section:

```
# Create a list of items in /var/adm/rational
find /var/adm/rational \
  -path /var/adm/rational/clearcase/config -prune \
  -o -not -type d -print \
  > /var/tmp/cc_filelist
find /var/adm/rational \
  -path /var/adm/rational/clearcase/config -prune \
  -o -type d -print \
  | awk '{print "%dir "$0}' \
  >> /var/tmp/cc_filelist
```

Problem 2 - Configuration

Then we update the `%files` section to include the list:

```
%files -f /var/tmp/cc_filelist
# Dirctories to be recursively traversed
/opt/LAP_cc
/opt/rational
```

Now we have something to clean up, so we should add a `%clean` section:

```
%clean
rm -f /var/tmp/cc_filelist
```

Since we know we're going to depend on a config package, we should include that requirement:

```
Requires: clearcase-config, compat-db, coreutils, gcc, make
```

Problem 2 – Configuration

Our configuration RPM is fairly simple:

```
#------------------------------------------------------------------------
# This spec file is Copyright 2009, Motorola, Inc.
#------------------------------------------------------------------------

Summary: IBM/Rational ClearCase Config for IL
Name: clearcase-config
Version: 7.0.1.4.il
Release: 3
Copyright: Copyright IBM Rational Software
Group: Motorola/ClearCase
Vendor: IBM Rational Software
Packager: Paul Waterman <my-email@motorola.com>
AutoReqProv: no

%description
This package contains the IL config for ClearCase 7.0.1

%files
# Dirctories to be recursively traversed
/var/adm/rational/clearcase/config
```

Advanced Topics – LVM Snapshots

What happens if you do this a lot and want to maintain a static, known system that you can use on demand?

What happens if you mess up your software install and thus mess up your static, known system?

You can avoid these problems using LVM snapshots:

- Create a static, known system.
- Create an LVM snapshot of the system.
- Work in the snapshot, preserving the original system.

Advanced Topics – LVM Snapshots

When creating a snapshot of your root filesystem and switching to use of that snapshot, you may end up seeing some quirky behavior if you try to do it with a live system.

I recommend using either rescue mode (iffy, because the LVM tools are not available unless you mount your root filesystem in at least read-only mode) or live media such as a Fedora Live CD.

Advanced Topics – LVM Snapshots

The basic process:

- Set up your static, known system.
- Boot in live mode.
- Create a snapshot of your root logical volume.
- Mount your boot partition and edit your grub.conf file to use the snapshot logical volume.
- Mount the snapshot logical volume and edit your fstab file to use the snapshot logical volume.
- Reboot and perform your install on the snapshot.

LVM Snapshots – Example Commands

Create a snapshot:

```
lvcreate --snapshot --name lv.rootsnap --size 10G /dev/vg00/lv.root
```

Update to grub.conf:

```
kernel /vmlinuz-2.6.9-89.ELsmp ro root=/dev/vg00/lv.rootsnap rhgb quiet
```

Update to fstab:

```
/dev/vg00/lv.rootsnap        /        ext3    defaults        1 1
```

Tip:
Your naming will probably vary from the examples above if you're using default volume group and logical volume naming schemes or have created your own naming schemes.

LVM Snapshots – Shortcuts

If you're using LVM snapshots, you can skip the steps of gathering before and after filesystem and checksum lists, because you can directly compare the original filesystem and the snapshot:

```
[root@hostname /]# mkdir /mnt/rootorig
[root@hostname /]# mkdir /mnt/rootsnap
[root@hostname /]# mount -o ro /dev/vg00/lv.root /mnt/rootorig
[root@hostname /]# mount --bind -o ro / /mnt/rootsnap
[root@hostname /]# diff -r --brief /mnt/rootorig /mnt/rootsnap | more
Only in /mnt/rootsnap: .autofsck
Only in /mnt/rootsnap/apps: internal
Only in /mnt/rootsnap/apps: public
Only in /mnt/rootsnap/apps: vendor
Only in /mnt/rootsnap: cc
--More--
```

Advanced Topics – Virtualization

Virtual machines are great resources for performing all of the activities in this presentation:

- You don't have to keep a whole physical computer sitting there just to do application installs and create packages.
- You can create a static, known system and leave it sitting dormant, then bring it up whenever you need it, copy it, etc.
- Many virtualization products have features such as VM snapshots that make all of this quite easy.

Advanced Topics – Virtualization

Q: Which virtualization product is best?
A: Whatever you're familiar with. They all work great!

Pick your poison:

- KVM
- Sun Virtual Box
- VMware (Infrastructure or Server)
- Xen
- Others

Conclusion

We've covered a lot of material, but you now should have the basic information that you need to take a third party software install and turn that into an RPM!

QUESTIONS?

TELL US WHAT YOU THINK:
REDHAT.COM/SUMMIT-SURVEY