SUSE LINUX Enterprise Server 10

www.novell.com

Advanced patching of SUSE Linux Enterprise Server 10 with ZENworks Linux Management 7.2

Version 0.2

Novell.

Disclaimer

Novell, Inc. makes no representations or warranties with respect to the contents or use of this document, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to revise this publication, and to make changes to its content, at any time.

Further, Novell, Inc. makes no representations or warranties with respect to any Novell software, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to make changes to any and all parts of Novell software, at any time, without any obligation to notify any person or entity of such changes.

Trademarks

Novell and NetWare are registered trademarks of Novell, Inc. in the United States and other countries.

The Novell Network Symbol is a trademark of Novell, Inc.

* All third-party trademarks are property of their respective owner.

Novell, Inc.
1800 South Novell Place
Provo, UT 84606
U. S. A.

Prepared By

Novell Presales

Document

Introduction to ZYPP patching

Traditionally, Linux systems are updated by regularly comparing the versions of installed software packages to the versions of a collection of packages residing in a remote catalog or bundle, and by replacing all outdated packages by the newer equivalents, in bulk. This approach has been applied for many years, because it is fairly simple to implement and understand, but it has the strong limitation that a system can only be in one of two predefined states : “up to date” or “not up to date”.

SUSE Linux Enterprise Server 10 overcomes this limitation. It does this by providing a mechanism where, by installing “ZYPP” patches for the respective software components, the system administrator can fully control the patch state of the system : in stead of being “up to date” or “not up to date”, a system can now have states like “original OS version plus patches foo-1234 and bar-5678”.

Besides fine granularity, ZYPP patches have other advantages. They come with a certain amount of meta data that normally cannot be conveyed with the traditional “package overlay” method of updating. A patch can e.g. contain the information that a system needs to be rebooted after installing the patch. Or, the patches can be categorized : SUSE Linux Enterprise Server 10 tags each patch as “security”, “recommended” or “optional”. A careful system administrator can decide to only automate the installation of security patches, and to install all other kinds manually, after verifying them in a test environment. This to obtain an optimal balance between security and stability.

The concept behind ZYPP patches if fairly simple : each ZYPP patch introduces at least one dependency requirement in the system. Such a requirement usually mandates that certain software must be at a specific version. ZYPP-aware software management tools will never break these requirements. Let's explain this with an example : by installing a ZYPP patch called “bind-2041”, the administrator effectively tells the SUSE Linux Enterprise Server 10 system that, if the bind software is installed, it can only be of version “9.3.2", release "17.7". If older bind software is installed on the system at the moment when the patch is installed, the ZYPP-aware tool that was used to install the patch will automatically trigger the now necessary update via one of the connected software repositories.

Introduction to ZENworks Linux Management (ZLM) 7.2

ZLM 7.2 is a powerful Linux systems management suite, that provides centralized control of up to hundreds of managed devices. One of the main capabilities is automated system patching, but by default ZLM 7.2 adopts the classic software package overlay approach, also for SUSE Linux Enterprise Server 10 devices. In this document we will thus explore how the native mechanisms of ZLM 7.2 can be used to orchestrate advanced ZYPP-based patching of SUSE Linux Enterprise Server 10 managed devices.

How ZLM 7.2 can orchestrate ZYPP patching (simple)

Before we can use ZLM 7.2 for anything related to ZYPP patching, it is necessary that we create one bundle group per ZYPP patch and insert that bundle into the ZLM 7.2 bundle repository. Each patch bundle contains the software packages in the exact version as specified by the patch. The easiest way to accomplish this is to set up a local mirror of the SUSE Linux Enterprise 10 update repository (a YUP tree – see below) and use a script to automate the parsing and loading of the patches in ZLM 7.2 .

After performing the above, the administrator has a graphical overview of installable patches and can link those patches to managed devices that need them. All this can be done via the graphical ZLM 7.2 ZCC web based console. The end result is thus an environment where the administrator can easily manage software and patch distribution to managed devices. If no patches that are applicable to the to be installed software are associated to the managed device, then the software as found on the original product installation medium will be installed. If a patch exists for the software and is associated to the device, then the installed software will be of the exact version as specified by the patch. If a patch is installed when the related software is already present on the system, then this results in an update of the software.

How ZLM 7.2 can orchestrate ZYPP patching (advanced)

In large environments, where absolute control and high granularity are necessary with regards to system patching, ZLM 7.2 can be of much more benefit than what is described above. In this paragraph and for the remainder of the document we will focus on an example of using ZLM 7.2 to orchestrate advanced patch management. The system that we will describe must allow for the following basic functionality :

- mirroring of patches, which are immediately inserted into one of five patch groups : Security Patches, Recommended Patches, Optional Patches, Security Related Kernel Patches and Non-security Related Kernel Patches
- patches are also immediately added into a test groups, per server category (architecture – host OS – application(s)) and per patch kind (security, optional, recommended, kernel-security, kernel-nonsecurity)
- for each server category, the system must warn the administrator when a certain associated patch test group contains patches that are not yet tested. Which patch groups trigger this alarm must be configurable
- the system must allow for a patch to be tested, and must allow the administrator to put the patch into an equivalent production group after testing
- patches that are in one of the production groups (per server category – per patch kind) raise a visual alarm if they are not yet installed on the linked managed devices
- the administrator can choose to install patches from a production patch group in a manually fashion, or can link that patch group to the corresponding devices with an installation schedule that will install the patches in that group automatically, according to that schedule

– the system must allow the main administrator to create other administrator accounts that can only perform a subset of tasks, related to specific objects. As an example consider an administrator account that can only see the test patches for a certain kind of server, and can only test those patches and then put them into the corresponding production patch group.

Setting up a YUP tree

Although not strictly necessary, it is advised to set up a local YUP tree. A local YUP tree is a copy of a SUSE Linux Enterprise Server 10 YUM update repository. The base update repository can be found on https://update.novell.com, but it makes little sense to use this base repository for your managed devices, since it wastes bandwidth , as each devices will try to contact this repository individually. Setting up a local copy is fairly easy , and can be done with the yup software package that can be found in the SUSE Linux Enterprise Server 10 Software Development Kit or on http://software.opensuse.org/download/home:/mge1512 .

Once you have installed “yup”, you need to configure it. The most important elements that you need, are the the “deviceid” and “secret” files of a machine that has been successfully registered in the Novell Customer Center (NCC). These files can be found in /etc/zmd, and contain the credentials that are needed to get access to the software updates. If you plan on using the same system to host your YUP tree and your ZLM 7.2 server, make sure to first register the system, make backups of the deviceid and secret files, and then install ZLM 7.2. The reason is that ZLM 7.2 will generate new deviceid and secret files which will overwrite your old ones, and you absolutely need the files that were used to register your system in the NCC to have yup correctly authenticate to the base SUSE Linux Enterprise 10 YUM repository.

Below an example configuration file for yup. The YUP_ID and YUP_PASS fields must be filled with the content found in the files /etc/zmd/deviceid and /etc/zmd/secret, respectively.

```
# cat /etc/sysconfig/yup
YUP_CURLPARAMS="--verbose --digest --remote-time --fail --insecure"
YUP_DEST_DIR="/var/yup"
YUP_ID="xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
YUP_PASS="xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
YUP_SERVER="update.novell.com"
YUP_DEBUGINFO="no"
YUP_SDK="no"
YUP_SRCRPM="no"
YOU_SERVER="you.novell.com"
```

When you run the “yup” command, the above configuration will create a mirror SUSE Linux Enterprise Server 10 patch repository in /var/yup.

Using the YUP tree to load ZYPP patch information into ZLM 7.2

ZLM 7.2 can be configured in such a way that it allows you to organize and manage your patches in a fashion as described in the above paragraph called “How ZLM 7.2 can orchestrate ZYPP patching (advanced)”. What is however needed, is a script which reads the patch information from the YUP tree and inserts corresponding patch bundles into ZLM 7.2 and into a set of preconfigured patch bundle groups. Such a script can be found in Appendix A of this document.

This script should ideally be run after every evocation of the yup tool, to make sure that a patch that was pulled in via yup is immediately inserted into ZLM 7.2 . Every time the script is run, it will verify and if necessary try to correct a predefined folder hierarchy in the ZLM 7.2 “Bundle” repository :

Bundles

Patches

Desktops

Servers

<server kind as defined by system administrator e.g. **App**>

SLES 10

<SLES 10 architecture as found in YUP tree e.g. **i586**>

Security Patches test group for App servers

Optional Patches test group for App servers

Recommended Patches test group for App servers

Security Related Kernel Patches test group servers

Non-security Related Kernel Patches test group for App servers

Security Patches production group for App servers

Optional Patches production group for App servers

Recommended Patches production group for App servers

Security Related Kernel Patches production group for App servers

Non-security Related Kernel Patches production group for App servers

SLES 10

<SLES 10 architecture as found in YUP tree e.g. **i586**>

Security Patches

Optional Patches

Recommended Patches

Security Related Kernel Patches

Non-security Related Kernel Patches

When inserted into ZLM 7.2, a ZYPP patch is first always inserted in two of the following patch bundle groups :

- Security Patches : if the patch is a security patch and not a kernel patch
- Recommended Patches : if the patch is a recommended patch and not a kernel patch
- Optional Patches : if the patch is an optional patch and not a kernel patch

– Security Related Kernel Patches : if the patch is a security patch and a kernel patch

– Non-security Related Kernel Patches : if the patch is a recommended or optional patch and a kernel patch

The above groups serve as reference patch bundle groups and as such always contain the latest patch bundles. Patches are never removed from these base bundles groups.

If the administrator has defined server types, the the patch is also inserted in the “test” group corresponding to corresponding the the patch category, for each server type. In this document we will adhere to the types “web” (for web servers), “app” (for application servers) and “dbs” (for database servers) :

– Security Patches test group for either web, app or dbs Servers

– Recommended Patches test group for either web, app or dbs Servers

– Optional Patches test group for either web, app or dbs Servers

– Security Related Kernel Patches test group for either web, app or dbs Servers

– Non-security Related Kernel Patches test group for either web, app or dbs Servers

To test the patches that reside in the test groups, virtual machines can be used : for each kind of server, a virtual machine is set up to test the patch and to, if needed, add pre- or post installation scripts to the patch that act on the patched service or fix anomalies caused by the patch. Virtual machines are ideal to perform patch testing on, since they can be freely instantiated and rolled back if multiple test rounds are necessary.

As said we want that the administrator gets some sort of visual notification if some or all of the test patch bundle groups that are applicable to a server kind (configurable) still contain untested patches. In ZLM 7.2, we implement this by assigning certain or all of these test patch bundle groups the applicable test server, with a schedule type which never actually automatically installs any bundle from the group. This causes the “Updates” icon for that test server to be lit with high priority as long as there are non-processed patches into one of it's patch test bundle groups, but only if the related base software is installed on the test server. It is thus necessary to keep the software installation state of the test server in sync with the one of the production servers.

As long as the “Updates” icon for a certain test server is lit, patch bundles from the linked test patch bundle groups should be examined and patches in them should be tested and eventually installed on the test server. When a patch bundle has been tested and is verified, is should be added to the production patch bundle group that corresponds to the test patch bundle group where it was originally found :

– Security Patches production group for either web, app or dbs Servers

– Recommended Patches production group for either web, app or dbs Servers

– Optional Patches production group for either web, app or dbs Servers

– Security Related Kernel Patches production group for either web, app or dbs Servers

– Non-security Related Kernel Patches production group for either web, app or dbs Servers

For each server kind, the actual production servers are assigned patch bundle groups in the following fashion. The reference patch bundle groups for Security Patches and Security Related Kernel Patches are assigned, but with a schedule which never actually automatically installs a bundle from those groups. These associations will

however play an important notification role : as long as there are security and kernel related security patch bundles to install on the device, this will make the “Updates” icon stay lit for the device. This ensures that all security related patches get eventually installed on the device (be it via different associations, discussed below).

The production patch bundles groups for Security Patches and Recommended Patches (and optionally Optional Patches) are associated with managed devices, based on the server kind, with a schedule that will install patch bundles from these groups fairly quickly (default in ZLM 7.2 is a refresh every two hours). The administrator can of course decide to push out a patch immediately when he adds the patch into the productions patch bundle group, and can choose an schedule for the patch bundle group which never actually automatically installs bundles from the group. In the latter case it is always up to the administrator to initiate the bundle installation process for each patch bundle individually.

The production patch bundle groups for Security and Non-security Related Kernel Patches are also associated with the devices, but again with a schedule which never actually installs a bundle. This is because kernel updates need to be followed by a system reboot – which usually requires some manual supervision. Kernel updates are also very critical and should be monitored while they are performed. It is thus up to the administrator to initiate that installation of such a bundle manually.

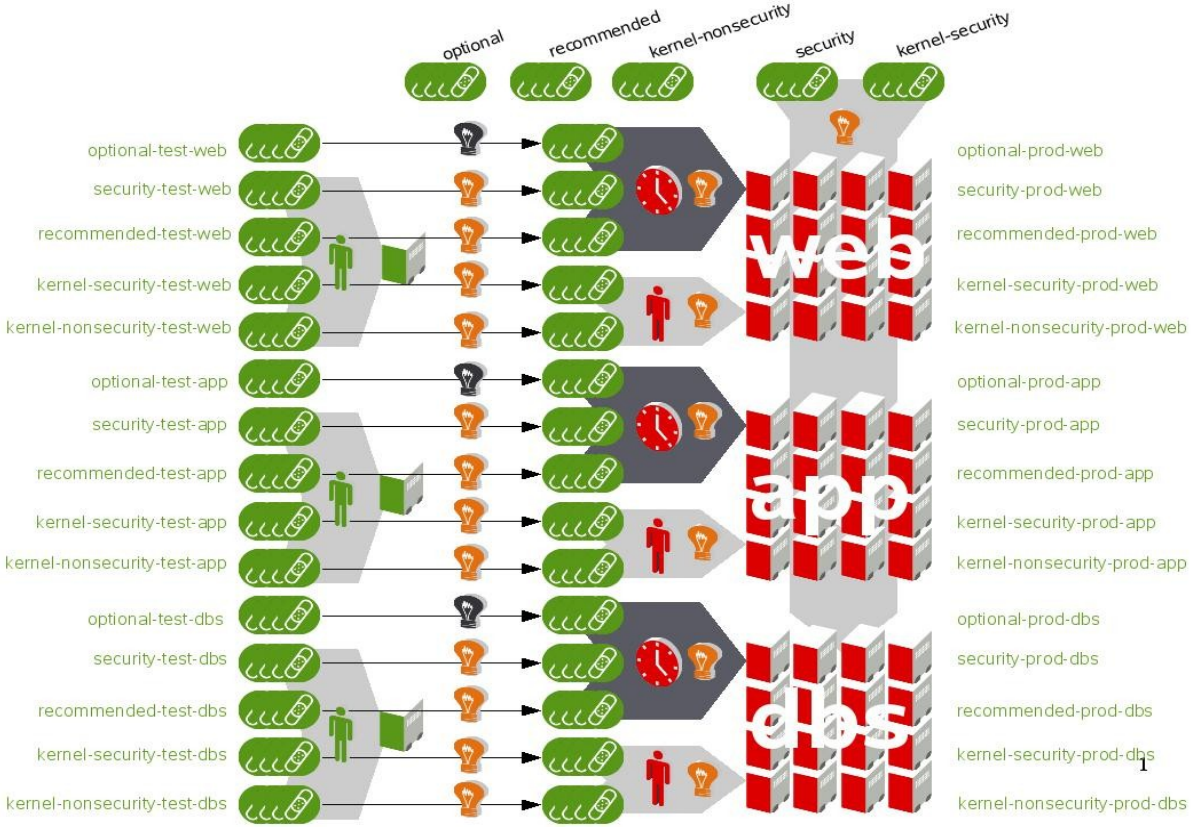

Fig 1.1 : graphical overview of the setup proposed in this document.

Additional required configuration

Don't forget to create a bundle containing all software packages that are found on the original SUSE Linux Enterprise Server 10 installation medium, to must make this collection of packages available to the managed devices via a catalog. Since this is standard ZLM 7.2 practice, we do not elaborate on this step in this document. Remember that the easiest way to make this catalog available to your managed devices, is to e.g. add it to the "Servers" folder, to which each SUSE Linux Enterprise Server 10 device will be automatically added when you register it.

Frequently asked questions

Q : Why are the patch states on my devices always "Needed", "Not needed" and "Not Applicable" ?

A : because we don't install the patch metadata on the managed device, we just install the corresponding software packages. As such the underlying libzypp library and related tools don't see the patch as "Installed", but rather "Not needed. "Not Applicable" means that the software to which the patch applies is not installed on the system.

Q : What happens when I associate an patch to a system which is already obsoleted by another patch ?

A : Nothing – unless you first explicitly deinstall the later patch with the option to remove the software packages from the managed device. Doing the latter is however absolutely discouraged, as it can bring you into a complex situation if on the managed device software resides which has dependencies on the patch related software. Generally speaking it is not a good idea to disassociate patches from devices with the option to deinstall the software packages. If you adhere to that best practice, then installing an obsoleted patch will have no effect, since the device will already have newer software packages then those related to the obsoleted patch. The software installed on the device will effectively always be of the version related to the latest patch that was ever installed on the system.

Q : How do I install new software on my managed devices ?

A : The same way you are used to when using ZLM 7.2 : you put the software in a bundle and associate that bundle to the device by using the command line tools or the ZLM 7.2 ZCC web based console. Remember that as part of the setup process, you had to create a bundle and a catalog for that bundle which makes all software that is part of the SUSE Linux Enterprise Server 10 product available to your devices. All you need to do is use the search engine to pinpoint a certain software from that catalog, create a bundle around it, and link that bundle directly to the managed device. The next time the device refreshes, it will pull the software and all it's dependencies in. ZLM 7.2 will automatically install the patched software if the related patch is associated to the managed device.

Appendix A : yup2zlm.sh version 0.2

```
#!/bin/bash

# Version 0.2 .

# This script serves merely as an example. It was deliberately kept as simple as possible and probably should be improved.

# use the '-s' option to skip the initial folder and group checking and creation. Never use this option for the first run.
# use the '-o' option to only perform the initial folder and group checking and creation.

# Base location of the YUP tree.
YUP2ZLM_YUP_TREE="/var/yup"
# How we must name the patch bundle groups
YUP2ZLM_PATCH_BUNDLE_BASE_NAME="Patches"
YUP2ZLM_SEC_PATCH_BUNDLE_EXT="Security"
YUP2ZLM_REC_PATCH_BUNDLE_EXT="Recommended"
YUP2ZLM_OPT_PATCH_BUNDLE_EXT="Optional"
YUP2ZLM_KER_SEC_PATCH_BUNDLE_EXT="Security-kernel"
YUP2ZLM_KER_NONSEC_PATCH_BUNDLE_EXT="Nonsecurity-kernel"
# Only supported products are SLES10 and/or SLED10 - this is YUP syntax.
YUP2ZLM_PRODUCTS="SLES10,SLED10"
# List of the architectures we are interested in for the products.
YUP2ZLM_ARCHS="i586"
# ZLM 7.2 folder where all SLES 10 related patches and patch groups should reside in.
YUP2ZLM_SLES10_PATCHES_FOLDER="/Patches/Servers"
# ZLM 7.2 folder where all SLED 10 related patches and patch groups should reside in.
YUP2ZLM_SLED10_PATCHES_FOLDER="/Patches/Desktops"
# Credentials to connect to ZLM 7.2 - you might want to give this file permissions 700.
YUP2ZLM_ZLMAN_CMD="/usr/bin/zlman -U administrator -P novell"
# Names of the server kinds for which we want to create test and production patch bundle groups
YUP2ZLM_SERVER_GROUPS="web,app,dbs"
# Names of the desktop kinds for which we want to create test and production patch bundle groups
YUP2ZLM_DESKTOP_GROUPS=""
# Environments in which you want patches to get loaded - comma or space separated.
YUP2ZLM_ENV_LOAD="test"
# Environments in which patches should not yet be loaded - comma or space separated.
YUP2ZLM_ENV_NOTLOAD="prod"
# What the transaction folder should be called
YUP2ZLM_TRANS_FOLDER="Committed"

if [ "$YUP2ZLM_SLES10_PATCHES_FOLDER" = "$YUP2ZLM_SLED10_PATCHES_FOLDER" ] ; then
 echo Server related patches folder and desktop related patches folder must differ.
 exit 1
fi

ENV_LOAD=`echo $YUP2ZLM_ENV_LOAD | tr ',' ' '`
ENV_NOTLOAD=`echo $YUP2ZLM_ENV_NOTLOAD | tr ',' ' '`
ENVIRONMENTS="$ENV_LOAD"" ""$ENV_NOTLOAD"

if [ "$1" = "-s" ] ; then
```

```
  :
else
  FORMEDPATH="/"
  for PATHELEMENT in `echo $YUP2ZLM_SLES10_PATCHES_FOLDER | tr '/' ' '` ; do
    $YUP2ZLM_ZLMAN_CMD bfc "$PATHELEMENT" "$FORMEDPATH"
    FORMEDPATH="${FORMEDPATH}/$PATHELEMENT"
  done
  FORMEDPATH="/"
  for PATHELEMENT in `echo $YUP2ZLM_SLED10_PATCHES_FOLDER | tr '/' ' '` ; do
    $YUP2ZLM_ZLMAN_CMD bfc "$PATHELEMENT" "$FORMEDPATH"
    FORMEDPATH="${FORMEDPATH}/$PATHELEMENT"
  done
fi
# --------------------------------------------------------------------------------------------------------------------
# ------------------------------ For every product in the YUP tree ... ---------------------------------------------
# --------------------------------------------------------------------------------------------------------------------
for PRODUCT_FOLDER in `ls ${YUP2ZLM_YUP_TREE}` ; do
  CURRENT_PAT_BUNDLE=""
  if echo "$YUP2ZLM_PRODUCTS" | grep "$PRODUCT_FOLDER" - 1>/dev/null 2>&1 ; then
    if [ "$PRODUCT_FOLDER" = "SLES10" ] ; then
      PRODUCT="sles-10"
      CURRENTFOLDER="$YUP2ZLM_SLES10_PATCHES_FOLDER"
      DEVICE_GROUPS="$YUP2ZLM_SERVER_GROUPS"
    elif [ "$PRODUCT_FOLDER" = "SLED10" ] ; then
      PRODUCT="sled-10"
      CURRENTFOLDER="$YUP2ZLM_SLED10_PATCHES_FOLDER"
      DEVICE_GROUPS="$YUP2ZLM_DESKTOP_GROUPS"
    fi
# --------------------------------------------------------------------------------------------------------------------
# ------------------------------ For every architecture for that product in the YUP tree ... -------------------------
# --------------------------------------------------------------------------------------------------------------------
    for ARCH in `ls ${YUP2ZLM_YUP_TREE}/${PRODUCT_FOLDER}`; do
      if echo "$YUP2ZLM_ARCHS" | grep "$ARCH" - 1>/dev/null 2>&1 ; then
        PRODARCH="${PRODUCT}-${ARCH}"
        cd ${YUP2ZLM_YUP_TREE}/${PRODUCT_FOLDER}/${ARCH}/repodata
        THIS_ARCH_OK="true"
        if [ "$1" = "-s" ] ; then
          :
        else
          $YUP2ZLM_ZLMAN_CMD bfc "$PRODUCT" "$CURRENTFOLDER"
          $YUP2ZLM_ZLMAN_CMD bfc "$ARCH" "$CURRENTFOLDER"/"$PRODUCT"
          $YUP2ZLM_ZLMAN_CMD bfc  "$YUP2ZLM_TRANS_FOLDER" "$CURRENTFOLDER"/"$PRODUCT"/"$ARCH"
          DEVICE_KIND=""
          for DEVICE_KIND in `echo $DEVICE_GROUPS | tr ',' ' '` ; do
            if [ -n "$DEVICE_KIND" ] ; then
              $YUP2ZLM_ZLMAN_CMD bfc "$DEVICE_KIND" "$CURRENTFOLDER"
              $YUP2ZLM_ZLMAN_CMD bfc "$PRODUCT" "$CURRENTFOLDER"/"$DEVICE_KIND"
              $YUP2ZLM_ZLMAN_CMD bfc  "$ARCH" "$CURRENTFOLDER"/"$DEVICE_KIND"/"$PRODUCT"
              for ENVIRONMENT in $ENVIRONMENTS ; do
                if [ -n "$ENVIRONMENT" ] ; then
                  $YUP2ZLM_ZLMAN_CMD bfc "$ENVIRONMENT" "$CURRENTFOLDER"/"$DEVICE_KIND"/"$PRODUCT"/"$ARCH"
```

```
        fi
      done
    fi
  done
  BUNDLE=""
  for BUNDLE in "$YUP2ZLM_SEC_PATCH_BUNDLE_EXT"_"$YUP2ZLM_PATCH_BUNDLE_BASE_NAME"_"$PRODARCH" \
                "$YUP2ZLM_REC_PATCH_BUNDLE_EXT"_"$YUP2ZLM_PATCH_BUNDLE_BASE_NAME"_"$PRODARCH" \
                "$YUP2ZLM_OPT_PATCH_BUNDLE_EXT"_"$YUP2ZLM_PATCH_BUNDLE_BASE_NAME"_"$PRODARCH" \
                "$YUP2ZLM_KER_SEC_PATCH_BUNDLE_EXT"_"$YUP2ZLM_PATCH_BUNDLE_BASE_NAME"_"$PRODARCH" \
                "$YUP2ZLM_KER_NONSEC_PATCH_BUNDLE_EXT"_"$YUP2ZLM_PATCH_BUNDLE_BASE_NAME"_"$PRODARCH" ; do
    if $YUP2ZLM_ZLMAN_CMD bgm "$CURRENTFOLDER"/"$PRODUCT"/"$ARCH"/"$BUNDLE" 1>/dev/null 2>&1; then
      :
    else
      if $YUP2ZLM_ZLMAN_CMD bgc "$BUNDLE" "$CURRENTFOLDER"/"$PRODUCT"/"$ARCH" ; then
        :
      else
        echo "$CURRENTFOLDER"/"$PRODUCT"/"$ARCH"/"$BUNDLE" could not be created - check folder path.
        THIS_ARCH_OK="false"
        break
      fi
    fi
  done
  if [ "$THIS_ARCH_OK" = "false" ] ; then
    echo Errors found - skipping architecture "$ARCH" for product "$PRODUCT".
    continue
  fi
    for ENVIRONMENT in $ENVIRONMENTS ; do
      if [ -n "$ENVIRONMENT" ] ; then
      for DEVICE_KIND in `echo $DEVICE_GROUPS | tr ',' ' '` ; do
        if [ -n "$DEVICE_KIND" ] ; then
          BUNDLE=""
          for BUNDLE in "$YUP2ZLM_SEC_PATCH_BUNDLE_EXT"_"$YUP2ZLM_PATCH_BUNDLE_BASE_NAME"_"$PRODARCH" \
                        "$YUP2ZLM_REC_PATCH_BUNDLE_EXT"_"$YUP2ZLM_PATCH_BUNDLE_BASE_NAME"_"$PRODARCH" \
                        "$YUP2ZLM_OPT_PATCH_BUNDLE_EXT"_"$YUP2ZLM_PATCH_BUNDLE_BASE_NAME"_"$PRODARCH" \
                        "$YUP2ZLM_KER_SEC_PATCH_BUNDLE_EXT"_"$YUP2ZLM_PATCH_BUNDLE_BASE_NAME"_"$PRODARCH" \
                        "$YUP2ZLM_KER_NONSEC_PATCH_BUNDLE_EXT"_"$YUP2ZLM_PATCH_BUNDLE_BASE_NAME"_"$PRODARCH" ; do
                                                                                  if     $YUP2ZLM_ZLMAN_CMD     bgm
"$CURRENTFOLDER"/"$DEVICE_KIND"/"$PRODUCT"/"$ARCH"/"$ENVIRONMENT"/"$DEVICE_KIND"_"$ENVIRONMENT"_"$BUNDLE"  1>/dev/null  2>&1;
then
            :
          else
                                                 if   $YUP2ZLM_ZLMAN_CMD   bgc   "$DEVICE_KIND"_"$ENVIRONMENT"_"$BUNDLE"
"$CURRENTFOLDER"/"$DEVICE_KIND"/"$PRODUCT"/"$ARCH"/"$ENVIRONMENT" ; then
            :
          else
                                                                                                                  echo
"$CURRENTFOLDER"/"$DEVICE_KIND"/"$PRODUCT"/"$ARCH"/"$ENVIRONMENT"/"$DEVICE_KIND"_"$ENVIRONMENT"_"$BUNDLE"   could   not   be
created - check folder path.
            THIS_ARCH_OK="false"
            break
          fi
        fi
      done
      if [ "$THIS_ARCH_OK" = "false" ] ; then
```

```
                  break
                fi
              fi
            done
          fi
          if [ "$THIS_ARCH_OK" = "false" ] ; then
            break
          fi
        done
      fi
      if [ "$THIS_ARCH_OK" = "false" ] ; then
        echo Errors found - skipping architecture "$ARCH" for product "$PRODUCT".
        continue
      fi
      if [ "$1" = "-o" ] ; then
        :
      else
        EXISTING_PATCHES=`$YUP2ZLM_ZLMAN_CMD bl "$CURRENTFOLDER"/"$PRODUCT"/"$ARCH"/"$YUP2ZLM_TRANS_FOLDER" | grep '.*-[0-9][0-9][0-9][0-9][ 0-9].*' | grep 'Software Bundle' | cut -d '|' -f 1 | tr -d ' ' | sort -n`
# ------------------------------------------------------------------------------------------------------------------------
# ------------------------------ End of preliminary work - we start adding patches now -------------------------------
# ------------------------------------------------------------------------------------------------------------------------
        for PATCHFILE in patch-*.xml ; do
          if [ -f "$PATCHFILE" ] ; then
            PATCHID=`grep -i patchid $PATCHFILE | cut -d '"' -f 2`
            PATCHNAME=${PATCHID%-*}
            PATCHNUMBER=${PATCHID##*-}
            KIND=""
            KIND=`grep '<category>' "$PATCHFILE" | cut -d '>' -f 2 | cut -d '<' -f 1`
            PATCHBUNDLE="${PATCHID}"
            if echo "$EXISTING_PATCHES" | grep "$PATCHBUNDLE" - 2>/dev/null 1>&2 ; then
              echo "Patch "$PATCHBUNDLE" was already processed correctly - skipping."
            else
              START=`grep -n '<description lang="en">' "$PATCHFILE" | cut -d ':' -f 1`
              GOODSTART=`expr $START + 1`
              START2=`tail +${GOODSTART} "$PATCHFILE" | grep -n  '</description>' | head  -1 | cut -d ':' -f 1`
              GOODEND=`expr $START + $START2 - 1`
              PATCHINFOID=`grep 'PATCHINFO' "$PATCHFILE" | cut -d '"' -f 2`
              DESCRIPTION=`sed -n "$START,${GOODEND}p" "$PATCHFILE" | cut -d '>' -f 2 | tr -d '\r\n' `
              DESCRIPTION_FILTERED=`echo $DESCRIPTION | tr -d '\'`
              TOTALDESCRIPTION="$PATCHINFOID"" - ""$DESCRIPTION_FILTERED"
                                    if  $YUP2ZLM_ZLMAN_CMD  bc  "$PATCHBUNDLE"  "$CURRENTFOLDER"/"$PRODUCT"/"$ARCH"  --description="$TOTALDESCRIPTION" ; then
                :
              else
                $YUP2ZLM_ZLMAN_CMD bd "$CURRENTFOLDER"/"$PRODUCT"/"$ARCH"/"$PATCHBUNDLE"
                                     if  $YUP2ZLM_ZLMAN_CMD  bc  "$PATCHBUNDLE"  "$CURRENTFOLDER"/"$PRODUCT"/"$ARCH"  --description="$TOTALDESCRIPTION" ; then
                  echo Replaced uncommitted bundle "$CURRENTFOLDER"/"$PRODUCT"/"$ARCH"/"$PATCHBUNDLE"
                else
                  echo Could not create bundle "$PATCHBUNDLE" in folder "$CURRENTFOLDER"/"$PRODUCT"/"$ARCH".
                  THIS_ARCH_OK="false"
                  break
```

```
  fi
fi
patchbundleok=true
ATOMLIST=""
for ATOM in `cat "$PATCHFILE" | grep atom | grep rpm | cut -d '"' -f 4,8,10 | tr '"' '-'` ; do
  for ATOMFILE in `ls ../rpm/*/$ATOM*.rpm` ; do
    if echo "$ATOMFILE" | grep -i 'patch.rpm$' -  ; then
      echo Skipping patch rpm $ATOMFILE
    else
      if ls $ATOMFILE ; then
        ATOMLIST="$ATOMLIST $ATOMFILE"
      else
        echo Could not locate rpm corresponding to atom "$ATOM".
        patchbundleok=false
        break
      fi
    fi
  done
  if [ "$patchbundleok" = "false" ] ; then
    break
  fi
done
if [ "$patchbundleok" = "false" ] ; then
  THIS_ARCH_OK="false"
  break
fi
if echo "$PATCHID" | grep 'kernel-' ; then
                                                 if   $YUP2ZLM_ZLMAN_CMD   bap   --force-nevra      --installtype=install
"$CURRENTFOLDER"/"$PRODUCT"/"$ARCH"/"$PATCHBUNDLE" $PRODARCH $ATOMLIST ; then
    echo Added rpms "$ATOMLIST" to bundle "$CURRENTFOLDER"/"$PRODUCT"/"$ARCH"/"$PATCHBUNDLE" .
  else
    echo Could not add rpm "$ATOM*.rpm" to bundle "$CURRENTFOLDER"/"$PRODUCT"/"$ARCH"/"$PATCHBUNDLE" .
    THIS_ARCH_OK="false"
    break
  fi
else
                                          if   $YUP2ZLM_ZLMAN_CMD   bap   --force-nevra     --freshen   --installtype=upgrade
"$CURRENTFOLDER"/"$PRODUCT"/"$ARCH"/"$PATCHBUNDLE" $PRODARCH $ATOMLIST ; then
    echo Added rpms "$ATOMLIST" to bundle "$CURRENTFOLDER"/"$PRODUCT"/"$ARCH"/"$PATCHBUNDLE" .
  else
    echo Could not add rpm "$ATOM*.rpm" to bundle "$CURRENTFOLDER"/"$PRODUCT"/"$ARCH"/"$PATCHBUNDLE" .
    THIS_ARCH_OK="false"
    break
  fi
fi
ADD_LIST=""
case "$KIND" in
security)
  if echo "$PATCHID" | grep 'kernel-' ; then
    echo Skipping kernel patch "$PATCHID" as security patch and adding it to kernel security patches.
    ADD_LIST="$ADD_LIST $YUP2ZLM_KER_SEC_PATCH_BUNDLE_EXT"
  else
```

```
                    ADD_LIST="$ADD_LIST $YUP2ZLM_SEC_PATCH_BUNDLE_EXT"
                  fi
                  ;;
                recommended)
                  if echo "$PATCHID" | grep 'kernel-' ; then
                    echo Skipping kernel patch "$PATCHID" as recommended patch and adding it to kernel non-security patches.
                    ADD_LIST="$ADD_LIST $YUP2ZLM_KER_NONSEC_PATCH_BUNDLE_EXT"
                  else
                    ADD_LIST="$ADD_LIST $YUP2ZLM_REC_PATCH_BUNDLE_EXT"
                  fi
                  ;;
                *)
                  if echo "$PATCHID" | grep 'kernel-' ; then
                    echo Skipping kernel patch "$PATCHID" as optional patch and adding it to kernel non-security patches.
                    ADD_LIST="$ADD_LIST $YUP2ZLM_KER_NONSEC_PATCH_BUNDLE_EXT"
                  else
                    ADD_LIST="$ADD_LIST $YUP2ZLM_OPT_PATCH_BUNDLE_EXT"
                  fi
                  ;;
                esac
                EXTENSION=""
                for EXTENSION in $ADD_LIST ; do
                                                                                    if     $YUP2ZLM_ZLMAN_CMD     bga
"$CURRENTFOLDER"/"$PRODUCT"/"$ARCH"/"$EXTENSION"_"$YUP2ZLM_PATCH_BUNDLE_BASE_NAME"_"$PRODARCH"
"$CURRENTFOLDER"/"$PRODUCT"/"$ARCH"/"${PATCHBUNDLE}" ; then
                    :
                  else
                     echo Could not add patch ${PATCHID} to bundle "$EXTENSION"_"$YUP2ZLM_PATCH_BUNDLE_BASE_NAME"_"$PRODARCH"
in folder "$CURRENTFOLDER"/"$PRODUCT"/"$ARCH".
                    THIS_ARCH_OK="false"
                    break
                  fi
                  for ENVIRONMENT in "$ENV_LOAD" ; do
                    if [ -n "$ENVIRONMENT" ] ; then
                      for DEVICE_KIND in `echo $DEVICE_GROUPS | tr ',' ' '` ; do
                        if [ -n "$DEVICE_KIND" ] ; then
                                                                                  if    $YUP2ZLM_ZLMAN_CMD    bga
"$CURRENTFOLDER"/"$DEVICE_KIND"/"$PRODUCT"/"$ARCH"/"$ENVIRONMENT"/"$DEVICE_KIND"_"$ENVIRONMENT"_"$EXTENSION"_"$YUP2ZLM_PATCH
_BUNDLE_BASE_NAME"_"$PRODARCH"  "$CURRENTFOLDER"/"${PATCHBUNDLE}" ; then
                          :
                        else
                                                                      echo  Could  not  add  patch  ${PATCHID}  to  bundle
"$DEVICE_KIND"_"$ENVIRONMENT"_"$EXTENSION"_"$YUP2ZLM_PATCH_BUNDLE_BASE_NAME"_"$PRODARCH"              in              folder
"$CURRENTFOLDER"/"$DEVICE_KIND"/"$PRODUCT"/"$ARCH"/"$ENVIRONMENT".
                          THIS_ARCH_OK="false"
                          break
                        fi
                      fi
                    done
                  fi
                  if [ "$THIS_ARCH_OK" = "false" ] ; then
                    break
                  fi
                done
                if [ "$THIS_ARCH_OK" = "false" ] ; then
```

```
                    break
                  fi
                done
                if [ "$THIS_ARCH_OK" = "false" ] ; then
                  break
                fi
                if $YUP2ZLM_ZLMAN_CMD bc "$PATCHBUNDLE" "$CURRENTFOLDER"/"$PRODUCT"/"$ARCH"/"$YUP2ZLM_TRANS_FOLDER" ; then
                :
                else
                  echo Could not add patch "$PATCHBUNDLE" to transaction folder.
                  THIS_ARCH_OK="false"
                  break
                fi
              fi
            fi
# ------------------------------------------------------------------------------------------------------------------
# ------------------------------ Patch was processed correctly - adding to transacion log folder ... ------------------
# ------------------------------------------------------------------------------------------------------------------
          done
        fi
        if [ "$THIS_ARCH_OK" = "false" ] ; then
          echo Errors found - skipping architecture "$ARCH" for product "$PRODUCT".
          continue
        fi
      else
        echo REFUSED ARCH : "$ARCH"
      fi
    done
  else
    echo REFUSED PRODUCT : "$PRODUCT_FOLDER"
  fi
done
```