bakonf usermanual

Iustin Pop

This document explains the concept and usage of bakonf 0.5.1, a system tool which backs up the configuration files on a system.

1. About this document

This is the usermanual for the bakonf project, version 0.5.1; homepage is at http://www.nongnu.org/bakonf/. You can also get new versions of this document there.

Revision: $Id: usermanual.sgml,v 1.10 2002/12/20 01:46:47 iusty Exp $

2. Introduction

Making backup is an important aspect of system administration. The techniques of backing up data are explained in any good document about system administration, and they won't be explained here again.

However, bakonf comes into play into a particular part of the backups: the backing up of the system's configuration.

The basic idea is that on a standard instalation of a Unix-like system you have a lot of data which can be very easily restored from the original media, thus there is no point in archiving it. For example, after a fresh install of a RedHat Linux 8.0, you have ~4.5GB of space used. However, only a *very* small part of this amount is holding important information, the other part being binaries, libraries and other kind of data which will never modify in normal usage. Only the configuration files are changing (of course, also the user data is changing, but we are talking about an empty system).

If we classify the files existing on a Unix system, we have:

configuration data

> These are the target of bakonf; they are usually small text files, partly coming from the system installation, maybe edited by the administrator, partly created by him. Size is (on the workstation I write this) ~15MB.

binaries, libraries, other system files

> These are mostly read-only; in a package based distribution, they came from the packages and are replaced when the package is upgraded. In classical systems, they come from the install archives. Size is (in our hypothetical rh8.0 full install) ~4.5GB.

user data

These are emails, web pages, documents, etc. - this is important data, and needs to be backed up regularly. They also don't come from the installation media, and are not touched by the system. Size is undetermined, but is guaranteed to be exactly the amount of free space on the system :).

system variable data

These are the files created and managed by the system, usually from the configuration files and other external variables. Examples: `/var/lib/logrotate.status`, `/var/lib/slocate/slocate.db`. These are not all critical files, some are needed to be included in a backup only for analysis purposes, others should not be included in backups (e.g. if you reinstall your system or restore from backup, some files will have for sure other contents, generated from the new installation).

From all these, only the configuration data and the user data are absolutely required to recreate the system. The binaries can come from the installation source. The system managed data will be recreated by the system. And since the difference in size between the configuration and user data is so great in a typical system, I believe it deserves another backup method (besides the inclusion in standard backup procedures, which are, I repeat, *CRITICAL!*).

3. Quick start

1. run bakonf with `-L0` to archive all config files and create its database:

   ```
   # bakonf -L0
   #
   ```

   If everything went well, bakonf has created an archive under `/var/lib/bakonf/archives` named after your host. Look into that directory to find it. If any errors have occured, bakonf will tell you:

   ```
   [user@test user]$ bakonf -L0
   Error: cannot read '/nfs/README': 'Permission denied'. Not archived.
   Warning: '/sbin/lsusb -vv' exited with status 1.
   Warning: '/sbin/sfdisk -d /dev/hda' exited with status 1.
   [user@test user]$
   ```

2. run daily (or more often) bakonf with `-L1` to archive only the changed files since the previous step. This archive should be much smaller. It will be easy, *after encryption*, to email it.

   ```
   # bakonf -L1
   #
   ```

3. every week, go back to the first step.

4. Details about bakonf

bakonf home page is http://www.nongnu.org/bakonf/. The project is hosted at http://savannah.nongnu.org/projects/bakonf/.

You can regard bakonf as a rescue archive creator. It creates an archive of configuration files and metadata information about the system. By metadata, I mean data which is not found anywhere in the filesystem as simple files: partition table layout, hardware configuration, kernel version, etc.

Table 1. Archive layout

Filename	Description	Created when
`README`	A file which contains informations about the archive: when it was generated, with wich options and on what host	Always
`unarchived_files.lst`	A file which contains details about which files couldn't be backed up; this can happen when bakonf is not run as root, or when it scans NFS directories	When filesystem backup has been performed
`commands_with_errors.lst`	A file which contains details about which commands have exited with non-zero status. Their output is still stored in the archive, though.	When metadata backup has been performed
`filesystem/`	Under this path are stored the files backed up.	When filesystem backup has been performed
`metadata/`	Under this path are stored the files resulted from metadata backup.	When metadata backup has been performed

4.1. Filesystem backup

Bakonf can be operated in two modes: level 0, level 1. In level 0, it archives all of the specified configuration files and registers the state of those configuration files in a database (called virtuals database), of type Berkley DB. In level 1, it archives only the files modified since the last level 0 backup; it does this by comparing the virtuals database with the filesystem.

4.1.1. File types and states

Here is how bakonf treats files

directories

: they won't be archived if they don't contain files to be backed up. On the other hand, for each file to be backed up, bakonf will also backup (non-recursively) its parent directories (except root) so that you have the user, group, mtime and permissions of each directory. For example, if `/usr/local/etc/myconfig` has been selected for archiving, bakonf will actually archive this list of items: `/usr`, `/usr/local`, `/usr/local/etc`, `/usr/local/etc/myconfig`.

regular files

: regular files will be archived by bakonf if they aren't excluded by the `noscan` configuration directive. In case this is a partial backup (as opossed to a full backup), bakonf will make the following tests:

1. size of file saved in database $\neq$ actual file size? if true, the file will be backed up;

2. md5 hash saved $\neq$ actual md5 hash? if true, file will be backed up;
3. sha hash save $\neq$ actual sha hash? if true file will be backed up;
4. file will not be backed up.

Here we could introduce another form of backup, only file metadata backup, in case file contents hasn't changed, but file mode (permissions, owner) have.

symbolic links

bakonf doesn't follow symbolic links ever; it treats a symbolic link like a configuration file (its configration value resided in its name and its target). The tests made by bakonf are, in order:

1. link target must be equal, or the file is backed up
2. user and group ownership must be equal, or the file is backed up
3. permission bits must be equal, or the file is backed up
4. the file is not backed up

block devices, character devices, fifos, sockets

bakonf always selects these to backed up. Of course, some of them won't be backed up by tar, but regarding bakonf, it will select those for backup.

when the file type in archive is different than actual file type

in this case, bakonf always selects the file for archiving

4.2. Metadata

Metadata allows you to have more information about a system than is available in the filesystem. The current implementation allows you to store output of shell commands. Suggestions about other items are welcome.

4.2.1. Partition table

The most important metadata item is partition table about your disks, in the eventuality that you have a data error in partition table. The command to back this up varies, for example:

OS: GNU/Linux
Command: sfdisk -d /dev/hda

OS: FreeBSD
Command: fdisk /dev/ad0

4.2.2. Device list

Having the device list is and their hardware configuration is useful in order to have a quick overview if you want to clone the configuration from one system to another (to see corespondece between config files and hardware config). Examples of scanning the configuration:

OS: GNU/Linux
Command(s): lspci -vv; lsusb -vv

OS: FreeBSD
Command(s): pciconf -lv; usbdevs -v

4.3. What can I use bakonf for?

At least for:

- Configuration rollback. Since the archives are small, you can keep many versions, but unlike in differential backup, here one archive contains all the needed data.
- Configuration cloning. You can take a bakonf-generated archive from one system to another and 'clone' as much of the settings as you want.
- Quick restore of a server in case of catastrofic harddisk failure. Just reinstall the OS and put the config files back.

4.4. Requirements

To use bakonf, you must have the following:

- a Unix-like operating system (with compatible filesystem layout)
- python (http://www.python.org/) version 2.2 or higher
- optik (http://optik.sourceforge.net/) - command line parser for python, version 1.3 or higher
- tarfiel (http://www.gustaebel.de/lars/tarfile/) - tar module for python (it is expected to be include in python 2.3)
- bz2 (http://python-bz2.sf.net/) bzip interface module for python, only if you want bzip2-compressed archives
- xml parsing support for python, in order to use python's xml.dom.minidom; feel free to hack bakonf to change the configuration system if you want so

5. Configuration

> **Note:** Older bakonf versions (< 0.5) had an entirely different config file. If you upgraded, be sure to forward you changes to the new config files.

bakonf uses a main configuration file `/etc/bakonf/bakonf.xml`, which does some standard settings and tells bakonf what other files to include. These additional files are usually located in `/etc/bakonf/sources` and tell bakonf how to handle some special cases.

5.1. Configuration language

The configuration file is written in XML, and must have the document tag *bakonf*.

The file can contain the following tags:

<include path="*CONFIGFILE_PATTERN*"/>

tells bakonf to also parse any files which match the given shell pattern. These are files which modify bakonf's own behavior, and are usually located in /etc/bakonf/sources. These are not directories to be backed up! (Altough, if modified, they will be, since are located under /etc usually).

<filesystem>

Starts declarations about files to be backed up, and can contain:

<scan path="*SHELL-PATTERN*"/>

tells bakonf to add any file or directory which matched shell pattern *PATTERN* to its include list. These can be files or directories. Bakonf will descend directories, but will not *follow* symbolic links! The symlinks are considered configuration items also, so they will be backeup up themselves.

<noscan regex="*REGEXP*"/>

tells bakonf to ignore any file or directory which matches the regular expression REGEXP from the archive; it won't even open or analise those files.

<metadata>

Starts declarations about metainformations to be included in the archive, and can contain:

<storeoutput command="*SHELL-COMMAND*" destination="*PATH IN ARCHIVE*"/>

Tells bakonf to execute the given *SHELL-COMMAND* and include its output in a file named `metadata/`*PATH IN ARCHIVE* in the created archive. For example, the element:

```
<storeoutput command="cat /proc/version" destination="proc/version"/>
```

will create a file in the archive with the name `metadata/proc/version` which will contain the `/proc/version` file.

The order of precedence for scan/noscan is:

- bakonf will start scanning all items defined with scan
- if at any point in the filesystem scan, the current file/directory mathes any one of noscan REGEXPs, the scan will ignore it. For directories, that means ignoring all the files they contain, so please be careful about it.

Using these, you can select where you want bakonf to look for files for archiving. The default config file includes `/etc`, `/usr/etc`, `/usr/local/etc` and some others (look in `/etc/bakonf` after installing).

Example main configuration file (the file included in the distribution):

```
<bakonf>
<!--
     Bakonf main configuration file. Tune to your system.
     See also the files in the sources subdirectory (included by default)
-->
<filesystem>
    <!-- Standard directories -->
    <scan path="/etc" />
    <scan path="/usr/etc" />
    <scan path="/usr/local/etc" />
    <scan path="/var/lib/alternatives" />
</filesystem>
<!-- Include by default the other configuration files -->
<include path="/etc/bakonf/sources/*.xml" />
</bakonf>
```

Example GNU/Linux proc configuration file (included in the distribution):

```
<bakonf>
<!--
     File which contains hardware related informations, mostly
     extracted from the /proc filesystem (under GNU/Linux)
-->
<metadata>
<!-- Proc values -->
<storeoutput command="cat /proc/version" destination="proc/version"/>
<storeoutput command="cat /proc/dma" destination="proc/dma"/>
<storeoutput command="cat /proc/interrupts" destination="proc/interrupts"/>
<storeoutput command="cat /proc/ioports" destination="proc/ioports"/>
<storeoutput command="cat /proc/iomem" destination="proc/iomem"/>
<storeoutput command="cat /proc/cpuinfo" destination="proc/cpuinfo"/>
<storeoutput command="cat /proc/partitions" destination="proc/partitions"/>
<!-- Also hardware information -->
<storeoutput command="/sbin/lspci -vv" destination="lspci.txt"/>
<storeoutput command="/sbin/lsusb -vv" destination="lsusb.txt"/>
</metadata>
</bakonf>
```

5.2. File list

bakonf is composed of:

`/etc/bakonf/bakonf.xml`

Main configuration file.

`/etc/bakonf/sources/*.xml`

Configuration files for special cases (config files outside of etc dirs).

`/usr/sbin/bakonf.py`

Main program

`/etc/cron.d/bakonf`

Cron file, by default it does not run bakonf, you must uncomment a line to run it.

`/var/lib/bakonf/archives`

Default directory for configuration file archives.

> **Caution**
>
> You must decide yourself what to do with the configuration archives after bakonf creates them!

6. Using bakonf

6.1. Backup phase

For details about the actual parameters to bakonf, see its manpage.

To use bakonf, choose to either:

- run it manually, when you want, either always with `-L0` or with a combination: weekly with `-L0`, daily with `-L1`
- use the provided cron script to run it automatically, or create your own

In any case, you have to do something with the generated archives. Write the to floppy, tape, CD, other machine, but don't just ignore them, you defeat the purpose of bakonf.

6.2. Restore phase

6.2.1. Configuration rollback

In this case, just make sure you have the bakonf-generated archive near the date in the past you are interested in. If so:

1. if your system uses packages/ports, compare the actual package list with the one recorded by bakonf when it created the archive

2. install/remove software as needed
3. copy the configuration files for the services you want to rollback over the current files

6.2.2. Complete system restoration

If you had a catastrophical failure, you should follow these steps:

1. Reinstall the operating system on a clean machine. Use the given information in the `/metadata` directory in the archive to achieve an as close as possible configuration as the old system (e.g. partition layout, packages installed, etc.)
2. Copy all the files in the archive in the filesystem, overwriting the defaults from the packages.