Backupninja

`backupninja` is a utility that coordinates backup activities on a system. It is run by `cron`, by default every hour, and executes scripts for which the system administrator has dropped a configuration snippet into `/etc/backup.d`. The term that `backupninja` uses for configuration snippets is "backup actions". A backup action can specify that it wants to be executed less often than every hour - the default is once every day.

I see `backupninja` as an extension of cron, but with a configuration syntax that is more amenable to express the needs of backup jobs.

Contents

Debian packages

The only package needed is

```
backupninja
```

An optional package which allows the `sys` backup action to save Debian package selections:

```
debconf-utils
```

Other optional packages which allow the `sys` backup action to collect various system information:

```
hwinfo
util-linux
```

References

- Website (https://0xacab.org/riseuplabs/backupninja)
- The original website (https://labs.riseup.net/code/projects/backupninja) has gone stale and has been placed into read-only mode. The wiki may contain information that is still usable, such configuration documentation (https://labs.riseup.net/code/projects/backupninja/wiki/Configuration).

Configuration

Main configuration

The main configuration file is

```
/etc/backupninja.conf
```

The main configuration file defines things like

- Location of the folder that contains backup actions. The default is `/etc/backup.d`.
- The default value for "when", i.e. the default frequency at which backup actions should be run. The default is every day at 01:00 AM.
- Email address that should receive notification emails. The default is `root`.
- Whether or not to send a notification when a backup was successfully created. The default is "yes", this can be changed to "no" so that emails are sent only in the case of errors.

Backup actions

`backupninja` executes so-called "backup actions". A backup action is defined by a configuration file that is located in

```
/etc/backup.d
```

The suffix of the configuration file identifies the type of the backup action. For instance, a configuration file with the suffix

```
.mysql
```

tells `backupninja` that this action performs a MySQL database dump.

Configuration files that end with `.disabled` or begin with `0` (digit for zero) are skipped.

Backup action handlers

A backup action is executed by a so-called "handler". The suffix of the backup action's configuration file defines which handler script is to be run. Handler scripts are `bash` scripts that are located in

```
/usr/share/backupninja
```

For instance, the following script handles all `.mysql` backup actions:

```
/usr/share/backupninja/mysql
```

My preferred solution how to add customized handlers to `backupninja` is outlined below in the section Custom backup action handlers.

Configuration file format

- Configuration files for backup actions are in .ini style, including support for sections
- Use backslash ('\') to break long lines into several lines
- The backup action's type determines which sections (if any) and options are allowed in the backup action's configuration file
- The only common option is the `when` option, which determines when the backup action is run. The `when` option must appear before any other option. The `when` option can be omitted if the default frequency of "everyday at 01:00 AM" is sufficient.

Important: Configuration files must **not** be world writable/readable, otherwise `backupninja` will refuse to run. `backupninja` does not just refuse to run that single backup action - it will *entirely stop working*!

Scheduling

A backup action is executed according to the value of the `when` option in its configuration file. For instance:

```
when = hourly
```

The main configuration file specifies a default value for `when`

```
when = everyday at 01:00
```

A backup action can specify multiple values for `when`, in which case all of the specified times apply. `backupninja` itself is run by `cron` every hour, at the hour, so backup actions cannot be run more frequently.

Important: If backup actions take a long time, multiple copies of backupninja might be running at the same time!

ninjahelper

The curses-based wizard

```
ninjahelper
```

has limited support for creating new backup actions. Not all action types can be created like this, only those action types for which `ninjahelper` has a helper script. It is possible to add new helper scripts to `ninjahelper` - read its man page for details, I have not investigated this.

Security

Because backup data contains sensitive information, it is important to know how `backupninja` handles file and folder permissions. The most important thing is that `backupninja` runs with this `umask`:

```
umask 077
```

This means that files and folders created by backup actions by default are **not** group-readable or world-readable. Because `cron` runs `backupninja` as `root`, files and folders by default are owned by `root`. A backup action handler is free to set its own permissions - for instance the PostgreSQL handler sets ownership to user `postgres` (which at least on Debian is OK).

Here are some examples for files and folders created by backup actions:

```
root@pelargir:~# ls -l /var/backups/sysreport.txt
-rw------- 1 root root 2160302 Jul 10 14:30 /var/backups/sysreport.txt

root@pelargir:~# ls -ld /var/backups/ldap
drwx------ 2 root root 4096 Jul 10 01:00 /var/backups/ldap

root@pelargir:~# ls -ld /var/backups/postgres
drwx------ 2 postgres root 4096 Jun 19 19:01 /var/backups/postgres
```

Other files except configuration

The log file

```
/var/log/backupninja.log
```

Folder with scripts that run the various backup actions - so-called "backup action handlers"

```
/usr/share/backupninja
```

Testing / debugging / manual execution

Perform a basic test of the configuration, without actually executing anything:

```
backupninja --test
```

Run all backup actions now instead of at the time when they are scheduled

```
backupninja --now
```

Run only a single backup action instead of all actions

```
backupninja --now --run /etc/backup.d/01-dump-all-postgresql-databases.pgsql
```

Print all log messages to the console

```
backupninja --now --debug
```

Tests can also be run via the curses-based wizard

```
ninjahelper
```

Custom backup action handlers

Storage

Support for a custom backup action type can be added by dropping a custom handler script with the name of the desired custom suffix into the folder that contains the handler scripts. Because that folder will be overwritten when the Debian package is updated, I place my custom handler scripts in

```
/usr/local/htb/share/backupninja
```

I then create symlinks in the folder that contains the handler scripts, which point to the actual custom handler scripts. For instance, the following symlink adds support for a custom tar handler that is called "tar-local":

```
ln -s /usr/local/htb/share/backupninja/tar-local /usr/share/backupninja
```

Note: The `htb` folder is a reference to "herzbube's toolbox", my personal collection of useful scripts. On my server I keep a local clone of the HTB git repository in `/usr/share/htb`. Cf. the HTB wiki page.

Custom "tar" handler

The default `tar` handler includes a mandatory date suffix so that two tar backup actions don't overwrite each other's results. This is not what I want, I want the tar ball to have always the same name, just like database dumps that also have the same name. In my opinion, it is the responsibility of a post-backup rotation job to provide for multiple levels of backups.

For this reason I wrote my own `tar` handler. I'm calling it `tar-local` to prevent a name clash with the existing `tar` handler. Assuming you have a git clone of the HTB repository in `/usr/local/htb`, the handler is activated like this:

```
ln -s /usr/local/htb/share/backupninja/tar-local /usr/share/backupninja
```

The custom handler is a copy of the original handler, with the following changes:

- Add option `date_suffix` that can take the values "yes" and "no"
  - Default is "yes" which causes the custom handler to behave as the original handler
  - If "no" is specified no date suffix is added to either the backup file nor to the ".list" or ".err" files
- Add option `absolute_names` that can take the values "yes" and "no"
  - Default is "no" which causes the custom handler to behave as the original handler
  - The difference is that value "no" now allows the backup action to specify relative paths for the "includes" and "excludes" options. The handler changes the current working directory to the root directory (`cd /`) before it executes the `tar` operation, and switches back to the original working directory after the `tar` operation so as not to cause trouble to the master program, which might expect a certain working directory.
  - If the backup action sets "absolute_names" to "yes" it can specify absolute paths for the `includes` and `excludes` options. The handler will not change working directories, but instead it will add the option `--absolute-names` to the `tar` command line, to force `tar` to accept absolute path names. Without that command line option, `tar` would convert absolute path names to relative path names by stripping away the leading "/". tar would also emit the following warning:

```
/bin/tar: Removing leading `/' from member names
```

    This is precisely what happened in the original `tar` backup action handler: Backup actions had to specify an absolute path, and tar would strip away the leading "/", emitting the warning into the `.err` file.

- The handler now respects `backupninja`'s `--test` option and does not perform the `tar` operation if that option has been specified
- The handler suppresses the warning "file changed as we read it" which is emitted by `tar` if a file changed while `tar` was in the process of archiving it. I don't want to treat this as an error, I assume that the changed file content will be in the next backup.
- Eliminated a bit of code duplication regarding how the file names are built

Example backup actions for this handler can be seen further down in the section Backup actions on pelargir.

You can find the most recent code of the handler at the HEAD of the HTB git repository (https://git.herzbube.ch/htb.git/blob/HEAD:/share/backupninja/tar-local).

Custom "git" handler

`backupninja` does not provide a handler for backing up `git` repositories, so I had to write my own. Assuming you have a git clone of the HTB repository in `/usr/local/htb`, the handler is activated like this:

```
ln -s /usr/local/htb/share/backupninja/git /usr/share/backupninja
```

Here are the main characteristics of the implementation:

- Can backup 1-n repositories. Each repository is backed up separately.
- Can backup both bare repositories (the default) and repositories with a working tree
- Repositories with a working tree: Temporarily clones the repository into a bare repository, then backs up that bare repository. This is probably not a very good solution if the repository is large, because it requires additional disk space and takes additional time for cloning. But it works for me, and that's sufficient.
- The backup is made with tar
- The handler assumes that there are no activities on the repository at the time of the backup. If there are activities, the handler will not detect this, and the backup might not represent a repository in a consistent state. Again, this works for me.

Example backup actions for this handler can be seen further down in the section Backup actions on pelargir.

You can find the most recent code of the handler at the HEAD of the HTB git repository (https://git.herzbube.ch/htb.git/blob/HEAD:/share/backupninja/git).

Backup actions on pelargir

Preconditions for all backup actions

Two preconditions that all my backup actions must meet:

- A backup action always writes to the same backup file. It is the responsibility of a post-backup rotation job to provide for multiple levels of backups. The Debian package `logrotate` is ideally suited for this.
- A backup action writes uncompressed data. The data can be compressed later on (e.g. by `logrotate`), or fed into a backup utility such as `bup` or `zbackup` that prefers uncompressed input to allow data deduplication.

PostgreSQL

The backup action for PostgreSQL databases uses the `pgsql` handler provided by `backupninja`.

```
root@pelargir:~# cat /etc/backup.d/10-all-postgresql-databases.pgsql
backupdir = /var/backups/postgres
# Note: when using 'all', pg_dumpall is used instead of pg_dump, which means
# that cluster-wide data (such as users and groups) are saved.
databases = all
compress = no
```

Notes:

- The values I specify for the options "backupdir" and "databases" correspond to their default values, but I like to keep these values explicit
- This backup action requires that the file `.pgpass` exists in the `postgres` user's home directory. See the PostgreSQL wiki page for details.

On `pelargir` the following backup files are created by this backup action:

```
root@pelargir:~# ls -la /var/backups/postgres
total 900
drwx------ 2 postgres root       4096 Jun 19 19:01 .
drwxr-xr-x 9 root     root       4096 Jul 10 14:30 ..
-rw------- 1 postgres postgres 913163 Jul 10 01:00 pelargir-all.sql
```

OpenLDAP

The Debian package for `backupninja` does not include a handler for ".ldap" backup actions. I had a look at the handler that upstream provides (https://labs.riseup.net/code/projects/backupninja/repository/revisions/13f247afebde199ffb03d23e8662d362d5681ca6/raw/handlers/ldap.in), but that is geared towards an OpenLDAP installation that works with a `slapd.conf` config file. On my system, though, OpenLDAP is using the `slapd.d` config scheme.

It didn't make much sense for me to write a custom handler because I don't have a scenario for reuse, so I just wrote a simple shell script that performs the backup. The shell script is implemented as a backup action that uses the `sh` handler provided by `backupninja`.

```
root@pelargir:~# cat /etc/backup.d/11-all-openldap-databases.sh
# ------------------------------------------------------------
# This script assumes that no authentication is necessary.
#
# The dumps are generated with slapcat, the resulting LDIFs
# therefore are suitable for use with slapadd, not ldapadd.
# See the slapcat man page for details.
#
# This script assumes that it is safe to run slapcat without
# shutting down slapd. According to the slapcat man page
# this assumption is safe to make for a couple of backend
# types, specifically for backend type "bdb".
#
# This script does not compress the dumps it creates. The
# dumps can be compressed later on, or fed into a backup
# utility such as bup or zbackup that prefers uncompressed
# input to allow data deduplication.
# ------------------------------------------------------------

# Space-separated list of database indexes. Index 0 is the
# configuration database, which is backed up separately as
# part of the /etc directory structure.
INDEXES_OF_LDAP_DATABASES_TO_BACKUP="1"

BACKUP_DIR=/var/backups
LDAP_BACKUP_DIR=$BACKUP_DIR/ldap

# Fake loop. This allows code inside the loop to use "break"
# in order to abort execution. Without this fake loop, code
# would need to use "exit" to abort execution, which would
# be bad because "exit" aborts the entire backupninja script.
while true; do
  SLAPCAT="$(which slapcat)"
  if test $? -ne 0; then
    error "slapcat not found on this machine"
    # The fake loop allows us to use "break"
    break
  fi

  if test ! -d "$LDAP_BACKUP_DIR"; then
    mkdir -p $LDAP_BACKUP_DIR
    if test $? -ne 0; then
      error "Failed to create backup directory: $LDAP_BACKUP_DIR"
      # The fake loop allows us to use "break"
      break
    fi
  fi
  for INDEX_OF_LDAP_DATABASE_TO_BACKUP in $INDEXES_OF_LDAP_DATABASES_TO_BACKUP; do
    info "Creating dump of LDAP database $INDEX_OF_LDAP_DATABASE_TO_BACKUP"

    LDAP_BACKUP_FILE="$LDAP_BACKUP_DIR/database-$INDEX_OF_LDAP_DATABASE_TO_BACKUP.ldif"
    if test -f "$LDAP_BACKUP_FILE"; then
      rm -f "$LDAP_BACKUP_FILE"
      if test $? -ne 0; then
        error "Failed to delete existing dump file: $LDAP_BACKUP_FILE"
        continue
      fi
    fi

    "$SLAPCAT" -n "$INDEX_OF_LDAP_DATABASE_TO_BACKUP" >"$LDAP_BACKUP_FILE"
    if test $? -ne 0; then
      error "Failed to create backup directory: $LDAP_BACKUP_DIR"
      contiune
    fi
  done

  # Unconditionally break out of the fake loop
  break
done
```

On `pelargir` the following backup files are created by this backup action:

```
root@pelargir:~# ls -la /var/backups/ldap
total 20
drwx------ 2 root root 4096 Jul 10 01:00 .
drwxr-xr-x 9 root root 4096 Jul 10 14:30 ..
-rw------- 1 root root 8824 Jul 10 01:00 database-1.ldif
```

MySQL

The backup action for MySQL databases uses the `mysql` handler provided by `backupninja`.

```
root@pelargir:~# cat /etc/backup.d/12-all-mysql-databases.mysql
backupdir = /var/backups/mysql
databases = all
compress = no

# Use mysqldump to create backup. The alternative is mysqlhotcopy.
# We don't specify mysqldump options, so the defaults are used.
# These are the defaults:
# --lock-tables --complete-insert --add-drop-table --quick --quote-names
sqldump = yes

# Run MySQL commands as system user "root". A valid .my.cnf must
# exist in root's home directory which contains the database
# username and password for the following commands:
# - mysql: The handler runs this to determine the database names
# - mysqldump: The handler runs this to peform the actual backups
user = root
```

Notes:

- As noted in the comments above, this backup action requires that the file `.my.cnf` exists in the `root` user's home directory. See the MySQL wiki page for details.
- The handler is nice enough to create a separate backup file per database. This is in contrast to `mysqldump --all-databases`, which is usally used to simplify a MySQL backup task, and which dumps all databases into the same file. This feature requires that the `backupninja` handler is capable of enumerating the databases, i.e. the handler must be able to run the `mysql` command.

On `pelargir` the following backup files are created by this backup action:

```
root@pelargir:~# ls -la /var/backups/mysql/sqldump
total 316980
drwx------ 2 root root      4096 Jul 10 16:42 .
drwxr-xr-x 3 root root      4096 Jul 10 15:44 ..
-rw------- 1 root root   6435934 Jul 10 16:42 bugzilla.sql
-rw------- 1 root root   1923771 Jul 10 16:42 drupal714_grunzwanzling.sql
-rw------- 1 root root   8450475 Jul 10 16:42 drupal714_herzbube.sql
-rw------- 1 root root   1967656 Jul 10 16:42 drupal714_kino.sql
-rw------- 1 root root   1061207 Jul 10 16:42 drupal744_grunzwanzling.sql
-rw------- 1 root root   6047405 Jul 10 16:42 drupal744_herzbube.sql
-rw------- 1 root root    759603 Jul 10 16:42 drupal744_kino.sql
-rw------- 1 root root   1038366 Jul 10 16:42 drupal750_grunzwanzling.sql
-rw------- 1 root root   6939294 Jul 10 16:42 drupal750_herzbube.sql
-rw------- 1 root root   1120343 Jul 10 16:42 drupal750_kino.sql
-rw------- 1 root root   5077189 Jul 10 16:42 information_schema.sql
-rw------- 1 root root 283126868 Jul 10 16:42 mediawikidb.sql
-rw------- 1 root root    555415 Jul 10 16:42 mysql.sql
-rw------- 1 root root     16859 Jul 10 16:42 performance_schema.sql
-rw------- 1 root root     17815 Jul 10 16:42 phpmyadmin.sql
```

Home directories

Home directories must be backed up at least because they contain `Maildir` folders. Home directories are backed up with `tar`, using the custom `tar-local` backup action handler presented further up on this page.

```
root@pelargir:~# cat /etc/backup.d/20-home-directory-francesca.tar-local
backupname = home-directory-francesca
backupdir = /var/backups/home-directories
date_suffix = no
compress = none
includes = ./home/francesca

root@pelargir:~# cat /etc/backup.d/20-home-directory-patrick.tar-local
backupname = home-directory-patrick
backupdir = /var/backups/home-directories
date_suffix = no
compress = none
includes = ./home/patrick

root@pelargir:~# cat /etc/backup.d/20-home-directory-root.tar-local
backupname = home-directory-root
backupdir = /var/backups/home-directories
date_suffix = no
compress = none
includes = ./root
```

On `pelargir` the following backup files are created by this backup action:

```
root@pelargir:~# ls -la /var/backups/home-directories
total 871532
drwx------ 2 root root      4096 Jul 20 15:49 .
drwxr-xr-x 9 root root      4096 Jul 20 06:25 ..
-rw------- 1 root root         0 Jul 20 15:49 home-directory-francesca.err
-rw------- 1 root root    232394 Jul 20 15:49 home-directory-francesca.list
-rw------- 1 root root 224102400 Jul 20 15:49 home-directory-francesca.tar
-rw------- 1 root root         0 Jul 20 15:49 home-directory-patrick.err
-rw------- 1 root root   1193342 Jul 20 15:49 home-directory-patrick.list
-rw------- 1 root root 666060800 Jul 20 15:49 home-directory-patrick.tar
-rw------- 1 root root         0 Jul 20 15:49 home-directory-root.err
-rw------- 1 root root      1653 Jul 20 15:49 home-directory-root.list
-rw------- 1 root root    829440 Jul 20 15:49 home-directory-root.tar
```

Directory /usr/local

The directory `/usr/local` must be backed up because it contains various server-local files such as custom shell scripts or the CAcert root certificates. The directory is backed up with `tar`, using the custom `tar-local` backup action handler presented further up on this page.

```
root@pelargir:~# cat /etc/backup.d/21-usr-local.tar-local
backupname = usr-local
backupdir = /var/backups/usr-local
date_suffix = no
compress = none
includes = ./usr/local
```

On `pelargir` the following backup files are created by this backup action:

```
root@pelargir:~# ls -la /var/backups/usr-local
total 52
drwx------ 2 root root  4096 Jun 20 01:55 .
drwxr-xr-x 9 root root  4096 Jul 10 14:30 ..
-rw------- 1 root root     0 Jul 10 01:06 usr-local.err
-rw------- 1 root root   974 Jul 10 01:06 usr-local.list
-rw------- 1 root root 40960 Jul 10 01:06 usr-local.tar
```

Directory /var/www

The directory `/var/www` must be backed up because it contains files served by the Apache web server. While some of those files represent code bases that could be easily replaced (e.g. Drupal), other files are hand-crafted, original content that must be preserved. The directory is backed up with `tar`, using the custom `tar-local` backup action handler presented further up on this page.

```
root@pelargir:~# cat /etc/backup.d/22-var-www.tar-local
backupname = var-www
backupdir = /var/backups/var-www
date_suffix = no
compress = none
includes = ./var/www
```

On `pelargir` the following backup files are created by this backup action:

```
root@pelargir:~# ls -la /var/backups/var-www
total 203316
drwx------ 2 root root      4096 Jun 20 02:18 .
drwxr-xr-x 9 root root      4096 Jul 10 14:30 ..
-rw------- 1 root root         0 Jul 10 01:06 var-www.err
-rw------- 1 root root   1016661 Jul 10 01:06 var-www.list
-rw------- 1 root root 207165440 Jul 10 01:06 var-www.tar
```

Git repository for /etc

I manage the content of the folder `/etc` in a Git repository so that I can track configuration changes over time. Instead of backing just the plain folder, I back up the entire Git repository. The repository is backed up with `tar`, using the custom `git` backup action handler presented further up on this page.

```
root@pelargir:~# cat /etc/backup.d/30-etc.git
repositories = /etc
repositories_are_bare_repos = no
compress = none
backupdir = /var/backups/git
```

On `pelargir` the following backup files are created by this backup action:

```
root@pelargir:/var/backups/git# ls -l etc.git.*
-rw------- 1 root root       0 Jul 10 01:06 etc.git.err
-rw------- 1 root root  152938 Jul 10 01:06 etc.git.list
-rw------- 1 root root 4106240 Jul 10 01:06 etc.git.tar
```

Git repositories managed by gitolite

I am hosting a number of Git repositories on my server using gitolite (see GitServer). The repositories are backed up with `tar`, using the custom `git` backup action handler presented further up on this page.

```
root@pelargir:~# cat /etc/backup.d/31-gitolite.git
repositories = /var/lib/gitolite3/repositories
repositories_is_container_dir = yes
repositories_are_bare_repos = yes
compress = none
backupdir = /var/backups/git
```

On `pelargir` the following backup files are created by this backup action:

```
root@pelargir:/var/backups/git# ls -l *.git.* | grep -v etc.git
-rw------- 1 root root        0 Jul 19 21:36 acexpander.git.err
-rw------- 1 root root     1898 Jul 19 21:36 acexpander.git.list
-rw------- 1 root root  4720640 Jul 19 21:36 acexpander.git.tar
-rw------- 1 root root        0 Jul 19 21:36 dgsmonx.git.err
-rw------- 1 root root     1517 Jul 19 21:36 dgsmonx.git.list
-rw------- 1 root root   460800 Jul 19 21:36 dgsmonx.git.tar
-rw------- 1 root root        0 Jul 19 21:36 doxystub.git.err
-rw------- 1 root root      949 Jul 19 21:36 doxystub.git.list
-rw------- 1 root root 10045440 Jul 19 21:36 doxystub.git.tar
[...]
```

System information

The presence of a backup action of type `sys` triggers a handler that collects information about the system. The content of the backup action's configuration file is largely irrelevant, the only thing that matters is the `when` option that, if present, allows for non-default scheduling.

```
root@pelargir:~# cat /etc/backup.d/90-system-information.sys
# The presence of this configuration file triggers a backup
# action of type "sys", which backs up various vital system
# information. The comments further down were copied verbatim
# from the following URL:
# https://labs.riseup.net/code/projects/backupninja/wiki/Sys


# this config file will save various reports of vital system information.
# by default, all the reports are enabled and are saved in /var/backups.
#
# (1) a list of all the packages installed and removed.
#     this file can be used to restore the state of installed packages
#     by running "dpkg --set-selections < dpkg-selections.txt
#
# (2) the partition table of all disks.
#     this partition table can be used to format another disk of
#     the same size. this can be handy if using software raid and
#     you have a disk go bad. just replace the disk and partition it
#     by running "sfdisk /dev/sdb < partitions.sdb.txt"
#     (MAKE SURE YOU PARTITION THE CORRECT DISK!!!)
#
# (3) hardware information.
#     a simple report is generated of the kernel modules, the devices,
#     and the model of the hardware which 'discover' is able to detect.
#
```

Notes:

- To collect information about installed Debian packages, the Debian package `debconf-utils` must be installed so that the command line utility `debconf-get-selections` is present
- To collect partitioning information, the Debian package `util-linux` must be installed - the package provides the utilities `lsblk` (to list block devices) and `sfdisk` (to read partitioning info)
  - The original handler on Debian 9 (stretch) uses `sfdisk` to list block devices from which to obtain partitioning information. Unfortunately `sfdisk` also lists devices such as RAID devices (/dev/mdx) which don't have partition tables. This causes `backupninja` to emit warnings, which in turn triggers an annoying daily mail to be sent. The problem has been fixed upstream (https://0xacab.org/riseuplabs/backupninja/merge_requests/7/diffs) by using `lsblk` to list block devices. I have manually applied the patch to the handler script `/usr/share/backupninja/sys`.
- To collect hardware information, the Debian package `hwinfo` must be installed

On `pelargir` the following backup files are created by this backup action:

```
root@pelargir:~# ls -lrt /var/backups
[...]
-rw------- 1 root     root   19017 Jul 10 14:30 dpkg-selections.txt
-rw------- 1 root     root   69796 Jul 10 14:30 debconfsel.txt
-rw------- 1 root     root 2160302 Jul 10 14:30 sysreport.txt
-rw------- 1 root     root     351 Jul 10 14:30 partitions.sdb.txt
-rw------- 1 root     root     361 Jul 10 14:30 partitions.sda.txt
-rw------- 1 root     root   35172 Jul 10 14:31 hardware.txt
```

Backup rotation

I am using `logrotate` to rotate backup files. This wiki page shows the actual configuration.

Retrieved from "https://wiki.herzbube.ch/index.php?title=Backupninja&oldid=6683"