Robinhood

Monitoring and purge tool for large file systems

v 1.0.6
January 15, 2009

Thomas LEIBOVICI
CEA/DAM

<thomas.leibovici@cea.fr>

Table of contents

1. Product description

1.1 Overview

Robinhood is a tool for auditing and purging file systems. It is designed in order to process all its tasks in parallel, so it is particularly adapted for scanning large file systems with millions of entries and petabytes of data.

Robinhood regularly checks the used space on a file system, and it also checks independently the usage of file system's storage units (specifically, OSTs' usage in Lustre file system). When one of those units (or the whole file system) exceeds a given high level threshold, then Robinhood is able to purge only the files that are stored on this specific storage unit, without purging files stored on other OSTs.
Files are deleted according to their last access/modification time, starting by the oldest entries. However, the administrator can also specify some purge priority modifiers based on multiple properties (file size, name…).

Indeed, with Robinhood:

- Administrator can define some "whitelist" rules, for preserving files he wants to keep indefinitely. Those rules can be a combination of multiple conditions about entries' owner, group, name, path, type, depth, etc…

- He can also penalize (using "blacklist" rules) some profiles of entries that could result in lower file system performances or harm to system stability and availability, and also files that monopolize storage resources. For example, if a user creates a file whose size is near the total file system size, you may think is it not normal this causes the deletion of other users' recent files… To avoid this, you can for example define a rule that penalize files that are bigger than a given size. Then, you can give a constant penalty to such files, or you can choose an amount depending on their size (using a linear function).

As a result, the product makes it possible to preserve the quality of service and the files of the nice and responsible users, whereas it will target files of "abusers" and harmful behaviors…
That's why it is called Robinhood ;-)

1.2 How it works

Robinhood is a daemon that is to be running continuously. It ever maintains a list of filesystem entries, with associated properties (standard POSIX attributes, the storage units on which they are located). Thus, it always has a "fresh" knowledge of filesystem's content in case it has to free some space quickly.
In order to build such a list efficiently, even if the filesystem has millions of files, a pool of threads scans filesystem directories in parallel.

An independent thread is also used for checking filesystem and storage units' usage. Its goal is to keep used space between a high and a low watermark. Thus, if a storage unit (or the global filesystem build upon those units) exceeds the high watermark, then this thread starts a purge of the files on this storage unit (only this one), until its usage reaches the low watermark.
In order to have a good reactivity, file deletion is done using a pool of dedicated threads, so that Robinhood can quickly free some space on the file system or unit, even if a filesystem scan is currently running.

1.3 Compiling

Robinhood distribution is based on autotools.
Untar Robinhood source tree and run the '`./configure`' script on your target system.

Then, run '`make rpm`' to build a rpm package for this platform.
The rpm is generated in the 'rpms/RPMS/<arch>' directory of the source tree. It contains Robinhood binaries, man pages and init script.

1.4 Command line

By default, binaries are installed in the `/usr/sbin` directory.

You can launch it with the following command line:

```
robinhood  [-d]  -f <config_file>
```

- **-d :** detach the daemon from the parent shell and attach it to init.
- **-f :** gives the path for Robinhood's configuration file.

It can also be executed using the following options:

- **-h :** display help about the command line
- **-@ :** display version info
- **-T <output_file> :** this generates a configuration template in output file

1.5 Dynamic reconfiguration

SIGHUP (kill -1) reloads daemon's configuration (see section 2.3 to know which parameters can be modified dynamically).

2. The configuration file

2.1 Generating a template file

To easily create a configuration file, you can generate a template using the –T option of robinhood.

Ex:

```
    /usr/sbin/robinhood    -T    /etc/robinhood.d/myfs.conf
```

2.2 Syntax

a – Global structure

The configuration file consists of several blocks, each of them consisting of key/value peers, separated with semi-colon:

```
BLOCK_1 {
      Key = value;
      Key = value;
      Key = value;
}

BLOCK_2 {
…
}
```

b – Type of values

A value can be:

- A string delimited by single or double quotes (‘ or “) ;
- A Boolean. Both of the following values are accepted and the case is not significant: TRUE, FALSE, YES, NO, 0, 1, ENABLED, DISABLED.
- A numerical value (decimal representation).
- A duration, i.e. a numerical value followed by one of those suffixes: *d* for days, *h* for hours, *min* for minutes, *s* for seconds. Examples: 1s ; 1min; 3h ; …
  NB: if you do not write a suffix, the duration is interpreted as seconds.
  Ex: 60 will be interpreted at 60s, i.e.1 min.
- A size, i.e. a numerical value followed by one of those suffixes: *PB* for petabytes, *TB* for terabytes, *GB* for gigabytes, *MB* for megabytes, *kB* for kilobytes. No suffix is needed for bytes.

c - Comments

A ‘#’ sign indicates the beginning of a comment (except if it is placed in a quote delimited string). The comment ends at the end of line.

Examples:

```
# this is only a comment line
x = 32 ; # a comment can also be placed after a significant line
```

2.3 Configuration parameters

a – The different kinds of blocks

As it was described in section 2.2.a, a configuration file consists of several blocks, related to distinct configuration aspects.

Thus, a configuration file can use four types of blocks:

- The « GENERAL » block: general daemon's parameters (only 1 general block in a configuration file).

- « WHITELIST » blocks: this type of block defines some conditions for "whitelisting" a certain kind of filesystem items. If a file or directory matches all the conditions given in a "whitelist" block, it will never be deleted (a logical AND is applied between the conditions in a whitelist block, and a logical OR is applied between all whitelist blocs).

- « WHITELIST_ALL_BUT » blocks: if such a bloc is defined in the configuration file, then all items of the file system will be whitelisted, except those that match the conditions of at least one of those blocks.
  NB: this kind of block cannot be used together with WHITELIST blocks.

- « BLACKLIST » blocks: those blocks define some penalties that can apply to non-whitlisted entries of the filesystem. Each "blacklist" block is only based on a single condition. For each non-whitelisted file, we apply the maximum penalty of the blacklist blocks it matches (not the sum).

b – The 'GENERAL' block

The 'GENERAL' configuration block defines the following parameters:

Note: some parameters can be reloaded at daemon's run-time using "kill -1". Those parameters are tagged 'dyn' in the following section.

File system relative parameters

- **path** (string) : give the path of the file system to be man**age**d.
  Ex: `path = '/mnt/parallel_fs' ;`
- **fstype** (string) : indicate the type of filesystem (as shown by *mount*).
  Ex: `fstype = 'lustre' ;`

Logs and alerts relative parameters

- **debug_level** (string, dyn): indicate the trace level of robinhood. Allowed values are (from the more verbose to the more quiet) :
  - **FULL**: highest level of verbosity. Trace everything.

  - **DEBUG**: trace information that could be used for debugging, and for determining the cause of a potential bad program behavior.
  - **VERB**: high level of traces (but usable in production). Until this level, the list of all files is dumped to the log after each filesystem audit.
  - **EVENT**: standard production log level (the file list is not dumped to the log file).
  - **MAJOR**: only trace major events.
  - **CRIT**: only trace critical events.

- **log** (string): the file where traces are to be written.
- **report** (string) : the file where report about purged files is to be written (size, access and modification time, …)
- **stats_period** (duration, dyn): robinhood periodically generates into log file some statistics about its activity (filesystem scan status, filesystem content summary, total purged volume, memory usage, …). This parameter thus defines stats generation periodicity.
- **mail_alert** (string): a comma separated list of mail addresses where alerts are to be sent.
- **dir_count_alert** (integer, dyn): this parameter defines a mail alert when a directory exceeds a given number of entries (0 disables this alert).
- **file_size_alert** (size, dyn): this parameter defines a mail alert when a file exceeds a given size (0 disables this alert).

Remote administration service options

- **max_top_count** (integer) : this is the maximum top size that can be retrieved by admin client. **/!** A high count will use more memory and CPU when scanning filesystem.
- **svc_register** (boolean) : indicate whether RPC administration service must register on portmapper.
- **rpc_service** (string or numerical value) : set the RPC service number or service name of the RPC server (used when svc_register = TRUE).
- **port** (string or numeric value, optional) : fix a given port number for the rpc server. If it is not specified, robinhood registers to any port. Use this option if you have trouble with network filters or firewalls.
- **remote_adm_grant** (string): you can specify several conditions for granting access to a client that uses "robinhood_adm" remote administration tool (see section 3). Conditions can be on "user", "group", "host" or "network". Each condition consists in a comma-separated list of <condition_type>=<value> peers. An admin client must match one of the "remote_adm_grant" strings for being granted. For matching a "remote_adm_grant" string, it must match all of its <condition_type>=<value> peers.

  Ex1: admin client must be root on the "123.12.0.0/16" network for being granted:
  ```
  remote_adm_grant = "user=root,network=123.12.0.0";
  ```

  Ex2: admin client must be a member of the "admin" group on host "cws-cluster":
  ```
  remote_adm_grant = "group=admin,host=cws-cluster";
  ```

Filesystem scan parameters

- **min_audit_period**, **max_audit_period** (durations, dyn): Robinhood is able to adapt its audit frequency to filesystem usage. Indeed, it is not necessary to scan filesystem frequently when it is empty (because no purge will be needed for a long time). However, the more the filesystem is used, the more we need a fresh list for purging files. Thus, specify delay between filesystem scans using:
    - **min_audit_period**: the audit frequency when filesystem is full;
    - **max_audit_period**: the audit frequency when filesystem is empty.

  Then, the effective period for scanning the filesystem will be:
  `min + (100% - max storage usage)*(max-min)`

- **audit_retry_delay** (duration, dyn): in case a filesystem scan aborts because of a hang (when a problem on the filesystem makes the threads stuck on syscalls), this defines the time before retrying to scan the filesystem.
- **nb_threads_audit** (integer): indicates the number of threads used for scanning the filesystem. The higher this value is, the fastest the scan will be, but the more the load on filesystem will be.
- **audit_op_timeout** (duration, dyn): indicate filesystem call timeout when scanning. When a thread is stuck more than this duration in a filesystem call, robinhood will abort the thread and will operate a scan recovery (It will continue auditing the filesystem, excluding this "suspect" part of filesystem tree. This makes it resilient to potential metadata corruption in a filesystem).

Purge general options

- **age_del_empty_dir** (duration, dyn): empty directories are not removed using the same policy as files (policy not based on access time). Indeed, a directory is removed if it has been empty for a given duration that this parameter specifies. Of course, a directory is not purged as long as it matches a whitelist rule. Setting this parameter to 0 disables directory removal.
- **min_age_purge** (duration, dyn): whatever the penalties that can be defined on a file, it is never purged if it has been used during the period this parameter specifies.
  Ex: if min_age_purge = 6h, robinhood won't purge files that have been accessed during the last 6 hours.
- **nb_threads_purge** (integer): the number of threads used for purging filesystem.The higher this value is, the faster the filesystem space could be freed, but the load on the filesystem will be more important.
- **fs_usage_check_period** (duration, dyn): specify the frequency for checking filesystem and storage units usage.
- **purge_op_timeout** (duration, dyn): indicate filesystem call timeout when deleting files. When a thread is stuck more than this duration in a filesystem call, robinhood will abort the thread and will skip the deletion of the corresponding file. So, this makes it resilient to potential metadata corruption in a filesystem).
- **post_purge_df_latency** (duration, dyn): some filesystems do not update their disk usage statistics immediately after unlink operations (specifically when freeing allocated data is made asynchronously). Thus, it can be necessary to wait a while after purging a lot of files, before issuing a new *df*, if we want to get a correct usage value.

So, this parameter defines the time robinhood has to wait before measuring usage again after a purge.

Global filesystem usage thresholds

- **high_purge_threshold** (integer between 1 and 100, dyn): robinhood will start purging the filesystem when usage exceeds this percentage (high watermark).
- **low_purge_threshold** (integer between 1 and 100, dyn): robinhood will stop purging files when usage reaches this percentage (low watermark).

Storage units' usage thresholds

- **storunit_high_purge_threshold** (integer between 1 and 100, dyn): robinhood will start purging files of a given storage unit (OST) when its usage exceeds this percentage (high watermark).
- **storunit_low_purge_threshold** (integer between 1 and 100, dyn): robinhood will stop purging files of an OST when the space used decreased to this percentage (low watermark).

Behavior and safety

- **no_delete** (boolean, dyn): if this boolean is set to TRUE, robinhood doesn't really delete files (simulation mode).
- **stay_in_fs** (boolean): if this Boolean is set to TRUE, robinhood won't traverse mount points during filesystem audits.
- **monitor_only** (boolean): if this Boolean is set to TRUE, robinhood will only generate stats about files, users, etc. but no massive purge will be done. This highly decreases memory usage since the daemon doesn't need to maintain a list of files to be deleted.
- **lock_file** (string): when this file is created, robinhood will finish its current operations and then, won't start any new scan nor purge as long as the lock file exists.

Advanced configuration

- **spooler_ckeck_period** (duration, dyn): this is the internal scheduler's time tick. This defines the period for checking if a filesystem scan must be started, or for testing threads' activity.
- **purge_queue_size** (integer): this is the size of the purge queue. It impacts "fluidity" of purge phases. You should set the following value :
  `(high_threshold - low_threshold) * total_fs_space / mean_file_size`
- **nb_prealloc_files** (integer): this defines the size of memory pages used for file structures allocation mechanism (number of pre-allocated structures). You should set it to a power of 2, near the average number of files in the filesystem.
- **nb_prealloc_tasks** (integer): this defines the size of memory pages used for tasks structures allocation mechanism (number of pre-allocated structures). You should set it to a power of 2, near the following value:
  `number of audit threads * filesystem average depth`
- **nb_prealloc_storitems** (integer): this defines the size of memory pages used for stripe structures allocation mechanism (number of pre-allocated structures). You should set it to a power of 2, near the following value:
  `number of files in the filesystem * mean stripe count`

c – ‘WHITELIST’ and ‘WHITELIST_ALL_BUT’ blocks

‘WHITELIST’ blocks describe entries that must not be deleted. ‘WHITELIST_ALL_BUT’ blocks are the opposite: they define the kind of entries that can be removed.
However, the two kind of blocks have the same syntax: inside such a block, an item must match all the conditions for being whitelisted (in the case of WHITELIST block), or eligible for removal (in the case of a WHITELIST_ALL_BUT block). Thus, a logical AND is applied between the conditions of such a block, and a logical OR is applied between the different blocks. As a consequence, an item is whitelisted if it matches any WHITELIST block. Note that WHITELIST blocks are tested in the same order as they appear in the configuration file. So, to improve performances, be sure to write first the conditions that are more likely to be matched.

Whitelist/Whitelist all but conditions

- **user**: this defines a condition about item’s owner name. It can be an exact name, or a shell regular expression (see *fnmatch(3)*). You can also use the special value “nouser” for matching entries whose owner does not correspond to any existing user.
  Ex : user = ‘toto’ ;
- **group**: this defines a condition about the group owning an item. It can be an exact name or a shell regular expression (see *fnmatch(3)*). You can also use the special value “nogroup” for matching entries whose gid does not correspond to any existing group.
  Ex : group = ‘s*’ ;
- **uid**: this is a condition about owner’s numerical uid. You can specify a minimum, maximum or an exact value, the following way (like *find* arguments) :
  - ‘+uid’: matches values ≥ <uid>.
  - ‘-uid’: matches values ≤ <uid>.
  - ‘=uid’: exactly matching <uid> value.
- **gid**: this is a condition about group’s numerical gid. The syntax and meaning are the same as the uid condition.
- **tree**: this is a regular expression or an exact path in the filesystem. All the sub-trees under matching entries are also matched (this is recursive).
  E.g: tree = “/tmp/keep_[1-9]*/dir”;
- **path**: unlike the tree condition, this only matches objects having this path (not the subdirectories) i.e. the condition “path=/a/*” will preserve “/a/b” directory from rmdir, but its child entry “/a/b/c” could be deleted.
  Ex : path = ‘/*/*.x’ ;
- **name**: this defines a pattern about file name or directory name, wherever it is in the filesystem.
  E.g: name = “*.log”;
- **depth**: this matches the depth for an entry in the filesystem. Like the uid or gid conditions, you can define a minimum, maximum or exact value, using ‘+’, ‘-‘ or ‘=’ signs. Note that the depth is counted from the “fs_path” given in the GENERAL block of the configuration file. Thus, if fs_path is “/tmp”, the object “/tmp/toto” will be considered as being at depth 0.
- **type**: with this, you can specify an object type of the filesystem. Expected values are “file”, “directory” or “symlink”.
- **size**: condition about file size. Like uid, gid and depth, this is a value prefixed with a ‘+’, ‘-’ or ‘=’ sign. What’s more, the value is followed by a size unit.
  E.g.: size = ‘+12GB’ matches files with 12 Gigabytes of data or more.

- **age**: this is a condition about the last data access or metadata change on the file (max of atime, mtime and ctime) **at the time when the filesystem is scanned**. For defining a minimum age at purge time, use the "GENERAL::min_age_purge" parameter instead.

- **external_command**: you can specify an external command for testing a condition on a filesystem object. This parameter can be a command line with several arguments, or a piped command sequence. The object path to be tested replaces '{}' sign (you must use exactly one '{}' in a command).

  E.g.: external_command = "/usr/bin/my_test_script.ksh –t test –f {} –u 4";

  /! Be careful not using *external_command* too much, because it results in *fork* operations and causes lower scan performances. We also advise not using it alone in a condition block: you should specify restrictive tests in the same configuration block, so the command will not be launched if it is not necessary. What's more, such a condition should be located in the last 'WHITELIST' block, so it will only be tested if the entry doesn't match another previous WHITELIST block.

  The command must respect to following specification:
  - This command must return a null status, except if an error occurs during command execution;
  - It must write "YES" on standard output if the file matches condition ;
  - It must write "NO" on standard output if the file doesn't match condition ;
  - Several instances of this command can be run at the same time.

Examples:

In the following example, we consider that robinhood runs on fs_path = "/tmp".

Example 1:

The following WHITELIST item preserves directories at the root of /tmp whose the name begins with a digit. However, their content can be purged.

```
WHITELIST
{
     type       = 'directory' ;
     depth      = '=0' ;
     name       = '[0-9]*' ;
}
```

This is also equivalent to:

```
WHITELIST
{
     type     = 'directory' ;
     path     = '/tmp/[0-9]*' ;
}
```

Example 2:

Now, we want to preserve all the log files in the directory "/tmp/rep" and its subdirectories:

```
WHITELIST
{
     tree  = '/tmp/rep' ;
     name  = '*.log' ;
}
```

Example 3:

For preserving all objects that owns to a certain range of users (uid ≤ 100):

```
WHITELIST
{
     uid = '-100'; # matches 100 or smaller uids.
}
```

Example 4:

Don't remove objects that match a user-defined condition. This condition is only expected for files that are not in filesystem's root, and whose name matches "*pattern*". When using '`external_command`', it is very important to write as much restrictive tests as possible in the same block, so the command will not have to be executed for all entries in filesystem.

```
WHITELIST
{
     type = file;
     depth = '+0';
     name = '*pattern*';
     external_command = '/usr/bin/test_condition.ksh {}';
}
```

c – 'BLACKLIST' blocks

With BLACKLIST blocks you can apply some penalties to specific profiles of entries in the filesystem. If an item matches several BLACKLIST conditions, the max of their penalties is applied (not their sum).

A 'BLACKLIST' block looks like this:

```
BLACKLIST
{
     criteria  = type_of_condition ;
     value     = 'the value to be matched for this condition' ;
     penalty   = 'the amount of penalty' ;
}
```

- **criteria**: this indicates the condition on which the penalty is based. Expected values are: *user*, *group*, *tree*, *name* or *size*.

- **value**: to be penalized, a filesystem entry must match this value for the given criteria. The syntax is the same as whitelist rules: regexp for users, groups, paths, and "+<value><unit>" for matching a size.
- **penalty**: this is the amount of penalty applied to files that match the blacklist condition. Penalty is a duration: applying a penalty makes a file older than it is actually, in order to remove it quickly.
  You can understand a penalty as "this kind of entry must be purged before files than have been accessed during the last …"
    - For *user*, *group*, *tree* and *name* conditions, this penalty must be a constant duration. E.g.: '1d' (1 day), '4h' (4 hours).
    - For a condition about *size*, you can give a constant duration, but you can also have a penalty that will be size dependant (using a linear formula). In this case, formula must have the following syntax : '<*duration*>/<*size_unit*>'.
      Example: '1d/100GB' means "1 day of penalty per 100GB".
      Thus, a 130GB file will have a penalty of 1.3 days.

Examples:

Example 1:

The following block penalizes all files in the '/tmp/grp/toto' filesystem tree (including subdirectories), so that they will always be removed before the files accessed during the last hour.

```
BLACKLIST
{
     criteria  = 'tree' ;
     value     = '/tmp/grp/toto' ;
     penalty   = '1h' ;
}
```

Example 2:

This will penalize files bigger than 100GB, with a penalty of 12h/100GB. Thus, a 600GB file will never cause the removal of files accessed during the last 3 days, a file of 1TB cannot remove files accessed during the last 5 days, etc…

```
BLACKLIST
{
     criteria  = 'size' ;
     value     = '+100GB' ;
     penalty   = '12h/100GB' ;
}
```

3. Remote administration tool

3.1 Overview

It can be useful for an administrator to consult statistics about filesystem (global space used, space used for each Lustre OST, number of files, max file size…).
What's more, it can be necessary to ask Robinhood to make an action on the filesystem: purging some space in the global filesystem, or only on a given OST…

To do this, *robinhood* provides a RPC-based client that makes it possible for the administrator to display statistics or submit some orders to *Robinhood*, even from a remote machine.

2 modes are available for using this client:

- an interactive mode, that is similar to a shell;
- a batch mode, for executing commands on a single command line or in a script.

3.2 Interactive mode

a – Command line

For using *robinhood-adm* in interactive mode, use the '**-i**' option :

```
robinhood-adm -i [-v|-c][-h host][-s service | -p port]
```

The following options can be specified:

- **-v**: use verbose mode;
- **-c**: output in *csv* format instead of human readable and pretty display format;
- **-h**: specifies the host on which *Robinhood* is running. If it is not specified, the question will be displayed in order for the user to answer in an interactive way.
- **-s**: specify the RPC service (name or number) on which *robinhood* is registered (see **rpc_service** parameter, described in 2.3.b).
- **-p**: if you don't want to use RPC portmapper and *robinhood* is registering to a fix port number (this can be useful if network traffic is filtered by a proxy…), this option specifies the port number robinhood is using (see **port** parameter in section 2.3.b).

b – Interactive session

When you launch it, the admin client tool asks the node on which r*obinhood* is running, and the service it registered on (if you did not specify them on the command line parameters). It first issue a request to *robinhood* daemon to ensure it responds well. A prompt is then displayed, so you can execute administration commands. To known the list of available commands, type '**help**'.

```
robinhood_adm>help

Commands :

   help   : display this help

   fsinfo : display information about filesystem
            (space used, statistics about its content)

   topdirs [count]: display the largest directories
            count: (optional) number of directories to be displayed

   toppurge [count]: display the first files to be purged
            count: (optional) number of files to be displayed

   topsize [count]: display biggest files in the filesystem
            count: (optional) number of files to be displayed

   userinfo [user]: display usage info by user
      <user> : only display info for the given user name

   ostdf [threshold] : display OST usage
      <threshold> : makes it possible to display only OSTs that
                    exceed the specified threshold
   Example : 'ostdf 80' : only display OST whose used space is over 80%

   purgefs <LW|percent> [force]: start a filesystem purge
      LW    : purge to the low watermark
      <percent> : specifies the percent of filesystem space to be purged
      force : do not ask a confirmation
   Example : 'purgeFS 10 force' : purges 10% of the filesystem

   purgeost <ostid> <LW|percent> [force]: start a purge on a single OST
      <ostid> : index of the OST to be purged
      LW      : purge this OST to the low watermark
      <percent> : specifies the percent of OST space to be purged
      force   : do not ask a confirmation
   Example : 'purgeost 1 15 force' : purge 15% of OST 1

   show_watermarks : display high and low watermarks from the configuration

   terminate_daemon : send a stop request to daemon
```

c – Interactive session example

```
bash$ /usr/local/bin/Robinhood-adm -i -h localhost -s robinhood_ptmp

        === Welcome to robinhood_adm interactive mode ===
        type 'help' for getting details about commands

robinhood_adm> fsinfo


Filesystem /ptmp on host localhost
Space used: 40.57%

Details :
  Total space      : 16.79 TB
  Used space       : 6.47 TB
  Free space       : 9.47 TB
  Number of files  : 9944
  Total volume     : 10.74 GB
  Biggest file     : 1.65 GB
  Oldest purgeable file : 2007/07/09 09:38:22 (age = 186d 15min 12s)
  Last audit done on 2008/01/11 08:50:57 in 03s (status: partial)
  No purge running
```

```
robinhood_adm> ostdf

--- Storage units usage ---
Storage unit #0: 38.27% used; 165097876 blocks of 4096 available=629.80 GB
Storage unit #1: 37.54% used; 167046724 blocks of 4096 available=637.23 GB
Storage unit #2: 33.29% used; 178410982 blocks of 4096 available=680.58 GB
Storage unit #3: 33.69% used; 177340330 blocks of 4096 available=676.50 GB
Storage unit #4: 39.81% used; 160975998 blocks of 4096 available=614.07 GB
Storage unit #5: 64.57% used;  94761356 blocks of 4096 available=361.49 GB
Storage unit #6: 63.14% used;  98596034 blocks of 4096 available=376.11 GB
Storage unit #7: 38.06% used; 165666270 blocks of 4096 available=631.97 GB
Storage unit #8: 37.61% used; 166864834 blocks of 4096 available=636.54 GB
Storage unit #9: 38.86% used; 163522125 blocks of 4096 available=623.79 GB

robinhood_adm> purgeost 0 10
Purge 10% of OST #0 ? [yY/nN] :y

Starting a purge of 10% on OST #0...
2008/01/11 08:56:20 : Purge is running...
2008/01/11 08:56:35 : Purge finished !!!


robinhood_adm>quit
```

3.3 Batch mode

a – Command line

For executing an administration command in batch mode, do not use the '-i' option, and specify your command on the command line (the same as you would write in interactive mode).

```
robinhood-adm [-v|-c][-h host][-s service|-p port] <command>
```

The following options can be specified:

- **-v**: use verbose mode;
- **-c**: output in *csv* format instead of human readable and pretty display format;
- **-h**: specifies the host on which *Robinhood* is running. If it is not specified, the question will be displayed in order for the user to answer in an interactive way.
- **-s**: specify the RPC service (name or number) on which *robinhood* is registered (see **rpc_service** parameter, described in 2.3.b).
- **-p**: if you don't want to use RPC portmapper and *robinhood* is registering to a fix port number (this can be useful if network traffic is filtered by a proxy…), this option specifies the port number robinhood is using (see **port** parameter in section 2.3.b).

b – Examples

➢ Displaying information about filesystem usage :

```
bash$ robinhood-adm -h localhost -s robinhood_ptmp fsinfo
```

```
Filesystem /ptmp on host localhost
Space used: 40.58%

Details :
  Total space      : 16.79 TB
  Used space       : 6.47 TB
  Free space       : 9.47 TB
  Number of files  : 9944
  Total volume     : 10.74 GB
  Biggest file     : 1.65 GB
  Oldest purgeable file : 2007/07/09 09:38:22 (age = 186d 25min 47s)
  Last audit done on 2008/01/11 09:04:06 in 03s (status: complete)
  No purge running
```

- Displaying information about users ('csv' mode):

```
bash$ robinhood-adm -h localhost -s robinhood_ptmp -c userinfo
```

User;FileCount;TotalVolume;MaxFileSize;MeanFileSize
toto;5;1231122;561146;246224
titi;7748;9325220101;1769154560;1203564
tutu;1;19;19;19
martin;46;265923;215626;5780
root;1211;46956523;45950399;38774
dupond;16;15792;11625;987
john;1;19;19;19
jimmy;4;2631;1311;657

- Purging 10% of OST #1 (force = do not ask confirmation)

```
bash$ Robinhood-adm -h localhost -s robinhood_ptmp purgeost 1 10 force

Starting a purge of 10% on OST #1...
2008/01/11 08:56:20 : Purge is running...
2008/01/11 08:56:35 : Purge finished !!!
```

4. Robinhood “one-shot”

4.1 Overview

As described in section 1.2, Robinhood is a daemon that is ever running, so it can cache a lot of information about filesystem (file stripping, storage units information, etc…), and doesn’t need to ask them again each time it refreshes its knowledge of filesystem’s content. What’s more, it continuously checks available disk space, and it only starts removing files when some free space is needed. Also, it can free disk space immediately without having to scan the filesystem again, because it has all the information it needs in memory. Thus it is particularly adapted for huge filesystems with an intensive use.

However, you might need using robinhood’s features (like fast multi-threaded scan, whitelist/blacklist items definitions) in a simple way, without being obliged to have such a process ever running on your machine. This is what “Robinhood one shot” does: you just launch it when you want to free some disk space, it remove files according to the rules you defined, and then stops.

4.2 Main features

a - Just like a ‘*find*’

Unlike its big brother (Robinhood daemon), Robinhood “one shot” doesn’t care about filesystem usage or Lustre storage units occupation. Just consider it as an improved “`find /tmp -exec rm`” with some matching conditions.
Unlike its brother, it doesn’t build a list of files sorted by access time: it removes files “on the fly” depending on the rules you defined.
Like its brother, filesystem scan and deleting file is done by a pool of threads, so it is very fast even on large filesystems.

b - Specifying conditions

Blacklist rules don’t have the same meaning as in the Robinhood daemon. Indeed, in the daemon version of this product, ‘BLACKLIST’ items specify some penalties for files, so they can be removed before other files, depending on a specific property.
Robinhood “one shot” rules are simpler: you just specify what kind of entries must be kept (using WHITELIST or WHITELIST_ALL_BUT items), and what files must be removed (with a new kind of block called “REMOVE”). The choice was made not to call it “BLACKLIST”, because they are different from Robinhood daemon’s “BLACKLIST” rules.

- If a filesystem entry matches any “WHITELIST” block, it will never be deleted.
- If a filesystem entry doesn’t match any “WHITELIST_ALL_BUT” blocks, it is preserved.
- A files that is not whitelisted is removed if it matches any “REMOVE” block.

c - "REMOVE" blocks

You can specify how many 'remove' items you want in a configuration file. They have the same syntax as 'WHITELIST' blocks (see 2.3.c).
If a (non-whitelisted) item matches all the conditions of such a block, it is removed immediately.

4.3 Command line

By default, program's binary is located in `/usr/sbin` directory.

You can launch it with the following command line:

```
robinhood-oneshot  -f <config_file> -P <filesystem_path>
```

- **-f :** gives the path of configuration file
- **-P:** specifies the path of the filesystem to be scanned

It can also be executed using the following options:

- **-h :** display help about the command line
- **-@ :** display version info
- **-T <output_file> :** this generates a configuration template in output file

NB: its configuration file is only a subset of Robinhood daemon's parameters. Generate a configuration file using the –T option to see what options are supported.