Yard Documentation

Tom Fawcett (fawcett AT croftj DOT net) November, 1998

Yard is a suite of Perl programs for making customized rescue disks for Linux. It exploits the compressed ramdisk feature of 1.3.48+ kernels. With this feature you can get a standard kernel image plus about 2.4 meg of utilities on a single floppy. You specify the set of files you want on the disk, and Yard does extensive checking of the files to increase chances that the rescue disk will work. A 2-disk option is available if one isn't enough. Yard can now be used either with or without Lilo.

Contents

1 Introduction

Yard is a suite of Perl programs for creating a so-called rescue disk (a single floppy disk comprising a kernel and a filesystem with useful rescue programs). Rescue disks are used when you can't (or don't want to) boot off your hard disk; they usually contain utilities for diagnosing and manipulating hard disks and filesystems. The Linux doc file *Bootdisk-HOWTO* contains much information about emergency bootdisks and how to make them.

Most Linux distributions (eg, Slackware) come with some kind of pre-fabricated rescue disk. These disks contain a fixed set of utilities that are useful but may not be enough in an emergency. For example, if you want to restore files off of backup tapes you'll need a restore program that matches your backup program. It's desirable to have a complete, self-contained set of up-to-date utilities and kernel. You don't want to discover after a disk crash that, for example, your floppy tape driver is newer than the rescue disk kernel and won't work.

Yard lets you specify a set of files to be included on the rescue disk. Yard's specifications are flexible. The primary feature of Yard is that it does some checking of your choices to make sure that the rescue disk will be usable. Yard checks for basic mistakes that I found myself making repeatedly: missing/unlinked libraries, unresolved symbolic links, typos, etc. These mistakes are tedious to search for and many of them can be detected automatically, which is why I wrote Yard.

Yard comprises three Perl programs: `make_root_fs, check_root_fs and write_rescue_disk`. These programs are designed to be run in sequence. They are separate so you can verify the output of one before going on to the next. Typically you iterate making and checking the root filesystem a few times before you write the rescue disk.

This document describes *Yard 1.17*.

2 Requirements

In order to use Yard, you need:

- A Linux kernel 2.0 or greater.

- The following features must be *built-in* to the kernel used on the rescue disk:
  - Ramdisk support
  - Ext2fs (second extended filesystem) support
  - Floppy disk support

  These *cannot* be provided as modules. The kernel needs to have them built-in so it can load the root filesystem.

- Perl version 5 or higher. Version 4 will not work. To see what you have, type

  ```
  perl -v
  ```

  .

- If you want to strip binaries, you need objcopy 2.6 or later (type `objcopy -V`)
- Root access. All of these scripts must be run as root.

3 Installation

From within the toplevel directory, issue these commands:

1. `su root`
2. `./configure`
3. `make`

   This creates copies of your `/etc/fstab` and `/etc/lilo.conf` and places them in the Replacements subdirectory with a few modifications, explained below.
4. `make install`

   Note that it will try to install copies of Bootdisk_Contents and Config.pl. If you have used a previous version of Yard, you may already have copies of these installed that you don't want overwritten. It will ask before overwriting any file; simply answer "no" in this case.

By default, the configure script will install files in the following directories:

```
Perl programs:                    /sbin
Replacement and extras dirs:      /etc/yard
Yard configuration files:         /etc/yard
  (Config.pl and Bootdisk_contents)
```

These defaults are based on my interpretation of the Linux File System Standard (FSSTND). If you don't like these, change the three directory definitions at the beginning of the `configure` script.

4 Using Yard – summary instructions

1. Customize Config.pl and Bootdisk_Contents.
2. Become root (`su root`).
3. `make_root_fs`
4. `check_root_fs`
5. Check/fix any errors reported in steps 3-4. Repeat steps 3-4 until satisfied, then insert fresh floppy into drive.
6. `write_rescue_disk`

Then shutdown and try to boot off the floppy.

5 Using Yard – detailed instructions

5.1 Customizing Config.pl and Bootdisk_Contents

If this is the first time you've used Yard, do the following:

1. Edit the two files Config.pl and Bootdisk_Contents in this directory. Both contain numerous comments which should explain in detail the options you're asked to set and the allowable values.

   Config.pl

   This contains basic information about what devices you'll be using and their capacities. Everything in it has to be in Perl syntax, but it's all simple variable assignments so you don't need to know Perl.

   Bootdisk_Contents

   This file specifies what your bootdisk will contain. Review this file carefully, especially the selections of `/etc` and `/sbin` files. Comments at the head of the file describe the options. I have included everything necessary for a simple boot sequence, but I don't know how much distributions vary in their structure. I've used Yard with both Slackware and RedHat with little modification.

   You will probably discover that the default Bootdisk_Contents contains more files than you can fit on a rescue disk. This is intentional: it is easier to delete unnecessary files from Bootdisk_Contents than to guess what files should be added to a minimal Bootdisk_Contents. Two other file sets are included for illustration.

   (a) `Bootdisk_Contents.sample` is a reasonable set of rescue files that fits on one disk.
   (b) `Bootdisk_Contents.minimal` is a small set of files.

Yard will catch many but not all errors. For example, if you change `/etc/inittab` to use `getty_ps` instead of `getty`, check_root_fs will make sure `getty_ps` is included on the disk, but it won't warn you that the calling syntax is different.

Check the files in the `Replacements` subtree that comes with the Yard distribution. By default, any file mentioned in `Bootdisk_Contents` will be copied unchanged from your hard disk. Some things have to be changed, however, since a rescue disk is pared-down and can't access the hard disk during boot. You can specify that a different file be used in place of one mentioned, eg:

```
/etc/inittab <= ./Replacements/etc/inittab
```

This causes `./Replacements/etc/inittab` (which comes with Yard) to be used for `/etc/inittab` on the rescue disk. These replacement files are very short so there isn't much to check.

When you ran "make copies", the program `create_fstab.pl` created the file `./Replacements/etc/fstab` from your fstab. Two modifications are done:

1. Every device mentioned is given a *noauto* option so it won't be mounted automatically when the rescue disk is used.
2. The mount points are placed under `/OLDROOT` on the rescue root filesystem. You can mount them manually under `/OLDROOT` to re-create selectively your disk directory structure. This makes repairing them easier and allows you to `chroot/OLDROOT` to test the structure.

If you have a `/etc/lilo.conf`, the `make copies` will also create a file `./Replacements/etc/lilo.conf`. If you intend to use Lilo, check this file.

5.1.1 Dynamically loaded libraries.

Your system may require dynamically loaded libraries that are not visible to `ldd`. Brief comments on this are included in `Bootdisk_Contents`, and explained here.

PAM (Pluggable Authentication Modules). If your system uses PAM (Pluggable Authentication Modules), you must make some provision for it on your bootdisk or you will not be able to login. PAM, briefly, is a sophisticated modular method for authenticating users and controlling their access to services. An easy way to determine if your system uses PAM is to check your hard disks's `/etc` directory for a file `pam.conf` or a `pam.d` directory; if either exists, you must provide some minimal PAM support. (Alternatively, run `ldd` on your `login` executable; if the output includes `libpam.so`, you need PAM.)

Fortunately, security is usually of no concern with bootdisks, since anyone who has physical access to a machine can usually do anything they want anyway. Therefore, you can essentially disable PAM by creating a simple `/etc/pam.conf` file in your root filesystem that looks like this:

```
OTHER   auth       optional    /lib/security/pam_permit.so
OTHER   account    optional    /lib/security/pam_permit.so
OTHER   password   optional    /lib/security/pam_permit.so
OTHER   session    optional    /lib/security/pam_permit.so
```

Also copy the file `/lib/security/pam_permit.so` to your root filesystem. This library is only about 8K so it imposes minimal overhead.

Note that this configuration allows anyone complete access to the files and services on your machine. If you care about security on your bootdisk for some reason, you'll have to copy some or all of your hard disk's

PAM setup to your root filesystem. Be sure to read the PAM documentation carefully, and copy any libraries needed in `/lib/security` onto your root filesystem.

Yard's `check_root_fs` can detect whether you need PAM, and will warn you if you have not configured it. It will also scan the configuration file(s) and warn if you have specified a service with no corresponding library.

NSS (Name Service Switch). If you are using glibc (aka libc6), you will have to make provisions for name services or you will not be able to log in. The file `/etc/nsswitch.conf` controls database lookups for various servies. If you don't plan to access services from the network (eg, DNS or NIS lookups), you need only prepare a simple `nsswitch.conf` file that looks like this:

```
passwd:     files
shadow:     files
group:      files
hosts:      files
services:   files
networks:   files
protocols:  files
rpc:        files
ethers:     files
netmasks:   files
bootparams: files
automount:  files
aliases:    files
netgroup:   files
publickey:  files
```

This specifies that every service be provided only by local files. You will also need to include `/lib/libnss_files.so.1`, which will be loaded dynamically to handle the file lookups.

If you plan to access the network from your bootdisk, you may want to create a more elaborate `nsswitch.conf` file. See the `nsswitch` man page for details. Keep in mind that you must include a file `/lib/libnss_`*service*`.so.1` for each *service* you specify.

Yard's `check_root_fs` can detect whether you need to configure NSS, and will warn you if you haven't. It will also scan the configuration file and warn if you have specified a service with no corresponding library.

5.2 Making the root filesystem

When you're done customizing these two files, su to root and run:

```
make_root_fs
```

This program constructs an initial, uncompressed root filesystem that the rescue floppy will contain. It works in four passes. First it processes the `Bootdisk_Contents` file and notes special cases (links and replacements). Next it sets up linked file chains mentioned in the first step but not included. It looks for required library files (eg, libc.so) and the loaders required for them. Finally, it notes any hard-linked files. After that, if the files all fit, it constructs the filesystem.

`make_root_fs` will produce occasional messages showing what it's doing. Any errors should be prominent. Detailed output will go to `make_root_fs.log` in case you want to see exactly what the program did. If this is the first time you've run `make_root_fs` for a bootdisk, you should review the log file. In particular, Yard includes information about the libraries needed by your chosen files, which is worth checking. It is especially worth checking if you run out of space.

When you are satisfied with the output, proceed to the next step. You can also chdir to the mount directory and look at the rescue filesystem as it will exist at boot time.

5.3 Checking the root filesystem

At this point, run:

```
check_root_fs
```

This checks the root filesystem for errors and omissions. It knows about the format of inittab, fstab, passwd, pam.conf, etc., and checks them for problems. It also checks scripts for missing interpreters, missing files, etc. It also checks configuration of PAM and NSS, mentioned in the previous section.

Look over the warnings and go back to step 1 if necessary. The warnings are saved on `check_root_fs.log` for later inspection.

Note that you may get a lot of warnings like:

```
Home directory of userX (...) is missing
Shell of userX (/mnt/bin/bash) doesn't exist
```

You can ignore these if you don't intend to login as userX.

5.4 Creating the complete rescue disk

When you're satisfied with the output from `check_root_fs`, unmount any existing floppy in the drive, insert a new diskette, and run:

```
write_rescue_disk
```

This prorgam takes your kernel and the (compressed) root filesystem and produces a rescue disk from it. `write_rescue_disk` shouldn't produce any errors at this point unless you've exceeded the floppy capacity after compression. The program's output is copied to `write_rescue_disk.log`.

If you've selected the disk_set=DOUBLE option, the program will write the kernel onto the first disk, then prompt you to change disks, after which it will write the root disk.

If you've selected the disk_set=SINGLE option and your compressed filesystem is too large to be contained on a single floppy with the kernel, the program will offer to create a double-disk set.

If your compressed filesystem is too large to be contained on a floppy, you can try formatting your floppy at a higher density. 1440K is the default, but higher densities are possible with many drives (I have an old, unexceptional floppy drive and BIOS that will support densities up to 1722K). See man fd

and

man fdformat

for details.

5.5 Benediction

You're done. Shut down your system and try to boot from the floppy(ies).

If the boot fails, see the following section on Troubleshooting.

Note that when you boot the rescue disk, it will not automatically mount any of the devices mentioned in your /etc/fstab. This is intentional: rescue disks are often used when a hard disk is inoperational or in an inconsistent state, so hard disks should not automatically be mounted. If you want to mount your other devices, you'll have to do that manually.

The `create_fstab` script arranges for your existing hard disk entries to be placed under /OLDROOT so you can mount them manually. So from the rescue disk you can type:

```
mount /OLDROOT
```

and your hard disk partition usually mounted as root will be mounted under OLDROOT. For example, if you want to run a bash shell under your old root filesystem, you may simply do:

```
mount /OLDROOT
chdir /OLDROOT ; chroot /OLDROOT bash
```

6 Troubleshooting

If you followed the instructions and your disk won't boot, the first step is to determine where the boot is failing. In general, the further along it gets the easier it is to diagnose (though not necessarily to fix!). These steps are arranged more-or-less chronologically.

6.1 Configuration problems

These should be explained in the doc subdirectory. I try to catch as many problems as I can in the `configure` script, but Yard relies on many external programs, which change occasionally and introduce bugs.

6.2 Problems building the root filesystem

- If you're using /dev/ram0 as `$device` and Yard produces the error:

```
Error: You've declared file system size ($fs_size) to be 4096 K
but Linux says /dev/ram0 may only hold 0 K
```

Check your `/etc/lilo.conf` file for a line like:

```
ramsize = 0
```

and increase it to something more reasonable, like 4096. Remember to run lilo and reboot.

- If you're using a loopback device, occasionally the ext2 filesystem seems to get corrupted for no apparent reason. If you start getting errors in the middle of `make_root_fs` from the sys() complaining that a directory doesn't exist, this may be what is happening. I don't know why this happens and I can't reproduce it consistently. If it happens, try unmounting, deleting and re-creating the file. This seems to get rid of the problem.

6.3 Problems in kernel loading

In the normal boot process you will see a message sequence like:

```
LILO loading floppylinux...
Uncompressing...done
Now booting the kernel
```

If the sequence halts or displays an error somewhere in this sequence, the problem is with LILO.[1] With some exotic floppy disk geometries (usually involving disk capacities greater than 1440K) LILO's map compaction won't work, causing the LILO boot sequence to halt. If LILO halts and you're using a non-1440K floppy, this is likely the problem.

The fix is to remove the `COMPACT` line from your bootdisk's `lilo.conf`, and run Yard again. This will simply turn off LILO's map compaction which should fix the problem (although kernel loading will slow down somewhat).

If that doesn't work, switch to a 1440K floppy and try again. If you really want to puzzle it out, go read the section "Disk Geometry" in LILO's README file – and good luck.

If 1440K doesn't work, something is very broken. Make sure LILO has been installed correctly. If you don't normally use LILO to boot, re-install a recent version. As a last resort, remake your kernel with "make mrproper".

If you're using a *double*-disk rescue set, both floppies *must* be formatted identically. The boot loader becomes confused otherwise.

6.4 Problems finding the compressed root filesystem

If the loader tells you it can't find a compressed root image, make sure you gave Yard the correct floppy device (eg, `/dev/fd0H1722` for 1722K). If you've constructed a *single*-disk rescue set and it prompts you to insert a root floppy disk, that's probably a Yard problem (the rdev in write_rescue_disk has failed for some reason and the failure wasn't caught by Yard). Go back and look over write_rescue_disk.log to make sure nothing failed.

[1] Not necessarily *in* LILO, but with LILO.

6.5 Problems with init or login

1. If the system repeatedly accepts a login name then offers the login prompt again, this is a sign that the NSS is not configured. On systems using the GNU Name Service Switch (NSS), you must include explicitly in Bootdisk_Contents a selected set of `/lib/libnss\_*` shared libraries, as well as an `/etc/nsswitch.conf` file.

2. If you get a message like:

   Id xxx respawning too fast: disabled for n minutes

   This comes from init, usually indicating that your *getty* [2] or login is dying as soon as it starts up. Check the *getty* and login executables, and the libraries they depend upon. Make sure the invocations in /etc/inittab are correct.

3. If halting occurs *after* the root filesystem is loaded, the problem is in Yard or in you. The first step is to check `write_rescue_disk.log` to make sure nothing failed. If you ignored warnings from `make_root_fs` or `check_root_fs`, look at them again. You should probably look over all three log files to make sure things are as they should be. `make_root_fs.log` has a listing of where each file was taken from – make sure all your system files are coming from the right places on your hard disk.

4. If you get strange messages from *getty*, it may mean the calling form in `/etc/inittab` is wrong. `check_root_fs` does not perform much inittab call checking because the options are so variable among the different *getty* programs[3]. If you're using a different call and/or program from what you use in your hard disk `/etc/inittab`, double check it.

5. When you login, you may get errors about commands that aren't present. This usually happens when an rc file uses a builtin command that does not exist in the rescue disk's shell, but did exist in the shell used to run Yard.

6.6 And if none of that works...

Check the Yard webpage: `http://www.croftj.net/\~{}fawcett/yard/`

On that page I'll put notes people have sent me about problems they've run into, before I've had a chance to fix and re-release Yard.

6.7 Suggestions

If you find a problem that that Yard didn't catch, but which it could have caught, please let me know and I'll try to add a check to `check_root_fs`.

If you think it's a Yard problem you can try diagnosing it yourself. In fact, you may have better luck since you're more familiar with your setup. Otherwise, package up the three .log files along with your Bootdisk_Contents, Config.pl and Bootdisk_Contents.ls and send them to me with a reasonable description of what went wrong. This command should work:

[2] The notation *getty* will be used to mean some getty-like program, eg getty, agetty, mgetty or getty_ps.

[3] Different versions of agetty are reported to have different incompatible calling forms.

```
tar cvzf yard_bug.tgz *.log *.ls \
    /etc/yard/Bootdisk_Contents /etc/yard/Config.pl
```

Be sure to let me know which Yard version you're using. You might check the *yard webpage* to make sure there isn't already a newer version of Yard that fixes your problem, or a note added there.

7 Known bugs and limitations

1. Space calculations will never be completely accurate because of inode overheads and object file stripping.
2. Yard requires that you have ldconfig in your file set so Yard can use it to regenerate the cache. This is somewhat inelegant, since ldconfig isn't generally useful on a rescue disk. Eventually ldconfig may have some kind of "chroot" option to eliminate this need.
3. Occasionally when using a loopback device the ext2 filesystem will become corrupted for no apparent reason. See the note at the end of D (the loopback appendix).
4. Yard's method of determining the release number of a kernel image (ie, to derive what `uname -r` would print) is usually correct. Occasionally it returns the wrong version. This seems to happen when a kernel is remade without doing `make mrproper`.

8 To be done

If anyone has any suggestions (or better, patches) for these, please let me know.

1. Distribute a copy of libc-lite.so with Yard. Libc.so has become so large (700K+) even after stripping that this has become desirable again. I don't know how to create libc-lite, though I've scanned the ELF-Howto and the libc source code. Is this no longer possible with ELF? From the libc source code it looks as if the ability has been stripped out, leaving nothing but a few stubs in the Makefiles.

   An alternative to libc-lite is to dynamically create a rescue libc.so by extracting symbols from the binaries.
2. Implement the base+extra option for $disk_set. This would probably require extensive hacking.

9 Getting the latest copy of Yard

The latest version of Yard and its documentation should always be available from:

<http://www.croftj.net/fawcett/yard/>

The secondary home for Yard is on:

<http://sunsite.unc.edu/pub/Linux/system/Recovery/>

You should first check the Incoming directory (`<http://sunsite.unc.edu/pub/Linux/Incoming/>`) to make sure a more recent copy hasn't been released.

An HTML version of this documentation, plus changes, should be available from:

`<http://www.croftj.net/fawcett/Yard_doc.html>`

A companion set of prefabricated specifications files are available in the Yard-prefabs package, also available from *Sunsite's system/Recovery/* directory. Note that these are now old and may not be very useful.

10 Thanks, acknowledgements and theft

Thanks are due to the following people:

- Rick Lyons for his CRAMDISK package and much e-mail help.
- Karel Kubat's *SAR-disk package.*
- Graham Chapman's original *Bootdisk-HOWTO* document
- Paul Gortmaker's ramdisk.txt (in `/usr/src/linux/Documentation`)
- Bugs, patches, suggestions, encouragement: Horst von Brand, Jeremy Buhler, Peter Chubb, Ted Cox, Nick Duffek, Gerald Erdmann, A.P.Harris, Stephan Hegel, Jonathan Kamens, Kenneth Corbin, Marty Leisner, Thomas Quinot, Rick Lyons, Rob McMullen, Dan Morrison, Keith Owens, Steve Orr, Andrew Runnalls, Roderich Schupp, Soenke Schwardt, Murugan Subramanian, Jerry Sweet.

A Other information

Here are some other documents of interest:

- The *Bootdisk-HOWTO* is available at `<http://www.croftj.net/~fawcett/Bootdisk-HOWTO/>`
- The ramdisk code rewrite and new options are dicussed in `/usr/src/linux/documentation/ramdisk.txt`.

B Format of Bootdisk_Contents

For convenience, the format rules for entries in `Bootdisk_Contents` is given in comments at the beginning of that file. Here is a list of the allowable forms:

- Blank lines and whitespace may be used freely.
- Lines beginning with # or % are comments.
- Filenames may be either relative or absolute. Any filename not beginning with a slash is relative and will be resolved relative to the current directory.

- Lines of the form

  ```
  filename1 -> filename2
  ```

  will create symbolic (soft) links on the root filesystem. For example, if you want `sh` linked to `bash` in the root filesystem, you specify:

  ```
  /bin/sh -> /bin/bash
  ```

  (There is no way to specify hard links, though hard linked files that exist on the hard disk will be hard linked on the floppy.)

- Lines of the form `filename1 <= filename2` will cause filename2 to be copied to filename1 on the boot disk. This is useful for specifying trimmed-down replacements for /etc/passwd, /etc/inittab, etc. filename2 will be found first by searching PATH, then by searching relative to the current directory.

  A useful variant of this is:

  ```
  filename1 <= /dev/null
  ```

- Glob designations (?, * and [...]) are generally allowed, eg `/dev/hd[ab]*`. Wildcards are not allowed in link specs or replacement specs because no one knows what they mean.

- You may refer to environment variables in these specs by using a dollar sign. $RELEASE will be set the the release string of the kernel you specify (`$kernel`).

In most cases you don't need to specify shared libraries or loaders. The script will detect dependencies (via `ldd`) and include them automatically. The exception for this is PAM.

You don't need to explicitly specify intermediate directories unless you just want to make sure they exist. For example, if you mention /foo/bar/slog, Yard will make sure /foo and /foo/bar exist.

C Options for Config.pl

Here is a distillation of the options allowed in Config.pl. For convenience, most of this text also exists as comments in the Config.pl file. These options should have reasonable defaults, so if you don't understand what an option is for you can probably leave it alone.

$verbosity

This controls only what is printed to the screen. If 0, only the important messages will be printed; if 1, all messages. All messages end up in the log file regardless of the setting.

$floppy and $floppy_capacity

These variables specify the floppy device where the rescue disk will be written and its capacity. Make sure the two agree. If $floppy is a non-standard size (eg, 1722K), make sure to use the complete name (eg, /dev/fd0H1722).

$disk_set

One of "single", "double" or "base+extra"

- SINGLE: Both the kernel and entire compressed root filesystem will be put on one disk.

- DOUBLE: The kernel will be put on the first disk and the compressed root fs will be put on the second.
- BASE+EXTRA: THIS OPTION NOT YET IMPLEMENTED. The first disk will contain the kernel plus a base set of files (enough to boot and run tar). The second disk will contain the remaining files.

It is safe to keep this at SINGLE. If Yard detects that you need an extra disk, it will offer to make a double disk set automatically.

NB. With double disk sets, both disks *must* be formatted identically.

$mount_point

A directory to be used as a mount point. This is where the root filesystem will be mounted during creation and where the floppy will be mounted when the rescue disk is being written.

$device

The device for building the compressed filesystem. This can be `/dev/ram0` or a spare partition. You can turn off swapping temporarily and use the swap partition on your hard disk. You can use a loopback device if your kernel supports them – see section asdfasdf for instructions.

$fs_size

The size limit of $device, in Kilobytes. For /dev/ram0, this value should be no more than the ramsize specified in your `/etc/lilo.conf`. For most devices, Yard can check this value against the available space.

$kernel

The absolute filename of the compressed kernel to be put on the rescue disk. This should be the *compressed* kernel. This is usually something like `/vmlinuz,/zImageor/boot/zImage`. If you've just remade your kernel (via `make zImage`) the kernel file will reside in `/usr/src/linux/arch/i386/boot/zImage`.

$kernel_version

make_root_fs will examine $kernel and try to determine its version. If Yard guesses incorrectly, or if you want to force it anyway, set $kernel_version. The value should be a version string such as that returned by "uname -r".

(If you compile your kernel properly via mrproper so that /usr/src/linux/arch/i386/boot/setup is recompiled, Yard's method will work).

$contents_file

The file specifying the bootdisk contents specification file. The default is `Bootdisk_Contents` in the installation directory.

$rootfsz

The file that will temporarily hold the compressed root filesystem.

$oldroot

Where the old (hard disk) root filesystem will be mounted on the ramdisk filesystem. create_fstab uses this to adapt your /etc/fstab for use on the rescue disk so you'll be able to mount hard disk partitions more easily. You shouldn't need to change this, but run `create_fstab` again if you do.

$strip_objfiles

If set to 1, binary executables and libraries will be stripped of their debugging symbols (using `objcopy`) as they're copied to the root filesystem. This may reduce their size somewhat. If you don't understand what this means, leave it at 1. If you're sure you don't have objcopy, or for some reason you want debugging symbols, set it to 0.

$yard_temp

If non-null, specifies directory where log files will be written. If null, log files will be written to current working directory.

$use_lilo

Controls whether to use Lilo for transferring the kernel to the boot disk.

- If 1, Yard will use Lilo to boot the kernel (configuration file is in ./Replacements/etc/lilo.conf, created by `make copies`). This allows you to use Lilo's APPEND clause and various other Lilo options.
- If 0, Yard will copy the kernel directly to the rescue disk. This saves a small amount of space.

@additional_dirs

This is an array that should contain any additional directories (besides those in $PATH) to be searched for rescue disk files. Directories inside the list must be separated by commas. You don't need a trailing slash on these directory names. Any directories you list here will be searched BEFORE those in $PATH.

D Using a loopback device

A loopback device allows a normal disk file to be mounted as a filesystem. Depending on how much physical memory you have, using a loopback device may be preferable to using /dev/ram0 for building a root filesystem with Yard.

To use a loopback device you'll need to do the following:[4]

1. Enable loopback device support in your kernel if you haven't already. Under *Floppy, IDE and other block devices*, select either Y or M for *Loopback device support*. Recompile your kernel and reboot.

2. Check your

   mount

   manpage to see if it supports loopback devices. If it doesn't, you'll need modified versions of mount and losetup. These are available from:

   `<ftp://ftp.win.tue.nl:/pub/linux/util/>`

   in the file `mount-2.5X.tar.gz`, where X is the latest version letter.

3. You'll also probably need mke2fs 1.02 or later. Check the manpage for `mke2fs`; if there is no "-F" option listed, you'll need a newer version. mke2fs is included in the e2fsprogs package available in:

[4] Thanks to Roderich Schupp for some of this information.

```
sunsite.unc.edu:/pub/Linux/system/Filesystems/ext2/
```

To use the loopback device/file with Yard, simply:

1. In Config.pl, set $device to the name of a temporary disk file to hold the disk image, eg `/tmp/fsfile`. The file need not already exist. Set $fs_size to its desired size.
2. Create the $device file:

   ```
   create_loopback_file
   ```

   or do it yourself with:

   ```
   dd if=/dev/zero of=$device bs=1k count=$fs_size
   ```

   and substitute your values for $device and $fs_size.
3. If necessary, load the loopback device module (via `insmod loop`) into your kernel.

Yard may then be run with no further alterations.

Warning: When using a loopback device, occasionally the ext2 filesystem seems to get corrupted for no apparent reason. If you start getting errors in the middle of `make_root_fs` from sys() complaining that a directory doesn't exist, this may be what is happening. I don't know why this happens and I can't reproduce it consistently; however, unmounting, deleting and re-creating the file seems to get rid of the problem.