init4boot

Requirements & Design

Version 11

Status: Beta

July 2, 2008

http://sourceforge.net/projects/init4boot/

Table of Contents

Requirements

1 Note About Status

This document has beta status. That means, that mostly everything may change.

2 Features

init4boot brings mostly everything that similar packages – like initramfs-tools or yaird – provide, but it adds the following functionality:

- it's extensible: to add functionality, only a small python module is needed.
- boot with init script generated by **init4boot** is faster, because
  - everything that can be handled is handled at script creation time – and not during boot time.
  - it drops compatibility to existing kernel parameters, e.g. an `ip=dhcp` in the initramfs-tools environment does the whole network device initialization twice: one time during kernel initialization, one time during initramfs network initialization.
- it's portable and the resulting initramfs can be used for mostly all systems.
- there is no data collected and used from the build-host system.
- all configuration is done with command line parameters and the dhcp rootpath parameter.
- iSCSI boot is supported.
- multipath is supported – also during boot time.
- boot as a Xen guest system is supported.

3 Name

The package is named **init4boot**, because it creates a init environment during the initial boot from initramfs.

4 Alternatives

There are a couple of other tools and packages, that do similar things:

- initramfs-tools from the mainline Debian distribution [Debian initramfs-tools].
- yarid from the mainline Debian distribution [Debian yarid].
- RedHat supports iSCSI boot starting with Release 5.1. To configure this, the initramfs must be extraced by hand, the 'configuration' must be

changed, i.e. the init script must be edited. Afterwards the initramfs must be recreated by hand. RedHat uses iscsistart and does not support multipath or multiple targets. [RedHat iSCSI Boot]

5 Introduction

Nowadays mostly all Linux® distributions use some kind of initial ram disk to set up the system. The Linux® kernel itself has some basic command line handling, e.g. for setting up some network and using NFS as root file system. But when it comes to more complicated things, like using LVM, md devices, multipath, udev and so on, it is more convenient to use a mini system environment to set up everything. During this phase programs like `busybox` or `ipconfig` are typically used.

The main aim for this project is, to get iSCSI boot running. This is not integrated in the current initramfs creation tools – and it looks, that it is complicated to extend an existing tool set to get the needed functionality. Also it looks, that the current initramfs-tools maintainers are not very interested in integrating iSCSI in the main-line tool set, because there was no response at all to a post in Feb 2008 [iSCSI Boot post]. So it was decided to start a new project to get things fixed.

6 Definitions

6.1 Wording

The key words **must**, **must not**, **required**, **shall**, **shall not**, **should**, **should not**, **recommended**, **may**, and **optional** in this document are to be interpreted as described in [RFC 2119].

6.2 Boot type

The *boot type* specifies the main way to get the root device. This is typically one of:

- *local* for local disks
- *iscsi* for root devices using the iSCSI protocol
- *nfs* for NFS mounted root devices

Note that this list might be extended.

Status Note: not only boot types might be supported during initial phase of the project.

6.3 Boot variant

The *boot variant* defines some minor aspect of the boot. Currently available boot variants are:

- network
- udev
- multipath

A boot variant itself is an aspect of the boot type.

Example: iSCSI boot type can be done with or without *multipath* boot variant.

Note that some boot variants might depend on others.

Example: iSCSI boot type (automatically) depends on boot variant *network*.

7 Requirements

7.1 General

Requirement #1: Create bootable initramfs

init4boot must provide a command line tool for creating a bootable initramfs.

Requirement #2: Debian packages on stable

When providing Debian packages, these must be build on the stable distribution.

Note: This minimizes the install dependencies.

7.2 initramfs

This chapter summaries the requirements for the initramfs that is created with the command line tool.

Requirement #3: General

The initramfs **must** be of general use for all possible setups that are supported by the kernel and operating system.

Note: This might be hard to test. One thing that follows, is that (mostly) all modules from the system are put into the initramfs.

7.3 init

Requirement #4: init as a shell script

The init script(s) **must** be a shell script executed by the /bin/sh (which is typically linked to busybox).

Note: That means, that (only) the busybox functionality must be used in the scripts – and not maybe some advances bash functionality.

Requirement #5: Efficient

The created init script(s) **must** be as efficient as it makes sense.

Note: This is not a real-fact hard requirement, this is just the remark, that everything that can be done during initramfs / init script creation must be done then, e.g. handling dependencies of the different modules.

7.3.1 Phases and Stages

This sections contains the list of all phases that are used during the boot process. Each module can be called at each phase and can adapt the script for it's needs. Note that the generic real actions described here are done in the Generic module.

Requirement #6: Phases

The following phases **must** be supported during boot:

- Intro
- FunctionDefinition
- CommandLineParsing
- CommandLineVerbose
- InitialSystemSetup
- CommandLineEvaluation
- HandleInitialModuleSetup
- SetupLowLevelTransport
- SetupHighLevelTransport
- SetupDiskDevices
- MountRoot
- PrepareRootDir
- CheckForInit
- RunInit

Requirement #7: Stages

Each phase **must** consists of the following stages:

- prepare
- pre_output
- output

- post_output
- cleanup

Note: Each of these stages create some special things in the init script. The stages are described first, because the phase definitions need the knowledge of the stages at some points.

7.3.1.1 Stages

Requirement #8: Stages are optional

All stages of a phase **must** be optional.

Note: There is no need, that every phase and every module must implement all stages.

Requirement #9: Stage prepare

During the prepare stage everything, that is special for this phase **must** be set up and / or initialized.

Note: Typical things done during these stages are setting up and initializing variables.

Requirement #10: Stage pre_output

During the pre_output stage, the loop start constructs **must** be set up.

Requirement #11: Stage pre_output only for Generic Module

Modules apart from the Generic module **should not** use the pre_output stage.

Note: Because here normally things like `for x in `cat /proc/cmdline`; do` are written out, and there should normally not the need, that other modules do such generic things.

Requirement #12: Stage output

During the output stage, all the real work statements **must** be written to the init script.

Note: This is the real working phase.

Requirement #13: Stage post_output

During the post_output stage, the loop end statement **must** be written.

Note: Typically something like done is written here.

Requirement #14: Stage post_output only for Generic Module

Modules apart from the Generic module **should not** use the post_output stage.

Note: Because this closes what was opened during the pre_output stage, there will be some problems, if many different modules use some loop constructs.

Requirement #15: Stage cleanup

During the cleanup stage, all that was set and is not needed any more **must** be unset or deleted.

Note: This stage typically holds things like `unset var` or `rm /tmp/somefile`.

7.3.1.2 Phases

Requirement #16: Phase Intro

During the Intro phase, the following things **must** be handled in the init script:

- Writing the header `#!/bin/sh` to the first line
- Add a comment that the script is automatically generated and should not be changed.
- Add the time of creation, name and version of the creation tool.

Requirement #17: Phase FunctionDefinition

During this phase, all needed and used functions **must** be written to the init script.

Requirement #18: Phase CommandLineParsing

During this phase, the command line parsing **must** be written to the init script.

Note: Parsing here means only parsing. The only thing that can and should be done, is setting some variables. During the pre_output stage, the following is emitted from the Generic module:

```
for x in $(cat /proc/cmdline); do
  case ${x} in
```

This makes it possible that the output stage of other modules can handle

special command line parameters.

The post_output stage closes the case with a default rule, that prints out that the given parameter is not supported. It also closes the for loop.

Requirement #19: Phase CommandLineVerbose

During this phase debug and verbose parameters **must** be evaluated and corresponding things must be set up.

Note: This is generally only implemented in the Generic module, which handles the verbose and debug parameters.

Requirement #20: Phase InitialSystemSetup

During this phase the needed system environment **must** be set up, like creating and setting up /dev, /sys, /proc, /tmp.

Requirement #21: Phase CommandLineEvaluation

During this phase the command line parameters **must** be evaluated.

Note: The evaluation should use the command line parameter evaluation from the CommandLineParsing phase.

Requirement #22: Phase HandleInitialModuleSetup

During this phase the module set up **must** be done.

Note: When a boot type or boot variant needs a special module, the loading must be done at this point.

Requirement #23: Phase SetupLowLevelTransport

During this phase all the low level transport set up **must** be done.

Note: Here the low-level set up means the physical devices (like FC device or network devices).

Requirement #24: Phase SetupHighLevelTransport

During this phase all the high level transport set up **must** be done.

Note: Here the high-level set up means the logical transport protocol set up, like logging in a SCSI target.

Requirement #25: Phase SetupDiskDevices

During this phase all disk device set up **must** be done.

Note: This means that at the end of this phase there must be a mountable root device available.

Requirement #26: Phase MountRoot

During this phase the root disk device **must** be mounted under ***/root***.

Note: This is typically done with the a mount command from the Generic module.

Requirement #27: Phase PrepareRootDir

During this phase the freshly mounted root directory **must** be adapted.

Note: Typically some boot types and variants need some special set up of the root directory.

Requirement #28: Phase CheckForInit

During this phase the correct init executable **must** be searched.

Note: Typically this is ***/(s)bin/init*** or (when specified) ***/bin/bash***.

Requirement #29: Phase RunInit

During this phase the real init **must** be executed in the new root environment.

Note: This is normally done only by the Generic module.

Note: Everything that is output in the post_output and cleanup phase is typically not executed – except, when the new init process cannot be executed or ends.

7.3.2 Modular

Requirement #30: Program must be modular

The program to create the initramfs with **must** be modular in the way that additional functionality can be added without changing or breaking any existing functionality.

Note: This is very important. This means also, that the **init4boot** is extensible.

7.3.3 Dependencies

Some modules and functionality need other modules and functionality. To get the order right, each module can depend (for each phase) on other modules.

Requirement #31: Dependencies

The program to create the initramfs with, **must** be handle the given dependencies.

Requirement #32: Specification of Dependencies

Dependencies **must** be specified as a list of dependent modules.

Requirement #33: Granularity of Dependencies

It **must** be possible to specify a dependency for each phase.

7.3.4 Command Line Parameters

The command line parameters are reached to the init script via `/proc/cmdline` from the kernel.

There are some general command line parameters that are handled for mostly all modules – these are described in this section.

There are also command line parameters, that are specific to boot types or boot variants. These are described further down in the specific section.

Requirement #34: Exclusion of existing kernel parameters

The command line parameters which are interpreted from the kernel **must not** be (re-)used.

Note: This mostly gives problems.

Example: When using `ip=dhcp`, the kernel itself tries some heuristic to get some network interfaces up and running. But this can take much time. When using about six Ethernet interfaces, all that cannot (and maybe should not) initialized every single interface configuration runs into a timeout. Therefore, for configuration of network, the parameter `nw=` is chosen.

Requirement #35: rfs: Root File System

The boot type **must** be configured with the parameter `rfs`.

Requirement #36: rfs: Syntax

The rfs parameter **must** follow the syntax:

`rfs=<boot type>:<additional parameters>`

Requirement #37: rfs: Boot Type

The `<boot type>` **must** be one of `iscsi`, `local` or `nfs`.

Note: The semantics of this should be clear.

Requirement #38: Additional Parameters

The `<additional parameters>` **must** be given as a list of `key=value` pairs, separated with a semicolon.

Example: The following is a syntactic correct additional parameters list:

```
fast=yes;use_fb=no;serverip=10.1.1.3
```

Requirement #39: Boot Variant

The boot variants **must** be specified as a comma separated list with the parameter `bv=`.

Note: A full list of boot variants can be found in the next sections.

Example: `bv=network,udev`

Requirement #40: Host ID

It **must** be possible to specify a unique id with the parameter `hostid=`.

Note: This id is used in some boot scenarios. When this is used is documented in the next sections.

7.3.5 Boot types

7.3.5.1 Generic

7.3.5.2 local

There is mostly no difference between booting from a local disk or booting from special Fiber Channel hardware: The things described in this section, are also valid for the HBA based Fiber Channel setup.

Requirement #41: local: Generic

It **must** be possible to boot from a local hard disk, connected by (S)ATA, SCSI, USB or IEEE1934 interface.

Requirement #42: local: path

The path to the root disk that **must** be specified with the additional parameter ***path=*** **must** be used as the root disk.

7.3.5.3 nfs

Note: There is currently no support for NFS based root disk.

7.3.5.4 iscsi

Requirement #43: iscsi: Generic

It **must** be possible to boot from an iSCSI target.

Requirement #44: iscsi: Parameter localiqn

The local IQN (i. e. the IQN of the client) **must** be specified with the additional parameter ***localiqn=***.

Requirement #45: iscsi: Parameter portals

All iSCSI target portals **must** be specified as a comma separated list with the additional parameter ***portals=***.

Requirement #46: iscsi: path

The path to the root disk that **must** be specified with the additional parameter ***path=*** **must** be used as the root disk.

Example for the rfs parameter for iSCSI:

```
rfs=iscsi:localiqn=iqn.2008-02.org.flonatel:00031;portals=192
.168.228.20,192.168.228.21;path=/dev/mapper/iboot-root1
```

7.3.6 Boot variants

7.3.6.1 udev

Requirement #47: udev: Generic

When the boot variant udev is specified, the Linux udev system **must** be initialized during boot.

7.3.6.2 multipath

Requirement #48: multipath: Generic

When the boot variant multipath is specified, the Linux multipath system **must** be initialized during boot.

Note: That of course implies, that the root disk can be accesses with multipath.

7.3.6.3 LVM

LVM is currently not supported.

7.3.6.4 network

Requirement #49: network: Generic

When the boot variant network is specified, the specified network devices **must** be initialized during boot.

Requirement #50: network: Parameter nw

The network initialization data **must** be specified with the ***nw=*** parameter.

Requirement #51: network: Syntax of nw Parameter

The ***nw=*** parameter **must** contain a comma separated list of ***<nwif>:<init params>***, where <nwif> is the name of the network interface and <init params> are the parameters to initialize the network interface.

7.3.6.4.1 DHCP

Requirement #52: network: DHCP

When the <init params> is 'dhcp', then the given network device must be initialized with the help of DHCP.

Example:

```
nw=eth3:dhcp,eth5:dhcp
```

7.3.6.4.2 static

Note: Static IP address assignment is currently not supported.

7.3.6.5 tftp

Under some circumstances, there is the need to have much more configuration data that the kernel command line allows. Currently (for Xen kernels a la 2.6.18), there is a maximum of 256, for newer kernels there is a maximum of 4096 characters.

To get around this limitation there is the possibility to use the tftp boot variant. This tries to get some data from an tftp server, unpacks it and tries to execute a shell script – when provided.

Note that there is one major disadvantage when using tftp: there is no access control. So mostly everybody can read all the filed that are placed on a tftp server. In data center environments, some mechanism using encryption and / or authentication (like sftp) should be used instead.

Requirement #53: tftp: Dependency to Network Boot Variant

When the tftp boot variant is enabled, the network boot variant **must** be also enabled.

Note: This dependency is normally checked and handled during the CommandLineEvaluation phase.

Requirement #54: tftp: Command Line Parameters

When the tftp boot variant is enabled, the command line parameter *tftp=* **must** hold a comma separated list of IP addresses of the tftp servers.

Requirement #55: tftp: File Name

When the tftp boot variant is enabled, the file named ***<hostid>.tar*** **must** be downloaded from a tftp server. As long as it is not possible to download the file, all given tftp servers **must** be tried.

Note: Each tftp server must be tried mostly once.

Requirement #56: tftp: File Format

The file of the tftp downloaded file **must** be a tar archive.

Requirement #57: tftp: File Extraction

The tftp downloaded file **must** be extracted in the initramfs root directory.

Note: So it is very easy to store some configuration files in the tar archive, e.g. something like ***/etc/multipath.conf*** or ***/etc/iscsi/iscsid.conf***.

Requirement #58: tftp: Execution of Shell Script

When after the extraction of the tftp downloaded file there exists a file ***/tftp.sh*** and this file is executable, this file **must** be sourced by the init script.

Note: This is sourced, so that it is possible to set shell variables.

Example of the command line parameters for tftp server:

```
tftp=192.168.228.24,192.168.228.25
```

8 Design decisions

8.1 Python

Python was used to create the initramfs and init script, because it's simple, it's short, it's fast and it works.

9 Implementation Details

9.1 Shared Library handling

Because there is the need that

- one initramfs should be usable mostly everywhere, and
- it must be possible to created the initramfs on every system,

it is not possible

- execute any program,
- to call ldd on a program,
- to make any assumptions about the modules that must be loaded.

Because of this, a new method was developed to get the dependencies correct. Instead of using the ldd command (which does nothing more than setting some ld.so environment variables and running the binary), objdump is used. This has the advantage, that objdump is able to read mostly all kind of binaries.

Currently all the modules that are available are also copied into the initramfs.

This leads to a somewhat large initramfs – something between 20 and 30 MByte. But: therefore there needs only be one. In a data centre environment this makes sense: there is no need to fiddle around with creating, unpacking and repacking the initramfs. For each kernel version (and OS release) there is only one initramfs. This saves management time and some disk space – compared with some hundreds of individual initramfs images.

10 Testing

The following types, architectures and variants influence the boot process. To have a full regression test for all of the systems, about 75.000.000[1] different system setups are needed – for each distribution.

Dependencies:

- Architecture: amd64, arm, armeb, alpha, hppa, i386, ia64, m68k, mips, mipsel, powerpc, s390, sh, sparc
- Virtualization: Bare Metal, Xen dom0, Xen domU
- Multipath: on, off
- udev: on, off? (Are there systems out there without any udev these times?)
- Boot type: iSCSI, local, nfs
- LVM: on, off
- Using disk id: off, label, uuid
- EVMS: on, off
- MD: on, off
- dm-crypt: on, off
- lurks: on, off (root fs encryption)
- loop-aes: on, off
- Transport Interface: SCSI, SATA, USB, IEEE1394, CCISS, ida, network
- dmraid: on, off
- usplash / splashy: on, off
- ? cramfs initrd: on, off
- USB keyboard: on, off
- ? uswsusp: on, off
- ? swsusp: on, off
- ? swsusp2: on, off

10.1 Tested systems

Init4boot was tested on the following systems and everything works fine (as expected):

Distribution	Debian	*Version*	4
Architecture	amd64	*Virtualization*	domU
multipath	on	*Boot Type*	iSCSI

1 Exact number is 74317824, which is 2^15*14*3*3*6*3.

udev	on	*lvm*	off
disk id	off	*evms*	off
md	off	*dm-crypt*	off
lurks	off	*loop-aes*	off
interface	network	*dmraid*	off
(u)splash(y)	off	*USB keyboard*	off

ISCSI devices tested: Linux IET, EMC AX150i

Also tested: some additional disks (not root) with LVM

Distribution	Debian	*Version*	4
Architecture	i386	*Virtualization*	bare metal
multipath	off	*Boot Type*	local
udev	on	*lvm*	off
disk id	off	*evms*	off
md	off	*dm-crypt*	off
lurks	off	*loop-aes*	off
interface	SATA	*dmraid*	off
(u)splash(y)	off	*USB keyboard*	off

Distribution	Fedora	*Version*	9
Architecture	i386	*Virtualization*	domU
multipath	on	*Boot Type*	iSCSI
udev	on	*lvm*	off
disk id	off	*evms*	off
md	off	*dm-crypt*	off
lurks	off	*loop-aes*	off
interface	network	*dmraid*	off
(u)splash(y)	off	*USB keyboard*	off

ISCSI devices tested: Linux IET

11 Releases / Versions / History

11.1 Releases

11.1.1 0.0.1 [2008-03-31]

Release 0.0.1 is the initial release. This works on Xen Guest with iSCSI boot and multipath.

11.1.2 0.0.2 [2008-04-04]

Beginning with release 0.0.2, **init4boot** supports booting from bare hardware. Also the size of the initramfs was dramatically reduced: only the used libraries (and not all) are now copied into the initramfs.

11.1.3 0.1 [2008-04-18]

This is a consolidation release. No new functionality was added. Man pages were added and the documentation was finished.

11.1.4 0.2 [2008-05-03]

This is a consolidation release:

- Add additional package for iSCSI boot (which kills the original iscsid and starts a new one)
- Build the package on debian stable (also?)

11.1.5 0.3 [2008-07-02]

Port to Fedora Core 9. The initial development was done on and for Deb

11.1.6 Short term aims

During the next releases, the following must be done:

- Backport to Debian :-) [The port to Fedora might have negative implications to the Debian systems.]
- Check (and maybe fix): Xen dom0, nfs, lvm
- Add: encrypted tftp / sftp (similar to tftp)

11.1.7 Aims to get to stable state

The stable state of the project will be reached, if the first normally used system has an uptime for at least one year and no problem occurred.

This can be earliest reached at 2009-05-04.

Additional there must be at least 100 users using this and 1.000.000 successful boots without any problems.

11.1.8 Long term aims

One aim is to have at least the same functionality and stability as similar packages (initramfs-tools, yaird).

Ports to SuSe and RedHat.

Official package in Debian, Redhat and SuSe.

12 Open Points / ToDo

- Must a fsck done before a mount of the root device? [It looks that Fedora and Debian have different philosophies here: in Debian it is not needed – the normal startup scripts will handle this. Under Fedora it might be that the root partition must be checked before mounting.]
- Add to doc: No program is executed - because it is possible that the system is different.
- Script can be executed by a normal user – no need to be root.
- Possible to use current system or some arbitrary other system.
- Add 'sftp' (which used keys and maybe a password provided with a command line parameter) instead of the no-security-at-all tftp.
- Add status change of documentation to beta: aims?

13 Bibliography

RFC 2119: *Bradner, S*, ***Key words for use in RFCs to Indicate Requirement Levels***, 1997, http://www.ietf.org/rfc/rfc2119.txt

iSCSI Boot post: *Florath, Andreas*, ***iSCSI Boot***, 2008, http://lists.debian.org/debian-kernel/2008/02/msg00719.html

RedHat iSCSI Boot: *RedHat*, ***Installation-Related Notes***, , http://www.redhat.com/docs/manuals/enterprise/RHEL-5-manual/release-notes/RELEASE-NOTES-U1-s390x-en.html

Debian initramfs-tools: *unknown*, ***initramfs-tools***, , http://packages.debian.org/source/sid/initramfs-tools

Debian yarid: *unknown*, ***yarid***, , http://yaird.alioth.debian.org/