SlimLinux

Linux for IoT and
embedded applications

A Guide

Table of Contents

What is SlimLinux

SlimLinux is a Linux server development kit for the purpose of developing small lightweight, secure Linux based network-aware embedded applications – both physical & virtual. The base installation takes 16Mb of disk space and uses 18.5Mb of RAM to run.

An “embedded application” means any Linux system that is intended to run “headless” (no standard UI) with almost exclusively unattended operation. So this includes, but is not limited to, IoT, network applications (Routers, Firewalls etc), Internet servers (real or VM), industrial applications, automotive applications, intelligent devices, home & industrial automation, basically anywhere you want to be able to put in a relatively sophisticated network capable computer that will run continuously without the need for human intervention.

SlimLinux is a hybrid distribution. It is an operating system design that uses other Linux distributions as its base, but then adds a range of open source packages compiled (from source) specifically for SlimLinux. This means you can copy binaries and libraries directly from the base distribution and get them to run on your SlimLinux.

We compile certain packages from source to ensure the SlimLinux scripts & configurations work unchanged on all binary platforms, and all we have to replace is are the binaries & libraries.

Because SlimLinux is a very cut-down operating system, if you do copy binaries from other distributions, you may also need to copy libraries and static configuration files that are required to support the binary you have copied over.

Our design philosophy has been to copy over as little as possible. All you need for a minimal Linux system is a kernel, some basic libs, and busybox – so that's about all you get in our base O/S.

Current binary base distributions supported are

- Slackware x64 v14.2
- Redhat Enterprise Linux x64, v7.5
- SUSE Enterprise Linux x64, v12 SP3

When you look at the “Redhat”, “SUSE” or “Slackware” SlimLinux what you will **not** see is an operating system that looks totally like the one it is based on – although standard directory naming conventions are used (e.g. “/etc”, “/lib64”, “/sbin” etc).

The base gives the SlimLinux full binary compatibility with the base you selected, but the structure of the operating system and what utilities are available will be different.

We have been running public facing internet server applications, based on Slackware/SlimLinux, on both physical & virtual servers, for over 10 years and would expect an “uptime” of between 3 & 4 years – i.e. running 3 to 4 years between reboots.

Because SlimLinux is so small, it enables you to run each of your server applications on a different physical or virtual machine – a technique called “network appliance” – this improves security without being as wasteful of resources as using a full standard install.

Differences between different Binary Bases

There is only one major difference between the RedHat/SlimLinux or SUSE/SlimLinux and the Slackware/SlimLinux and that is, the RedHat & SUSE have support for PAM authentication.

This means the SUSE & RedHat/SlimLinux has all the libraries & configuration files needed for a basic PAM to work, although we have only tested the "pam_unix" module. If you want other modules to work, you can just copy them over from a standard install of RedHat or SUSE. Although you may also need to copy over additional dynamically loaded libraries as well.

If you are having problems getting your additional authentication method to work, try starting the authentication process then, in a separate window, run an "`strace`" on the authentication program and see if it is looking for a library that is missing.

To get PAM support for console login, we have also had to enable PAM authentication in "`busybox`" (`CONFIG_PAM=y`) - where as the Slackware "`busybox`" binary has it disabled – this is the only difference in the way "`busybox`" was built between the two platforms.

The additional code required to support PAM means that the base SUSE & RedHat/SlimLinux is about 2Mb larger than the Slackware/SlimLinux.

License Terms

SlimLinux is released as open source under the license GPLv3.

You are free to do whatever you like with the software, but all modifications, enhancements, bug fixes to any part of the works must be made public as well as any additional features / binary bases / overlays that depend on, or make use of, the open source code.

All projects that are based on SlimLinux, or use any parts of SlimLinux, must be fully open source and made free under GPLv3.

If you wish to use SlimLinux as part of a proprietary product, and so do not wish to publish the source code to that product, other license terms are available. Closed source projects based on SlimLinux are not permitted under the terms of this open source license.

Developers that submit changes, updates, enhancements etc, agree that their code will always be released as open source under GPLv3, as well as being included in alternative closed source license terms offered to third parties.

We totally respect all developers who do not wish to agree to this but, due to pre-existing agreements, we can not integrate their code in the Primary branch.

This is commonly referred to “selling exceptions” and there are mixed feelings about it in the open source community. However, where it enables businesses to sustain open source projects it will probably continue to be part of the industry.

Getting Started

```
$ git clone https://github.com/james-stevens/SlimLinux.git
$ cd SlimLinux
$ ./configure
$ make
```

This will create a bootable ISO image that, when booted in a target machine, will install the SlinLinux with the options you have chosen. Pre-built ISO images with all option packs and no option packs can be downloaded from our website – www.slimlinux.net

If you are not sure what to do with an ISO image, I recommend you google "`boot iso image`".

The ISO image is "hybrid", which means it will boot from either a CD/DVD (real or virtual) or a USB stick.

If you change the "`mkiso.cfg`" file, or run "`./configure`" again, you then need to run :-

```
$ make clean all
```

The "`configure`" utility checks you have the programs "`mkisofs`" and "`isohybrid`" that are needed for creating the ISO. If you do not have these you will have to refer to your documentation to add them.

Or google "`add mkisofo <name>`" - where "<name>" is the name of the operating system you are using.

We have only tested the build scripts on a Linux system (using various distributions).

If you do not have a Linux system, but want to test out SlimLinux, we recommend you download one of the pre-built ISO images available from our website - www.slimlinux.net

Design

SlimLinux is designed using a basic set of libraries and binaries that is close to the minimum required to support a network-capable Linux server. The base system includes the server processes "sshd" and a range of basic operating system utilities, but not much else – so the total is about 16Mb. However, this still enough to build a useful firewall or NTP server!

The operating system utilities are largely provided by "busybox" (https://www.busybox.net/) and the kernel is an unpatched version of one of the most recent stable "longterm maintenance" development trees. The "config" file required to build the kernel and "busybox" are included in the "build_files" subdirectory.

Currently SlimLinux is available based on **Slackware x64 v14.2**, **Redhat Enterprise Linux x64 v7.5** and **SUSE Enterprise Linux x64 v12 service pack 3.**

Slackware is not widely know, but is incredibly reliable. A SlimLinux server based on Slackware should give you an uptime in the region of 3 to 4 years between reboots. Both sets of binaries use the same kernel (built using the provided "config") which is also the same kernel used for the CD boot/install.

On top of the "base" operating system you can add "overlays", these are extremely similar to packages provided by other distributions, but the intention is that they represent application groups, instead of program groups. For example, you might add a "LAMP" overlay, which includes Apache, MySQL & PHP - or a "Mail Server" overlay which includes an SMTP, POP3 and IMAP4 server.

One significant difference is that with SlimLinux you always build a custom install ISO that only includes those parts that you want. You can also include configuration & application data on the ISO that installed alongside the operating system.

Some of the binaries that are in SlimLinux are *not* a direct copy of the binary from the respective distribution, but the binary has been built fresh from its source code (the source code bundles used and their build options are listed in the "open_source/README" file). This has two advantages

1) It enables us to reduce the number of libraries the binary is dependant on, and so reduce the overall size of SlimLinux
2) We can ensure the same version is run whether you are using the RedHat, SUSE or Slackware binary base – this ensures all the scripts that control the system will be 100% compatible, no matter which binary base you choose.

We have tried to keep things as simple as possible. All binaries that may need to be run from the command like are in "/sbin" ("/bin" is just a symlink to "/sbin"). All libraries are in "/lib64". Binaries that do not need to run from the command line (like daemon processes), are in "/usr/sbin". "/etc/rc.d/boot.d/<files>" are the file run at start-up. "/usr/scripts" contains the operating system scripts.

The Control Process and Run Levels

There are many great and sophisticated "`init`" (process control) daemons, like "`systemd`", but its more complex than what we were looking for, where as the "`init`" in "`busybox`" was just too simple! So we've gone with the "`sysvinit`". Its only major drawback is that it does not support supervising process that fork into the background. This offers run levels 2 to 5 for managing system processes. Here's how we've used the different run levels.

Run Level 2	Console login only, unless that's been disabled, in which case only Ctrl-Alt-Del will work! The DHCP client also runs at this level
Run Level 3	"`sshd`", "`ntpd`" (optional) and "`crond`" running – so basic remote access
Run Level 4	Recommend for bringing up user processes, so you can test the system, but the system's external services are not running
Run Level 5	Full external services running

If the server has a fixed IP Address, its default run level will be 5.

If the server has a DHCP assigned IP Address, it will start at run level 2 until it has obtained an IP Address, then it will transition to run level 5.

All processes that run at a lower level will also (usually) be running at levels above that. This may not always be true, but it is usually.

Sysvinit's behaviour is specified by the file "`/etc/inittab`" and controlled by the command line utility "`telinit`". "`telinit`" is just a symbolic link to "`init`", so you can use "`init`" instead to change the live run levels.

If you are connected by "`ssh`" and you run "`init 2`" you will not loose your connection, but you will have terminated the "`sshd`" login daemon - so make sure you bring the server back to at least level-3 before logging out!

"`init 0`" will shut the system down and "`init 6`" will reboot it. The utilities "`halt`", "`shutdown`" & "`poweroff`" do the same as "`init 0`", and "`reboot`" does the same as "`init 6`". Once the system reaches run level-0 (`halt/shutdown`) it will always try and power-off.

You can also shut the system down and power it off by clicking the server's physical power button.

Default Accounts and Groups

There are two accounts in the default system configuration. They are "`root`", the standard Unix "super-user" account, and the user account "`slim`". Their passwords are "aa" (`root`) and "bb" (`slim`) – so we **strongly** recommend you change them.

The passwords are stored in "`/opt/config/system.cfg`" and are salted & hashed using SHA-512. You can use the command line utility "`mkpasswd`" to make a replacement password

```
$ mkpasswd my-new-pass
```

```
$6$F84D2y3aCCsVghNH$Yk/gUoTuDo8qRVv6N9brUDx8K8HhGXNtveGfzb/sDaqfIixgzvPb7DSO
60BKGLUG2P1PF/VgJbev9ZDfCcQhp1
```

"mkpasswd" will default to SHA-512, other hashes are available, but it would seem pointless to use them, unless you have a good reason.

The user "`slim`" is also in the groups "`user`" and "`sudo`". Being in the group "`sudo`" gives the user permission to run any command with "`root`" privileges using the "`sudo`" command.

The groups "root", "nogroup" are always created and the groups "users" and "sudo" will be created only if you have users in them.

Partition Structure

SlimLinux uses is own, very simple, partition structure. This starts with a basic design philosophy that divides an operating system's files into three types

1) Binaries, Libraries and static files, partition-1
2) Configuration files that rarely change, partition-2
3) Application data that may be constantly changing, partition-4

Each data type is put into a separate partition, with different "`mount`" options (set in "`/etc/fstab`"). There is also a swap partition (partition-3) which defaults to 8Gb.

This means the minimum disk size for a *default* install is about 10Gb. However, you can change the partition sizes by providing a file called "`sfdisk.cfg`" in the top level of the "`iso`" directory. The file is piped to "`sfdisk`" to partition the disk. The default is:-

```
label: gpt
,512000,L,*
,21000,L
,16700000,S
,,L
```

This makes Partition-1 (R00T) 250Mb, Partition-2 "`/opt/config`" 10Mb, Partition-3 (`swap`) 8Gb and Partition-4 "`/opt/data`" the remaining disk space.

If you are using "*server instance*" or "*server application*" overlays (see below), you can include an "`sfdisk.cfg`" file in the overlay directory to change the partitioning for that server.

When the partitions are formatted, they are formatted with the following "Labels"

Partition	Purpose	Format	Label
Partition 1	ROOT file system	Ext2, Read-Only	SlimRootFS
Partition 2	Configuration Data	Ext2, NoExec+Sync	SlimConfig
Partition 3	Swap	Swap	SlimSwap
Partition 4	User Data	RieserFS, NoExec	SlimData

The file system type of partition-4 can be easily changed by editing the script. However, you will also need to add the appropriate "fsck" utility to the operating system.

If you wish to have your data on a separate disk, you can do this by removing the "SlimData" label from the boot disk and adding it onto the partition you want to use for your data.

1 ... Binaries, Libraries and static configuration files

Binaries & Libraries are kept in the ROOT partition and it is mounted read-only & execute allowed. Therefore, if you wish to add files to it, you must remount it writeable, make the changes you want, then mount it read-only again.

We provide the user "`root`" with three helpful shell aliases to do this, they are

> "`rw`" - make the ROOT file system writeable, with write-cache disabled
> "`rwr`" - make the ROOT file system writeable, with write-cache enabled
> "`ro`" - make the ROOT file system read-only.

"`rw`" is useful if you are making a small change, e.g. editing a script file. "`rwr`" is useful if you are uploading a large number of binaries / libraries. You can upload a lot of files with write-cache disabled, but it will be a lot slower.

If returning the ROOT file system to read-only status fails, usually with a message saying its busy, this usually means you have replaced a binary / library that is currently running. The running process will be running the old copy, until you restart it, so the file-delete of the old copy is still pending, until the process is restarted. So just kill the process you have replaced and try again.

If you have replaced a library, you will need to kill all the processes that are using that library. If you can't get the ROOT file system to return to read-only, you can always just reboot.

If you add binaries from the standard distributions your SlimLinux is based on, you may also need to adjust their start-up options in order for them to be happy to run in a read-only ROOT partition. This is usually not too difficult and is both necessary and worth doing.

Having the ROOT partition read-only has two big advantages. Firstly, it will never become corrupt in the event of a crash – this vastly improves the likely hood your server will be able to restart and automatically recover. Secondly, it makes it more difficult for malicious agents to make changes.

By default the ROOT partition is formatted EXT2. Its does not need to be a journaling file systems as it does not change.

2 ... Configuration Data

Configuration data are the configuration files that differentiate one instance of the same server type from another and it is stored in "`/opt/config`". Two servers of the same type should have identical contents in their ROOT file system, unless changes have been made by hand.

This is partition 2 of the disk and is mounted with write-cache disabled and execute permission denied (`sync,noexec`). This means changes made to files in this partition are immediately written to the physical disk.

In a traditional operating system, these files are generally scattered all over the disk in various different directories & partitions. This can make it very difficult for a sysadmin to identify and back-up the system's config.

You can think of the contents of this directory as being the "personality" of an instance of the server. If you copy the entire contents of this directory from one machine, of the same type, to another, then reboot the target machine, it will have adopted the entire personality of the source machine. You can even switch the ROOT partition's binary base

and reboot, and the new machine will behave the same as the previous binary base.

Where, on a tradition distribution, the same configuration value may appear multiple times in different places, on SlimLinux the value appears once (usually in the "`system.cfg`" file) and any configuration files that need this value are than dynamically constructed. A good example might the the server's IP Address.

By default the config partition is 10Mb in size and formatted EXT2 – because there is no write-caching, it does not need to use a journaling format.

3 … Application Data

Application Data is stored in "`/opt/data`" which is partition four. It takes up all the rest of the disk (except for 8Gb of swap in partition 3), and is formatted ReiserFS. Everybody has their preferred filesystem – if you are not happy with ReiserFS, its not difficult to change. The kernel we provide also has built-in support for EXT4, but adding (say) XFS would be trivial.

"`/opt/data`" is mounted with write-cache enabled (for improved performance), execute-denied and "`noatime,nodiratime`" - very few applications make use of "`atime`", so the performance hit of having it is usually not worth it and by minimising changes to the filesystem metadata, you minimise the risk of catastrophic corruption in the event of a crash.

Building Your ISO

To build your bootable install ISO (which can be booted from either a CD / DVD or USB stick), you simply run "make". This will prompt you to create a configuration file called "mkiso.cfg" this has the following options

binary	The base operating system binary platform – either **slackware_x64_v14.2** or **rhel_x64_v7.5** – the directory "baseos/rootfs" will contain one directory for each possible base operating system, excluding "common". There is no default, so this option is mandatory.
overlays	A space separated list of overlays to add on top of the base operating system. You can see the list of possible ones in the directory "overlays". More overlays may be added in automatically, if there are dependencies. With no overlays, only the base O/S will be installed.
upload_ip	IP Address to upload the finished ISO to
upload_dir	Remote directory on the server identified by "upload_ip" to place the ISO image in.
kernel	The file name of an alternative kernel to use for both the operating system and install. The default is "kernel" The file you specify must be in the "kernel" subdirectory.

If either "upload_*" option is missing, the upload stage is skipped. The "upload" stage runs a "tar" over "ssh", so assumes you have "ssh" access to the target IP Address.

The ISO image is created in the subdirectory "output". All ISO images are "hybrid" - this means they can be booted from either a CD/DVD or a USB stick.

NOTE: because this builds a root file system for an operating system, it will need "root" access to be able to set ownership and permissions on some of the files. When this is the case it will tell you, and re-run a script under "sudo". This is unavoidable.

Two different ISOs

When you build the ISO image you are actually building three things

1) The root file system of the SlimLinux you are going to run – this is in the subdirectory "R00T"

2) An ISO image of an "install" CD – this designed to install the SlimLinux operating system to an internal hard disk of a target machine. As well as installing any configuration data & user data you have chosen to include.

3) An ISO image of a "LiveCD" - this is a copy of the SlimLinux operating system you have chosen to create that will boot directly from a CD/DVD or USB Stick. It will include any configuration data you have bundled in, but will not include any user data.

The "LiveCD" ISO vs the "install" ISO

The "install" ISO is a compressed copy of the operating system, as well as any configuration & user data you have added, which is installed onto an internal hard disk of a target machine.

When you boot the "install" ISO, it will run some basic hardware checks, then install onto the hard disk of the machine you have booted it in. If you boot from a CD/DVD with a USB stick (but no hard disk) plugged into the target machine, you should be able to install onto the USB Stick, as if it was a hard disk – then boot & run SlimLinux from the USB Stick. Installing USB Stick to USB Stick might work, but is not support (see "`bootusb`" option below).

After you have run the install, you can remove the CD/DVD/USB and run entirely from the internal hard disk.

The "LiveCD" is designed to boot SlimLinux from the CD/DVD/USB stick, loading the operating system and any optional configuration data into a RAM-Disk and then continue running from the RamDisk (called an "`initrd`").

Any changes made to the operating system, or configuration, in the RAM-Disk will be lost if the machine is rebooted.

Once the machine is booted & running you can remove the CD/DVD/USB but, if the machine is ever reset, it will then fail to boot.

If you include a configuration data set in you ISO, you could run an server (from a LiveCD) that requires no original data – e.g. a firewall, NTP Server or DNS Secondary. By not making any access to the "LiveCD" after booting the media should last a considerably long time. This is at the expense of using a little extra memory and having the O/S and config hard coded into the memory stick.

IMPORTANT: "LiveCD" Security Warning

The "LiveCD" stores the server keys for "sshd" in the "LiveCD" ISO image. They are created at the time the ISO image is created. So if you use the exact same "LiveCD" image for multiple servers, they will all have the same "`ssh`" keys – this is not a good idea.

If you want multiple "LiveCD" images to use on multiple servers, where they all have the same configuration (which is pretty unlikely), we **strongly** recommend you run a "`make clean all`" to make a new unique "LiveCD" ISO for each server.

The alternative would be for the "sshd" keys to change every time the server rebooted – this is also undesirable – partly because it means the keys could be constantly changing and partly because the amount of entropy the system will have at that time would be very limited, so the chances of the generated keys being exactly the same for everybody's "LiveCD" SlimLinux server is undesirably high.

Booting from a hard disk creates a little more entropy than booting into a RamDisk, so the issue is reduced. We have added in some of the latest optional kernel features to ensure the system generates as much entropy as possible. We also run an extra entropy daemon (haveged) to try and solve this issue – its not a perfect solution, but probably better than nothing.

Burning your ISO

Once you have built your ISO, you can use "make" to do a range of different things with it. These options all basically run the "burn" script passing the option as the only parameter.

dvd	Burn the "install" ISO to a CD/DVD – requires you have a CD-RW or DVD-RW drive in the machine you are running on.
usb	Copy the "install" ISO to a USB memory stick
livedvd	Burn the "LiveCD" ISO to a CD/DVD – requires you have a CD-RW or DVD-RW drive in the machine you are running on.
liveusb	Copy the "LiveCD" ISO to a USB memory stick
bootusb	Fully install bootable SlimLinux to a USB memory stick plugged into the machine you are running on. Also requires "root" permission, so will re-run itself under "sudo" if necessary.

If you use any of the "USB" options, we **strongly** recommend you temporarily disable any feature your system might have to automatically "mount" the memory stick when you plug it in. Otherwise the process can get a little confusing – although it should probably still work OK.

The "bootusb" option installs SlimLinux to a USB memory stick, as if the memory stick was a standard hard disk – i.e. the USB stick will be split into the four standard partitions with the normal partition "Labels". A "bootusb" stick should boot fine in a machine that also has internal hard disks, and the data on them will not be touched.

Once installed, you can plug the stick into a machine that has no hard disk and boot & run SlimLinux from the USB Stick as if it was a hard disk – assuming the machine has built-in support to boot from a USB Stick.

Because the "bootusb" is effectively treating the USB Stick as a hard disk, once SlimLinux has booted from the stick, you should shut it down before it is powered off, and only remove the memory stick after the machine has powered off.

If you boot the "LiveCD" image, either from a CD/DVD or from a USB stick, SlimLinux will not touch any of the machine's internal hard disks, but run entirely from a Ram-Disk – so any data you have on that machine will be safe.

If you boot the "LiveCD", there is no need to even shut the server down, you can just power it off, although the usually shutdown / reboot options are still available.

The option to burn a CD/DVD currently supports using either the graphical tool "xfburn" (tried first) or the command line tool "cdrecord" - if you do not have either of these installed, it will be unable to burn a CD/DVD.

If you can supply us with a command line to invoke your CD/DVD burning software, we would be more than happy to add it. The command you give us must take the name of the disk image as one of the options. For example …

```
$ cdrecord -tao <iso-image>
```

… or …

```
$ xfburn -i <iso-image>
```

Supported "make" Options

`clean`	Remove all files & directories that were created in the process of making the ISO, excluding the "`mkiso.cfg`"
`all / iso / default`	make anything that needs making to build the ISO images
`rootfs`	Recreate the SlimLinux ROOT file system image, config & data, then recreate the ISO images
`initrd`	Recreate the installation operating systems (run by booting the ISO), then recreate the ISO images
`config`	Remove and re-prompt for the configuration, then rebuild the ISO images

Note: the "`make`" is not really a "`makefile`" in the true sense – in that it does not automatically rebuild only what needs to be rebuilt when certain files change. It is simply a "`make`" interface to the "`mkiso`", "`burn`" & "`configure`" scripts, which are what do the real work.

If you change the "`mkiso.cfg`" file, the "`makefile`" will **not** correctly rebuild the ISO, you need to rebuilt it manually using "`make clean all`".

To install the SlimLinux operating system you have specified, you simply boot the ISO in the target machine. If the hard disk on the target machine is already partitioned, you will be asked to confirm you wish to repartition and reformat the disk.

At this prompt you can answer …

`Stop`	No repartitioning or formatting is done and the target machine will switch itself off.
`Continue`	The install will proceed as if the disk had been completely blank – the disk will be repartitioned and the partitions will be reformatted.
`Update`	Only the ROOT partition is reformatted and reinstalled – this replaces the kernel & operating system, but will **not** touch the config or data. This can be used to update the server to different code or to switch the server to a different binary base.

Once you have finished the install, the ISO will prompt you before ejecting the CD and rebooting. We **highly** recommend that you reboot before powering down, as this ensures all data is flushed to the disk.

By default the SlimLinux kernel includes the "`virtio`" drivers for Ethernet and SCSI hard disks and we have tested SlimLinux on physical hardware as well as running as a Linux/KVM guest. We have no reason to believe it will no run under other hyper-visors – although we can not guarantee all the required hardware drivers are present.

The Kernel is an unmodified standard kernel and we have provided the config file we used to create it. The main difference between the way we build a kernel and the way a standard distribution might is that we disable dynamically loadable modules and include all the drivers in the kernel. For more information see the Kernel section, below.

Build Tree's Directory Structure

The directory structure of the build environment is as follows

`baseos/rootfs/common`	Contains files that are common to all binary platforms – typically this means script files, static config files and symbolics links.
`baseos/rootfs/<platform>`	Contains the binaries & libraries that belong to the corresponding binary platform. If you wish to add your own binary platform, create a subdirectory here and place the files as if they were in the ROOT file system of a running system
`build_files`	Contains the config files we used to build the kernel & busybox as well as the list of files the `install_cd` operating system needs copied from the `baseos/rootfs` directory.
`install_cd`	Files & scripts to run the create the operating system for the installation ISO – becomes an "`initrd`"
`kernel`	The kernel for the install & operating system
`overlays/<name>`	One subdirectory for each overlay
`source`	The source code for minor utilities that are included in the installation process

There are also some scripts

`configure`	Prompts you to build your "`mkiso.cfg`"
`mkiso`	The script that reads the "`mkiso.cfg`" file and builds the bootable install ISO image
`Makefile`	A "`make`" interface to the options available in "`mkiso`"
`tstroot`	After you have built an ISO image, you will have a directory called "R00T", which is a copy of what your root file system will be if you run an install of the ISO. This script uses "`chroot`" to emulate running the "R00T" directory structure as a root file system. This allows you to test out development code without having to actually install it. This is especially useful for testing out library dependencies.

Overlay's Directory Structure

Within each overlay there is:-

`rootfs/common`	Directory containing the files & directories needed by this overlay that are common to all binary platforms
`rootfs/<platform>`	Directory containing the files & directories needed by this overlay that are exclusive to a particular binary platform
`config`	Directory containing files to go into "`/opt/config`"
`data`	Directory containing files to go into "`/opt/data`"
`<platform>.cfg`	Additional install options this overlay needs for a specific binary platform
`common.cfg`	Additional install options this overlay needs for all binary platforms

Currently supported additional install options, for "`common.cfg`" or "`<platform>.cfg`" are

`depends`	Space separated list of other overlays this overlay requires to be installed in order to work correctly. If the overlay is selected, the dependant overlays will be automatically added in.

You can also include any configuration options that are valid in the "`mkiso.cfg`" file. For example, this allows you to specify an alternative kernel for a specific overlay.

We have ensured that no one file is included more than once across all the overlays. This makes maintenance easier, as it only needs to be replaced in one place. However, it is perfectly valid for the same file to appear in the same place in multiple overlays.

The overlays are added-in in the order you list them in the "`mkiso.cfg`". If there are dependant overlays that are not included in "`mkiso.cfg`", they will be added to the end of the list you have given.

Please note: ALL files ending "`.cfg`" are treated as shell-script include files. Therefore, if you need to include special characters (including white space) in your configuration data, you must put the value in quotes.

Server Instance & Server Application Overlays

You can create your own custom overlays to build specific server instances – these are called *server instance overlays*.

This *can* include additional operating system files, but often all you will want is a custom configuration file and a list of overlays to include.

To achieve this all you need is a "`common.cfg`" (in the top level directory of your server instance overlay) which lists which overlays this server instances requires and a "`config/system.cfg`" which contains that server's specific configuration.

Then, in your "`mkiso.cfg`" the only overlay you need to specify is the server instance overlay and the "`mkiso`" script will recursively bring in all the overlays it needs.

If you have different configurations for different binary distributions you can rename the file "common.cfg" file as "`<binary>.cfg`" – e.g. "`slackware_x64_v14.2.cfg`" - for example, this allows you to specify different overlay dependencies for different binary platforms.

You can also create *server application overlays* that pull-in existing overlays, but add more functionality, e.g. by adding your own code and scripts. The server application overlay should also list all the pre-existing overlays it depends on.

You can then create *server instance overlays* that only needs to depend on one of your server application overlays in order to create an instance of that application server type.

For example, lets say I'm setting up two servers that are going to do web and email forwarding. I might create an *application* overlay called "`forwarder`" that depends on overlays like "`dns apache sendmail`" as well as adding a few scripts and custom binaries to make the whole thing hang together, but the application overlay would probably not contain any configuration data, but might contain some user data.

I could then create two server *instances* called "`fwd1`" and "`fwd2`" that both depend on the "`forwarder`" overlay, but also include configuration files that are specific to each server *instance* – e.g. static IP Addresses, host names etc.

So "`forwarder`" is the *server application overlay* and "`fwd1`" & "`fwd2`" are the *server instance overlays.*

Adding Your Own Overlay

To create your own overlay, simply create a subdirectory in the "`overlays`" directory and start adding files by copying them into the correct sub-structure from an installation of the appropriate binary base.

Before you build your ISO, remember to edit the "`mkiso.cfg`" file and add your overlay into the list of overlays to be included.

When the "`rootfs`" for the ISO is built, you will see a new directory called "`ROOT`" appear at the top level of the SlimLinux build tree. This is what will become the root partition of your SlimLinux. If you wish to test out your new overlay, you can use the "`tstroot`" script.

"`tstroot`" mounts ram-disks & virtual file systems (like "`/proc`" & "`/sys`"), as well as running some basic set-up scripts, then puts you into a "`chroot`" shell in the "`ROOT`" directory, so you can do as much testing as possible. Whatever is mounted, is then unmounted when you exit the shell.

If you are having real problems getting your binary to work, or if you want to use a binary from a different Linux distribution, and you have access to that binary's source code, a reasonably fool-proof way to get it working is to statically link the binary.

This means, so long as it doesn't dynamically load executable code at run time (like Apache's modules), all the libraries needed by your binary will be bound into the binary itself. This, of course, makes the binary a lot bigger, but may be useful to help your solve problems.

Adding Your Own Binary Base O/S

Adding a new binary base for Linux distributions that are based on a 64-bit PC architecture is relatively trivial. Just create a new directory in "`baseos/rootfs`" and start copying the files from an installation of that distribution.

You will probably find the utility "`ldd`" extremely useful for finding out what libraries also need to be brought across – but you can also refer to the binaries & libraries that are present in the existing binary bases.

Where we have listed a specific item of open source code, we strongly recommend that you build a fresh binary from that code, to ensure compatibility with all the existing scripts and config files.

You will also need to build, from the source code provided, binaries for the overlays you want to use.

Once you have built as much as you think you need, edit the "`mkiso.cfg`" file and change the "`binary=`" option to point to your new binary base, then run "`make rootfs`"

Once you have created a fresh "`R00T`" directory, you can also use the "`tstroot`" command to run some basic testing on your new binary base.

It is our intention to continue to focus on 64-bit platforms, with 64-bit ARM as one of our higher priorities.

Adding a new binary base that is not based on a PC architecture is going to be more complicated. We've not done it yet, so we'll let you know how it goes.

Replacing the Kernel

The kernel we selected (v4.14.x) was the latest version with "longterm maintenance" support - https://www.kernel.org/category/releases.html – it was first released in Nov-2017 and the "projected EOL" is Jan-2020

There are lots of reasons why you might want to use a different kernel, than the one we have provided, the most common might be you need a newer kernel with newer hardware drivers, or you may just want to enable a missing hardware driver in the current kernel.

Replacing the kernel with a newer one is relatively trivial.

1) Download and unpack the kernel you want to use
2) Copy the `build_files/kernel_config_x64_4.14.43B` file from the SlimLinux directory to your kernel's source tree as the file "`.config`"
3) Run "`make olddefconfig`" - this will read the kernel config values from the "`.config`" file you have copied over, and use standard default values if an option is missing.

If, at this point, you simply run "`make`" you should end up with a kernel that is pretty much 100% compatible with the one we have provided. If the kernel source code you downloaded is also v4.14.43, then the binary kernel you have created will be nearly identical to the one we provided.

If all you wanted to do was update to a newer release of v4.14, then that should be everything you need to do. If you wish to alter the kernel options, run "`make menuconfig`", do your changes, quite & save, then run "`make`" again.

Now copy your new kernel into the "`kernel`" subdirectory of the "SlimLinux" build tree, giving it a new name – e.g. "`mykernel`"

Edit the "`mkiso.cfg`" file to add a line specifying your new kernel – e.g. "`kernel=mykernel`". Now run "`make clean all`" in the SlimLinux build directory.

You can also follow this procedure if all you want to do is keep the same kernel version, but alter some of the configuration options. If you know what platform you are developing for, you can probably remove quite a few hardware drivers you do not need.

If you want to build a kernel completely from scratch, the you should be OK, so long as you ensure the following options are enabled – `CONFIG_DEVTMPFS`, `CONFIG_DEVTMPFS_MOUNT`, `CONFIG_TMPFS`, `CONFIG_BLK_DEV_INITRD` – and to boot from a CD you will need hardware support for the CD drive and `CONFIG_ISO9660_FS` enabled.

SlimLinux is designed to use a kernel with dynamic loadable modules disabled. This means all the drivers you need must be built into the kernel itself. On a general purpose distribution this can make the kernel excessively large but, because of the lightweight nature of SlimLinux this is generally not a problem.

This also means that any drivers that require dynamically loaded firmware will also need to have that firmware linked into the kernel. The most common drivers that require this are the Broadcom Ethernet chips used by Dell. Our default kernel does not include support for these chips. Adding it is not hard, but it is also not trivial!

Configuring a System

SlimLinux keeps all its configuration files in "`/opt/config`" - the core options are in the file "`system.cfg`", but many of the processes support you adding an individual custom configuration file into this directory, so you have complete control over how they run.

If you have not added a custom configuration file for a particular process, the system will usually build a useful basic configuration for you, which can often be controlled using options in the "`system.cfg`" file.

If you create a fresh install of the same overlays, copy the entire contents of this directory onto the new machine and reboot it, it will take over all aspects of the configuration of the original machine. There is nothing about the configuration of the server that is not kept in this directory.

When adding your own overlays, you should follow this principal.

Apart from "system.cfg" we also support the following other configuration files in "/opt/config" - if they do not exist, sensible defaults will be used instead.

`iptables.conf`	IPv4 Firewall in "iptables-save" format
`ip6tables.conf`	IPv6 Firewall in "ip6tables-save" format
`busybox.conf`	A custom "busybox.conf" file - default=empty
`snmpd.conf`	A custom config for the SNMP daemon
`zebra.conf`	A custom config for Quagga's "zebra" daemon
`bgpd.conf`	A custom config for Quagga's BGP daemon (supports Cisco router command format)
`dns.conf`	A custom config for ISC "bind" DNS Server

In addition, the system will store the server keys for "sshd" and a record of what overlays were originally installed.

Base Operating System Options

rootPassword	Password for the user "root", pre-encrypted (as seen in "/etc/shadow") – the default is "aa" (encrypted)
allowRootSSH	Boolean (Y/N) - Allow "root" to ssh directly to the server – if you disable this you will either have to login on the console (assuming that is not disabled) or login as an ordinary user, then use "su" to get root privileges.
allowConsoleLogin	Boolean (Y/N) - Run login prompts on the console (Alt-F2/3/4)
runningSSHd	Boolean (Y/N) - Add "sshd" into inittab – if you disable **both** allowConsoleLogin and runningSSHd you will have locked yourself out and you will have to boot the ISO to recover.
serverHostname	Host name for this server
serverAdmin	E-Mail address of the server admin
syslogToDisk	Boolean (Y/N) – syslog to disk (Y) or RAM (N) – syslog to RAM can be read with the command line utility "logread". Syslog to disk goes into "/var/log/messages".
syslogSize	File size to allow the syslog to grow to before rolling the log files
ntpServers	Space separated list of NTP servers to correct the system clock to – if this is present an "ntp" daemon will run, which you can also choose to have act as a server as well as a client by setting the "ntpAllow" option
ntpAllow	Space separated list of IPv4/v6 addresses or subnets which are allowed to access this as an NTP Server. We *strongly* recommend, if you enable this option, you make it as restrictive as possible.
rdateServers	Space separated list of rdate servers to correct the clock to
sshRemoteAllow	Space separated list of IPv4/v6 addresses or subnets which are allowed to "ssh" to the server
firewallLogging	Boolean (Y/N) – syslog firewall blocks. This should be enabled for debugging only, otherwise it can become an attack vector
keyboardMap	Console's keyboard layout – currently only "uk" or "us" supported
static4IP	<IP>/<netmask> (in bit count) of static IPv4 addresses to assign to the server. This option can appear multiple times. If not present, the server will run a DHCP client on "eth0"
static4GW	IPv4 default route / gateway
static6IP	<IP>/<netmask> (in bit count) of static IPv6 addresses to assign to the server – this option can appear multiple times
static6GW	IPv6 default route / gateway – if you are running "zebra" from the "bgp" overlay, it can obtain the IPv6 gateway from the LAN.
staticResolvers	Space separated list of IPs (v4 or v6) of DNS resolvers. If this is blank, sensible defaults will be used. If DHCP is running the resolvers will be taken from the DHCP.
extraUser	"<user>[:<group>[,<group>]] <password> <uid>" If no "uid" is present, one will be assign starting from 100. This has the potential to cause issues as it can mean a user's Id could change after a reboot. This may or may not matter to your applications.

	If <group> is specified, the GID will be the same as the UID. If a group is not specified, the group "users" is used. If a list of groups are specified, the user's primary group will be the first one – e.g. "slim:users,sudo" means the user "slim" will be in the groups "users" & "sudo", but its primary group will be "users". The <password> must be pre-encrypted. If the password field is just a hypen/dash "-", then this user can not login
staticRoute	"<ip>[/<mask>] <gateway>" - this option can appear more than once and specifies that the IP or subnet specified is routed via the gateway given. IPv4 & IPv6 are both supported.
pingAllow	Space separated list of IPv4/v6 addresses or subnets which are allowed to ping the server. If not present ,or left blank, then any IP Address can ping this server.
root_keys	SSH keys to put into the "root" user's ".ssh/authorized_keys" file
user_keys	SSH keys to put into every non-root user's ".ssh/authorized_keys" file
auth_keys	SSH keys to put into both "root" and non-root user's ".ssh/authorized_keys" file
<username>_keys	SSH keys to put into the specified user's ".ssh/authorized_keys" file – e.g. "dns_keys=..."

All "keys" options can occur more than once. Only users that have a password field, that is not dash/hyphen ("-"), will have a home directory, and so only those users can have the keys set up. If you want a user to be able to login, but **not** by password, only using ssh keys, then set the user's encrypted password to "x".

The system will try and keep the "system.cfg" file owned by "root" and permissions "600" (read & write by owner only).

The “snmp” Overlay

Adding the “snmp” over will allow you to do basic SNMP monitoring of your server. This can be useful as quite a wide range of monitoring systems support SNMP polling.

You can provide your own SNMP configuration file at “/opt/config/snmp.conf”. If you do not provide one, a basic configuration will be created for you.

The “snmpd” process will run at run levels 4 & 5.

Snmp Specific Options (in system.cfg)

runningSnmpd	Boolean (Y/N) – Add “snmpd” into inittab
serverFacility	Descriptive text of this server's Facility
serverLocDesc	Descriptive text of this server's physical location
serverLocation	Geographic co-ordinates of this server's location
snmpRemoteAllow	Space separated list of IPv4/v6 addresses or subnets which are allowed to poll this server over snmp

The “bgp” Overlay

The “bgp” overlay adds the “bgp” & “zebra” daemons from the quagga network package.

If you provide a BGP configuration file at “/opt/config/bgpd.conf”, this will be used, otherwise a basic BGP configuration file will be made for you. Because BGP can be quite complex to get right, it is highly likely that making your own BGP configuration file will be the best option.

Note – the “bgp” overlay requires a user called “quagga” - this should be specified as an “extraUser” in the “system.cfg” file. For example,

```
extraUser="quagga:quagga – 98"
```

This creates the user called “quagga” in the group “quagga”, both with the Id “98”, and no password – i.e. this user is unable to login – the user is for owning files and running processes only – its home directory will be “/tmp” and its shell will be “/sbin/false”.

The “bgpd” process will run at run level 5 only. The helper process called “zebra”, which communicates with the kernel, will run at levels 4 & 5.

Both “bgpd” and “zebra” support remote access via ports 2605 & 2601 respectively. “bgpd” config file and command line syntax is reasonably compatible with Cisco router commands. To get remote access, you simply “telnet” to the appropriate port and enter the password when prompted. As the connection is not encrypted, we recommend instead, you “ssh” to the server, then use “telnet 127.0.0.1 260[1|5]”.

With the addition of just the “bgp” overlay, SlimLinux can become a route server for an IX, and adding the “snmp” overlay would allow monitoring of its traffic flow.

“bgp” Specific Configuration Options (in system.cfg)

extraUser	“quagga:quagga – 98” - This extra user must not be removed for “bgp” to work correctly, although the user Id could be changed.
runningBGPd	Boolean (Y/N) – add “bgpd” & “zebra” into inittab
bgpPassword	Plain text password for read-only access to the configuration and status of “bgpd” & “zebra” on ports 260[1|5]
bgpEditPassword	Plain text password to upgrade to be able to modify the configuration of “bgpd” and “zebra”.
bgpRemoteAllow	Space separated list of IPv4/v6 addresses or subnets which are allowed to access port 2601 & 2605
bgpWithAS112	Set up BGP for running as an AS112 anycast node
bgpMyASN	AS number to use, if you are not using AS 112
bgpPeers	Peers to connect to by BGP – this makes the appropriate changes to the firewall and adds them into the “bgpd.conf” file.

The “dns” Overlay

The “dns” overlay adds ISC's “bind” dns server. This can be used a DNS Primary, DNS Secondary or recursive resolver. If you add the “bgp” Overlay, it can also be used as an anycast node, and has specific support for running as an AS112 anycast node.

It comes with the utilities “named-checkconf”, “names-checkzone”, “dig” and “rndc”. If you provide an RNDC key, rndc support will be added to the default configuration.

You can provide your own configuration as “/opt/config/dns.conf” or a reasonable one will be build using the configuration options provided in “system.cfg”.

Note – because of the security model under which the DNS server runs, if you have provided your own configuration file, when the DNS server starts up it will **copy** your file into its runtime directory structure. This means there are now **two** copies of your “dns.conf”, one that the server is running from and the original in “/opt/config/dns.conf”. If you make changes to the one the server is running from, make sure you copy those changes into the one in the config directory.

“dns” Specific Configuration Options (in system.cfg)

dnsLogging	Boolean (Y/N) – Enable / Disable logging of every DNS query – should be used for debugging only, or it can become an attack vector
dnsDNSSEC	Boolean (Y/N) - Enabled DNSSEC Support
dnsWithAS112	Boolean (Y/N) – add all AS112 zones into the DNS configuration
dnsResolver	Boolean (Y/N) – Enable this DNS server as a recursive resolver
dnsResolverAllowed	Space separated list of IPv4/v6 addresses / subnets which are allowed to use this recursive resolver (default = any)
firewallBlockRD1	Y/N or number – use the kernel firewall to block UDP DNS queries with RD bit on (RD=1). “Y” = block, “N”=don't block, number=allow this number of queries per second with RD=1 Authoritative DNS servers (Primary/Secondary) should **not** receive queries with RD=1, only resolvers. Sources listed in “dnsResolverAllowed” are not blocked or throttled.
FirewallBlockResp	Boolean (Y/N) – Use the kernel firewall to block UDP DNS responses sent to port 53. ISC's bind should **never** see incoming responses to port 53.
dnsRndcKey	Key to use for “rndc” authentication
dnsRndcAllow	Space separated list of IPv4/v6 addresses or subnets which are allowed to query the DNS server by “rndc”
dnsSecondary	“<zone> <ip> <ip>....” - Act as a Secondary DNS for this zone. Followed by any number of Primary IP (v4 or v6) addresses to retrieve the zone from. This option can appear multiple times.
dnsPrimary	“<zone> <ip> <ip>....” - Act as Primary for this zone, followed by a list of IP (v4 or v6) Addresses / Subnets allowed to XFR the zone. The default is to block all zone transfers. If “<ip>” is an IP Address, and not a subnet, it will also be explicitly sent a NOTIFY. This option can appear multiple times