Slackware Live Edition

Preface

Welcome to the Slackware Live Edition! This is a version of Slackware 14.2 (and newer), that can be run from a DVD or a USB stick. It is an ISO image meant to be a showcase of what Slackware is about. You get the default install, no custom packages or kernel, but with all the power of Slackware. The ISO is created from scratch using a Slackware package mirror, by the “liveslak” scripts.

Slackware Live Edition does not have to be installed to a computer hard drive (*however you do have that choice if you want to: using the `setup2hd` script*). You can carry the USB stick version with you in your pocket. You'll have a pre-configured Slackware OS up & running in a minute wherever you can get your hands on a computer with a USB port.

The USB version is “persistent” - meaning that the OS stores your updates on the USB stick. The CD/DVD versions (and the USB stick if you configure it accordingly) operate without persistence, which means that all the changes you make to the OS are lost when you reboot.

In order to protect your sensitive private data in case you lose your USB stick (or in case it gets stolen) you can enhance your persistent USB Live OS with an encrypted homedirectory and/or an encrypted persistence file, to be unlocked on boot with a passphrase that only you know.

Why yet another Slackware Live

The reasons I had for creating the Slackware Live Edition are as follows:

1. Provide a Live version of Slackware proper; i.e. show Slackware as it is, but without having to install it. No hiding of kernel messages scrolling across the screen at boot; no custom wallpapers, etcetera. Meant for educational, evaluation and demonstration purposes.
2. The target should be slackware-current, the bleeding edge. Many people want to know what Slackware's development edition looks like but are hesitant to install slackware-current for fear that it breaks stuff and causes productivity loss.
3. Provide a way to generate a Live ISO with just a mirror of Slackware's packages as the source, fully scripted and deterministic.
4. Still be able to customize its content. For instance provide stripped-down or minimalist versions of Slackware but also allow for the inclusion of 3rd party packages.
5. Option to create a bootable USB stick running Slackware Live (which is different from 'dd'-ing the hybrid ISO to a USB stick!)
6. KISS: Keep It Simple Stupid!

ISO variants

The “liveslak” scripts can generate a variety of Slackware flavors:

1. a complete 64bit Slackware-current Live Edition (in a 4.0 GB ISO);
2. a slimmed-down XFCE ISO (700 MB) with XDM as the graphical login manager. It fits on a

CDROM medium or a 1 GB USB stick;
3. a ISO image (4.3 GB) of Slackware64-current containing 'ktown' Plasma 5 instead of Slackware's KDE.
4. A Digital Audio Workstation (DAW) based on a custom Slackware package set plus a basic Plasma5, containing a rich software collection for musicians, producers and live performance artists.
5. a Mate variant (3.2 GB) where KDE 4 has been replaced by Mate (a Gnome 2 fork);
6. a Cinnamon flavour (a fork of the Gnome 3 Shell replacing Slackware's KDE 4).
7. a Dlackware variant, which is Gnome3 + PAM + systemd on top of Slackware and stripped of KDE4.
8. a StudioWare edition containing all the project's audio, video and photo editing software packages.
9. a *Custom* variant which you can give your own name, its own package list and custom post-install configuration.

Downloading ISO images

Common download locations are:

- Primary site: https://download.liveslak.org/ (rsync://liveslak.org/liveslak/)
- Darren's https://slackware.uk/liveslak/ (rsync://slackware.uk/liveslak)
- Willy's http://repo.ukdw.ac.id/slackware-live/

Enduser Documentation

Using the ISO image

The ISO images are hybrid, which means you can either burn them to DVD, or use 'dd' or 'cp' to copy the ISO to a USB stick. Both methods will give you a live environment which will allow you to make changes and seemingly “write them to disk”. The changes will actually be kept in a RAM disk, so a reboot will “reset” the live OS to its original default state. In other words, there is no persistence of data.

Slackware Live Edition knows two user accounts: “root” and “live”. They have passwords, and by default these are… you guessed: “root” and “live”. Also by default, the ISOs will boot into runlevel 4, i.e. you will get a graphical login. The bootloader allows you to pick a non-US language and/or keyboard layout and (on boot of an UEFI system) a custom timezone.

Slackware Live Edition deviates as little as possible from a regular Slackware boot. Once you have passed the initial Liveboot stage and brought up the actual OS, you login as user “live”. From that moment onwards, you are in a regular Slackware environment.

Booting the Live OS

BIOS boot

Slackware Live Edition uses syslinux to boot the Linux kernel on BIOS computers. To be precise, the “isolinux” variant is installed to the ISO image and the “extlinux” variant is installed into the Linux partition of the USB Live version.

Syslinux shows a graphical boot menu with a nice Slackware-themed background and several options:

- Start (SLACKWARE | KTOWN | XFCE | MATE | DAW) Live (depending on which of the ISOs you boot)
- Non-US Keyboard selection
- Non-US Language selection
- Memory test with memtest86+

You can select a keyboard mapping that matches your computer's. Also you can boot Slackware in another language than US English. If you stick to US English interface language you will probably still want to change the timezone because it will default to UTC. You have to specify a custom timezone manually by adding “tz=YourGeography/YourLocation” because the syslinux bootmenu does not offer you a selection of timezones. Syslinux allows you to edit the boot commandline by pressing <TAB>. Press <ENTER> to boot after you made your changes or <ESC> to discard your edit and return to the menu.

UEFI boot

On UEFI computers, Grub2 handles the boot and it will show a menu similar (and similarly themed) to the Syslinux menu:

- Start (SLACKWARE | KTOWN | XFCE | MATE | DAW) Live (depending on which of the ISOs you boot)
- Non-US Keyboard selection
- Non-US Language selection
- Non-US Timezone selection
- Memory test with memtest86+
- Help on boot parameters

Editing a Grub menu before booting it is possible by pressing the “e” key. After making your changes to the boot commandline, press <F10> to boot. To discard your changes, press <ESC>.

Another difference between Syslinux and Grub2 menus: in Grub2 you can select a non-US keyboard, language and/or timezone and you will return to the main menu every time. You still have to select “Start SLACKWARE Live” to boot the computer. In the Syslinux menu, only the keyboard selection menu will return you to the main menu. Any non-US *language* selection on the other hand will boot you into Slackware Live immediately; without returning to the main menu. This is a limitation of syslinux which would require exponentially more menu files to construct a menu with more choices. Grub2 supports variables which make it easy to modify a menu entry's characteristics.

UEFI Secure Boot

On computers with Secure Boot enabled, extra measures may be required to boot an Operating System. Slackware for instance, is unable to boot on a computer that has Secure Boot enabled. Historic liveslak based ISOs are also not able to boot there. From liveslak-1.5.0 and onwards, Secure Boot is supported for the 64-bit ISO images.

Secure Boot enforces that the first-stage bootloader is signed with an encryption key known to Microsoft. For Linux based Operating Systems, the most widely used solution is to place an small single-purpose bootloader before the regular Linux bootloader. This EFI bootloader is called 'shim'. Shim must be cryptographically signed by Microsoft for it to successfully boot a computer. This is not a trivial process, Microsoft is very strict about the signing process because in essence your signed bootloader will boot anything on a Secure Boot enabled computer, including malware if that was signed by your 'distro key'. That would create a huge security hole and defy the purpose of Secure Boot.

Signing your Grub bootloader and your kernel also becomes mandatory, because the 'shim' refuses to load un-signed binaries. This complicates the process of upgrading to a new kernel further.

The Slackware Live OS boots on a Secure Boot enabled computer if created with liveslak-1.5.0 or newer, and only for the 64-bit liveslak ISO images. The Slackware Linux distro does not ship a 'shim' which is signed by Microsoft, so how to get around the dilemma of requiring a signed 'shim'?

To realize this, the Slackware Live ISO 'borrows' a 3rd-party 'shim'. The binaryis actually called `bootx64.efi` in the `/EFI/BOOT/` directory and has been extracted from another distro's officially signed 'shim' package; Fedora by default but the Debian and openSUSE shim are also supported by the `make_slackware_live.sh` script. This 3rd-party 'shim' binary has been signed by 'Microsoft UEFI CA' which will allow it to boot on any computer. We just need to tell it that is OK to load Slackware's Grub and kernel into memory.

A distro 'shim' like Fedora's contains an embedded distro SSL certificate and 'shim' will trust the signature of any binary (grub, kernel, etc) which has been signed using that certificate. Of course, 3rd-party 'shim' binaries do not embed a Slackware SSL certificate. Therefore, another means must be used to establish trust. Secure Boot recognizes additional SSL certificates in the computer's MOK (Machine Owner Key) database as valid. The 'shim' trusts custom SSL vertificates of signed binaries, if they are present in the MOK database. It is up to the user (the Machine Owner) to enroll a custom SSL certificate into that database.

The Grub and kernel images of Slackware Live Edition are signed with an 'Alien BOB' SSL certificate and private key. This SSL certificate needs to be added to the MOK database of your Secure Boot enabled computer. All liveslak ISOs use this specific certificate plus its associated private key. The private key will of course never be distributed but a 'DER-encoded' version of the public certificate is distributed as part of the ISO. You can find it as `/EFI/BOOT/liveslak.der` inside the ISO. On a persistent USB stick which you created from the ISO, this will be on the second partition (the ESP).

Add the ''liveslak.der'' certificate to the MOK database

There are two ways to add or enroll this certificate.

- When you boot a Secure Boot enabled liveslak ISO for the first time, the 'shim' will fail to validate the certificate of liveslak's Grub. It will then start the 'MokManager' showing you a nice blue screen with a dialog requesting you to enroll a public key (aka the SSL certificate) from disk. You can use the file selector to browse to the 'efi' partition and there to the `./EFI/BOOT/`

directory. Select the `liveslak.der` and confirm that this is the correct certificate. The computer will then reboot and after reboot, you will automatically end up in the Grub boot menu without any further intervention.
- If you already have a Linux OS up and running on that computer, you can use the program `mokutil` to enroll the key before you boot a liveslak ISO:

```
# mokutil --import liveslak.der
```

. This command will schedule a request to shim, and the first time you boot a liveslak ISO the MokManager will ask confirmation to enroll the scheduled key. In other words, you won't have to 'enroll from disk'.

Note that MOK key enrollment is a one-time action for the official liveslak based ISOs. All future liveslak ISOs will also be signed using this `liveslak.der` certificate and as long as it stays in your computer's MOK database, the 'shim' will load Grub and the kernel without complaint.

Note that you can create your own SSL certificate plus private key and use those to generate custom liveslak ISO images with Secure Boot support. All you need to do is to enroll the public key (the DER-encoded version of your SSL certificate) into the MOK database of your computer. The MOK database has room for multiple keys so yours as well as liveslak's keys (and more) will fit there.

Boot from an ISO file on disk

If you downloaded a liveslak ISO file and want to boot that ISO directly from its location on your computer's hard drive, you can use this Grub configuration block and add it to your `/boot/grub/grub.cfg` (the example code assumes you downloaded the XFCE ISO and stored it as "`/data/ISOS/slackware64-live-xfce-current.iso`"):

```
menuentry "LIVESLAK ISO" --class gnu-linux --class os --class icon-linux {
  set iso='/data/ISOS/slackware64-live-xfce-current.iso'
  set bootparms='load_ramdisk=1 prompt_ramdisk=0 rw printk.time=0 kbd=us
tz=Europe/Amsterdam lang=nl'

  search -f $iso --set=root
  loopback loop $iso
  linux (loop)/boot/generic livemedia=scandev:$iso $bootparms
  initrd (loop)/boot/initrd.img
}
```

This example will add a 'LIVESLAK ISO' menu entry to your local computer's boot menu, through which you can start a downloaded XFCE Live ISO pre-configured for a US keyboard, Dutch language and Amsterdam timezone. You should of course change the "`bootparms`" string so that it matches your requirements.

Transfering ISO content to USB stick

A script is available which allows you to transfer the ISO image content to a USB stick, making some modifications depending on the script's parameters.

The USB stick will be erased and re-formatted when running this script (except when using the '-r' refresh option)! Before inflicting any irreversible damage, the script will show you a prompt at which point you can evaluate whether it is safe to continue.

This script, called 'iso2usb.sh', accepts the following parameters:

```
  -c|--crypt size|perc       Add a LUKS encrypted /home ; parameter is the
                             requested size of the container in kB, MB, GB,
                             or as a percentage of free space
                             (integer numbers only).
                             Examples: '-c 125M', '-c 2G', '-c 20%'.
  -d|--devices               List removable devices on this computer.
  -f|--force                 Ignore most warnings (except the back-out).
  -h|--help                  This help.
  -i|--infile <filename>     Full path to the ISO image file.
  -o|--outdev <filename>     The device name of your USB drive.
  -p|--persistence <name>    Custom name of the 'persistence'
directory/file.
                             If it does not exist yet, create it manually.
  -r|--refresh               Refresh the USB stick with the ISO content.
                             No formatting, do not touch user content.
  -s|--scan                  Scan for insertion of new USB device instead of
                             providing a devicename (using option '-o').
  -u|--unattended            Do not ask any questions.
  -v|--verbose               Show verbose messages.
  -w|--wait<number>          Add <number> seconds wait time to initialize
USB.
  -C|--cryptpersistfile size|perc
                             Use a LUKS-encrypted 'persistence' file instead
                             of a directory (for use on FAT filesystem).
                             Format for size/percentage is the same
                             as for the '-c' parameter.
  -P|--persistfile           Use an unencrypted 'persistence' file instead
                             of a directory (for use on FAT filesystem).
```

Examples:

- Create a USB version of Slackware Live, where the USB stick is known to the system as '/dev/sdX. Note - the value for the output parameter is the device name of the stick and not one of its partitions!

```
# ./iso2usb.sh -i ~/download/slackware64-live-14.2.iso -o /dev/sdX
```

- Create a USB Live like above, but this time adding an encrypted /home filesystem with 750 MB of space, and at the same time increase the wait time on boot to 15 seconds (useful for slow USB media that fail to start the Live OS otherwise):

```
# ./iso2usb.sh -i slackware64-live-current.iso -o /dev/sdX -c 750M -w 15
```

- Create a USB Live with an encrypted /home (allocating 30% of the stick's free space for /home) and where the persistent data will be stored in a container file instead of a directory:

```
# ./iso2usb.sh -i slackware64-live-current.iso -o /dev/sdX -c 30% -P
```

- Create a USB Live with both the /home and the persistent data encrypted (the persistence filesystem will be 300 MB in size):

```
# ./iso2usb.sh -i slackware64-live-current.iso -o /dev/sdX -c 30% -C 300M
```

- Refresh the system modules on a USB Live using a Live ISO as the source. Let the script scan for insertion of a USB stick instead of specifying the device name on the commandline. Note that the addons and optional modules will not be touched by this action:

```
# ./iso2usb.sh -i slackware64-live-current.iso -r -s
```

You might have noticed that the “-P” parameter does not accept a size parameter. This is because the unencrypted container file is created as a 'sparse' file that starts at zero size and is allowed to grow dynmically to a maximum of 90% of the initial free space on the Linux partition of the USB stick.

Using the Live OS to install Slackware to hard disk

All variants of Slackware Live Edition contain a script “setup2hd”, a tweaked version of the regular Slackware setup program. The “setup2hd” script supports regular Slackware network installations. In addition it allows you to install the Slackware release on which the Live OS is based, to the computer's local hard disk. You must boot the Live OS first, and then start `setup2hd` either in an X Terminal in your graphical Desktop Environment (aka Runlevel 4), or from the console in Runlevel 3. The fact that you can start “setup2hd” from a graphical terminal means that during installation, you can continue browsing, listening to music, watching video, reading an e-book or whatever else makes you pass the time.

The “setup2hd” script has some capabilities that the original Slackware 'setup' lacks:

- It will launch fdisk/gdisk if you forgot to create Linux partitions in advance;
- It will allow you to create a regular user account and set its password;
- It will prompt you to set the root password in a graphical dialog.

Updating the kernel (and more) on a USB stick

A script is available which allows you to tweak the content of a USB Live stick.

Specifically, the script is able to:

- Upgrade the kernel and modules, making a backup of the old kernel and modules.
- Restore the backed-up kernel and modules if the new kernel is not working.
- Add network support modules for PXE boot (if missing).
- Increase (or decrease) USB wait time during boot.
- Replace the Live init script inside the initrd image with a new script that you supply.
- Move current persistence data to a new squashfs module in 'addons' afther which the persistence store will be re-initialized. The new module's name is time-stamped (/liveslak/addons/0099-slackware__customchanges-yymmddHHMMSS.sxz) so that this action can be repeated many times.

The script is meant to be used while you are running Slackare Live from that same USB stick but this is not mandatory. With the exception of the '-p' option which moves the persistence data into a squashfs module, its functions can be used on any Linux computer where you can insert the USB stick.

Before making any modifications, the script will show you a prompt at which point you can evaluate whether it is safe to continue.

This script, called 'upslak.sh', accepts the following parameters:

```
  -b|--nobackup              Do not try to backup original kernel and
modules.
  -d|--devices               List removable devices on this computer.
  -h|--help                  This help.
  -i|--init <filename>       Replacement init script.
  -k|--kernel <filename>     The kernel file (or package).
  -m|--kmoddir <name>        The kernel modules directory (or package).
  -n|--netsupport            Add network boot support if not yet present.
  -o|--outdev <filename>     The device name of your USB drive.
  -p|--persistence           Move persistent data into new Live module.
  -r|--restore               Restore previous kernel and modules.
  -s|--scan                  Scan for insertion of new USB device instead of
                             providing a devicename (using option '-o').
  -v|--verbose               Show verbose messages.
  -w|--wait<number>          Add <number> seconds wait time to initialize
USB.
```

Examples:

- Get a listing of all available removable devices on the computer:

./upslak.sh -d

- Updating kernel and modules, providing two packages as input and assuming the USB stick is known as /dev/sdX:

./upslak.sh -o /dev/sdX -m kernel-modules-4.19.0-x86_64-1.txz -k kernel-generic-4.19.0-x86_64-1.txz

- Restore the previous kernel and modules after a failed update, and let the script scan your computer for the insertion of your USB stick:

./upslak.sh -s -r

- Replace the Live init script with the latest template taken from the git repository:

./upslak.sh -o /dev/sdX -i liveslak/liveinit.tpl

PXE booting the Live OS

Slackware Live Edition can do a network boot boot using PXE protocol off a NFS export. Extract the

content of the ISO to (for instance) a new directory called `slackware-live` below your TFTP server's `/tftproot` directory and export that directory via NFS. Then add lines like this to your `pxelinux.cfg/default` file (assuming your NFS server has IP address `192.168.0.1`):

```
label liveslak
kernel slackware-live/boot/generic
append initrd=slackware-live/boot/initrd.img load_ramdisk=1 prompt_ramdisk=0
rw printk.time=0 kbd=us tz=Europe/Amsterdam locale=us_EN.utf8
nfsroot=192.168.0.1:/tftpboot/slackware-live hostname=pxelive
```

As shown in the example above, a boot parameter `nfsroot` is used for network boot. The parameter value defines the IP address of your NFS server and the path of the NFS export where you extracted the Slackware Live ISO. Hint: to get a listing of the exported shares of your NFS server, run `showmount -e localhost` on the NFS server.

Actually, two boot parameters are available to properly support network boot. A second boot parameter `nic` can be used to define the characteristics of your Live environment's network configuration, like the name of the network interface, static IP adress and such. If you are on a network where a DHCP server configures your clients, then the `nic` parameter will not be needed as Slackware Live Edition will figure out all the details by itself.

Syntax of these two parameters:

```
nfsroot=ip.ad.dr.ess:/path/to/liveslak
nic=<driver>:<interface>:<dhcp|static>[:ipaddr:netmask[:gateway]]
```

Example uses of the two network boot parameters:

```
nfsroot=192.168.1.1:/tftproot/slackware-live
nic=auto:eth0:dhcp
nic=auto:eth0:static:10.0.0.21:24:
nic=:eth1:static:192.168.1.6:255.255.255.248:192.168.1.1
```

After you have setup your PXE environment (DHCP, TFTP and NFS servers) properly using the above information, boot one of your PXE-capable computers, interrupt the boot and select “network boot” and type or select the appropriate label (in the above example, that would be `liveslak`). You will see the kernel and initrd being downloaded and booted, and then the Live OS will start just as if it was running from a local medium.

If your DHCP server takes too long in acknowledging the client's request, the DHCP client times out and the boot of your Live OS will fail because the NFS-mounted Live filesystem will not become available. In that case you can try increasing the wait time before the DHCP client decides that it is not going to get an IP address from the server. Add the boot parameter `dhcpwait=30` (example value) where the number `30` is the number of seconds the DHCP client should wait for a server response. You should of course pick a value that is sufficiently large for your network setup.
The default DHCP wait time of the Live OS is 20 seconds.

Persistence is not supported in this configuration; currently the overlayfs does not support NFS as a writable layer in the live filesystem.

PXE server

Slackware Live Edition is not just capable of booting as a PXE client; it is able to run a PXE server all by itself.

What does that mean?

A practical example would be that your bring a USB stick with Slackware Live Edition to a LAN party, use it to boot one of the computers and then all the other computers in the (wired) LAN will be able to do a network boot and run the same Slackware Live Edition a couple of minutes later. The computer with the USB stick will act as the PXE server and all the other computers will be its PXE clients, reading the Slackware data off that USB stick. The clients will inherit the server's timezone,language and keyboard settings by default but those can be overridden. The PXE clients will not have 'persistence'. If the server has access to the Internet, the clients will have access as well.

How to start the PXE server?

When you boot the Live OS you can then start a script “pxeserver” from the console in runlevel 3 or from an X terminal in runlevel 4. The script will gather all required information and if it is unable to figure something out by itself it will ask you. If it is unable to figure out the wired network interface that it should use, you can add the name of your interface (for instance, eth1) as a single parameter to the script when you start it.

The PXE server uses dnsmasq to offer DNS to the PXE clients. The dnsmasq program will enable its internal DHCP server capabilities if your LAN does not have its own DHCP server. Dnsmasq will also start a TFTP server which the PXE clients will connect to in order to retrieve the boot files (kernel and initrd). The `pxeserver` script also starts a NFS server which will be used by the Live initrd to obtain the squashfs modules and boot the Live OS. If your PXE server has multiple network interfaces, for instance a wireless interface which is connected to the outside world and a wired interface connected to another computer which will become a PXE client (or indeed connected to a switch with a whole bunch of prospective PXE clients behind that) then the PXE server will setup packet forwarding so that the PXE clients will be able to access the outside world through the wired interface and out to that other interface.

If you have multiple network interfaces, it is important to know that dnsmasq will only bind to the interface where you want PXE clients to connect to. In a multi-NIC situation where a second NIC is connected to the outside world (your local network), this means that the DHCP/DNS server started by dnsmasq will not interfere with an existing DHCP server in your local network.

Once the PXE server is running, the script will show you the dnsmasq's activity log in a dialog window so that you can monitor the PXE clients that are connecting.

If your PXE server computer has sufficient RAM, it is strongly advised to boot the server's Live OS from the USB stick with the '`toram`' parameter. When more than a few PXE clients start reading OS files from the PXE server, the USB stick will become a bottleneck. Running the server OS from RAM will get rid of that bottleneck.

Boot parameters explained

Press <F2> in the syslinux boot screen for an overview of (most) boot parameters. When booting

Grub, select the menu “Help on boot parameters” instead. The Grub help is ugly, I know, but Grub does not offer anything better than this.

The following parameters are recognized by Slackware Live Edition. To boot with default values just press ENTER.

Desktop Environment

0|1|2|3|4|5|6|S|s|single ⇒

```
Select a runlevel to start with.
The default is 4 for graphical login.
```

kbd=fr xkb=ch,fr ⇒

```
Example of custom X keyboard layout.
The parameter xkb can be set to "XkbLayout,XkbVariant,XkbOptions".
The boot menus will configure some of these for you but you can
of course always modify the values.
Note that the optional XkbOptions can be several comma-separated values.
The XkbLayout and XkbVariant values must not contain commas.
You can set just the XkbVariant by adding something like "kbd=ch xkb=,fr"
```

livepw=“somestring” ⇒

```
Change the password for user "live".
The password is passed as a cleartext string.
```

locale=nl_NL kbd=nl tz=Europe/Amsterdam ⇒

```
Example of language,
keyboard and/or timezone customization.
```

rootpw=“somestring” ⇒

```
Change the password for user "root".
The password is passed as a cleartext string.
```

Custom software

load=nvidia ⇒

```
Load and configure Nvidia drivers if available
in the ISO (not for SLACKWARE and XFCE variants by default).
```

load=mod1[,mod2[…]] ⇒

```
Load one or more squashfs modules
```

```
from the directory "/liveslak/optional".
By default none of these "optional" modules are loaded on boot.
```

noload=mod1[,mod2[...]] ⇒

```
Prevent loading of one or more
squashfs modules from the directory "/liveslak/addons".
By default all these "addon" modules are loaded on boot.
```

Network boot

dhcpwait=<numseconds> ⇒

```
Maximum wait time for the DHCP client to configure a network interface
(default: 20 seconds).
```

nfsroot=ip.ad.dr.ess:/path/to/liveslak ⇒

```
defines the IP address
of the NFS server, and the path to the extracted content
of Slackware Live Edition.
```

nic=<driver>:<interface>:<dhcp|static>[:ipaddr:netmask[:gateway]] ⇒

```
network device customization, usually this parameter is
not needed when your network runs a DHCP server.
Specify a driver if UDEV does not detect the device. Specify the
interface if Slackware Live can not figure it out. If you specify
'static' you need to also specify ipaddr and netmask. The gateway
is optional but needed to access the internet for instance.
```

Hardware related

localhd ⇒

```
initialize RAID/LVM on local hard drives.
```

tweaks=tweak1[,tweak2,[,...]] ⇒

```
Implemented tweaks:
nga - no glamor 2D acceleration, avoids error "EGL_MESA_drm_image required".
nsh - no 'new style' sub-pixel hinting in freetype.
tpb - enable TrackPoint scrolling while holding down middle mouse button.
syn - start the syndaemon for better support of Synaptics touchpads.
ssh - start the SSH server (disabled by default).
```

nomodeset ⇒

```
Boot without kernel mode setting, needed with
some machines.
```

rootdelay=10 ⇒

```
Add 10 second delay to give the kernel more
time to initialize USB. Try this if booting fails. Default is 5.
```

swap ⇒

```
Allow the Live OS to activate all swap partitions on
the local hardware. By default, no swap is touched.
```

Media tweaks

cfg=[skip|write] ⇒

```
Specify 'skip' to skip disk-based configuration file containing
OS parameters; or specify 'write' to write current OS parameters
to disk.
```

domain=your_custom_domain ⇒

```
Specify a custom domain name.  Defaults to 'example.net'.
```

hostname=your_custom_hostname[,qualifier] ⇒

```
Specify a custom hostname.  A qualifier 'fixed' can be appended
to prohibit hostname modification in  case of network boot.
```

livemedia=/dev/sdX ⇒

```
Tell the init script which partition
contains the Slackware Live OS you want to boot. This can
become necessary if you have another copy of Slackware Live
installed in another partition.  Also accepted: UUID or LABEL.
```

livemedia=/dev/sdX:/path/to/live.iso ⇒

```
Use this if you want to
load the live OS from an ISO file on a local harddisk partition.
```

livemedia=scandev:/path/to/live.iso ⇒

```
Use this if liveslak should
scan all device partitions to locate the ISO file.
```

livemain=directoryname ⇒

```
Use this if you copied the content
of the ISO to a different directory than "liveslak".
```

luksvol=file1[:/mountpoint1][,file1[:/mountpoint2],...] ⇒

```
Mount LUKS container "file1" at mount point "/mountpoint1"
in the Live fs.  Multiple files must be separated by a comma.
Specify "luksvol=" to *prevent* mounting any LUKS container,
including an encrypted /home .
```

nop ⇒

```
No persistence, i.e. boot the virgin installation in
case your "persistence" directory got corrupted.
If you want to ignore any persistent data during boot,
including LUKS data, specify "nop luksvol=" .
```

nop=wipe ⇒

```
Wipe all data from persistence directory or container.
Useful in cases where your persistent data got corrupted.
```

persistence=name ⇒

```
Use this if you are using a different
directory/file than "persistence" for storing persistent data.
```

persistence=/dev/sdX:/path/to/mypersistence persistence=scandev:/path/to/mypersistence ⇒

```
Use this if the persistence directory or container is not located on
the USB stick, but on a local hard disk partition.
Useful for network (PXE) boot where you still want to offer users
persistence.
```

toram ⇒

```
Copy the OS from the media to to RAM before running it.
You can remove the boot media after booting.
```

toram=all ⇒

```
Prevent writes to disk since we are supposed to run from RAM;
equivalent to parameter "toram".
```

toram=core ⇒

```
Load Console OS modules into RAM.
Console-only Slackware loads fast, contains 'setup2hd' and frees up your USB
drive
```

```
so you can overwrite it with a Persistent Live OS.
```

toram=os ⇒

```
Load OS modules into RAM, but write persistent data to USB.
```

Troubleshooting

blacklist=mod1[,mod2[...]] ⇒

```
Add one or more kernel modules
to the kernel blacklist to prevent them from loading, in case
they cause issues during operation.
```

debug ⇒

```
During init, pause at strategic locations while
assembling the overlay filesystem and show mount information.
```

debug=<number> ⇒

```
'2' enables verbose script execution;
'4' dumps you into a debug shell right before the switch_root.
```

rescue ⇒

```
After initialization, you will be dropped in a
rescue shell to perform lowlevel maintenance.
```

Layout of the ISO

The Live ISO contains three directories in the root of its filesystem:

- EFI/
- boot/
- liveslak/

The USB variant with persistence may have an additional directory in the root:

- persistence/

- The EFI/ directory contains the Grub configuration used when you boot the Live OS on a computer with UEFI.
- The boot/ directory contains the syslinux configuration used when the Live OS boots on a computer with a BIOS. This directory also contains the kernel and initrd files which are used to actually boot the OS.
- The liveslak/ directory contains all the squashfs modules which are used to assemble the filesystem of the Live OS, as well as files that are copied directly into the root of the Live

filesystem. It contains four subdirectories:
- addons/ - modules which are stored in this directory will always be added to the Live filesystem unless you prevent that with a “noload=” boot parameter;
- optional/ - modules in this directory will not be added to the filesystem of the Live OS unless you force this with a “load=” boot parameter;
- system/ - this directory contains all the modules which were created by the “make_slackware_live.sh” script. All these modules are numbered and the Live init script will use that to re-assemble the Live filesystem in the exact same order as they were created initially.
- rootcopy/ - this directory is empty by default. Anything you (the user of the ISO) add to this directory will be copied verbatim into the filesystem by the init script when the Live OS boots. You can use this feature for instance if you do not want to create a separate squashfs module file for your application configuration files.

Developer Documentation

Scripts and tools

make_slackware_live.sh

The first script:

The script “make_slackware_live.sh” creates an ISO file as its output which contains the Live OS. Thanks to Linux kernel 4.x and the squashfs-tools package in Slackware, the process of creating a Slackware Live ISO requires **no** (re)compilation of Slackware content or installing 3rd party packages.

The script's inner workings can be subdivided into several distinct stages. For the full Slackware ISO the process stages are as follows:

Install the Slackware packages

Stage one:

- The script reads a package sequence for the Live variant and installs all packages in this sequence to subdirectories of a temporary directory tree.
- Every Slackware package set (a, ap, d, … , y) or package list (min, noxbase, x_base, xapbase, …) is installed into a separate 'root' directory.
- Each of those root directories is “squashed” (using squashfs) into a separate squashfs module. Such a module is a single archive file containing the compressed directory structure of the installed packages.
- These module files are subsequently loop-mounted and then combined together into a single read-only directory structure using an “overlay mount”. The overlayfs is relatively new; earlier Live distros have been using aufs and unionfs to achieve similar functionality, but those were not part of any stock kernel source and therefore custom kernels had to be compiled for such a Live distro.
- This “overlay assembly” is the filesystem that will be booted as the filesystem of the Live OS.
- On 'top' of this series of overlayed read-only filesystems, a writable filesystem is added by the

"make_slackware_live.sh" script. This writable filesystem is used to store all the customizations to our distro that we want to be applied when Slackware Live boots up. See the next section for more detail.
- Note that when you boot the Live OS later on, another writable overlay will be used to capture any write operations performed by the OS. You will perceive the Live OS as a real OS that can write and remove files. That writable filesystem will be:
  - a RAM-based filesystem when the Live OS runs without persistence.
  - a directory or a loop-mounted container file on your USB stick, if you use a USB stick with persistence.

Configure the Live filesystem with useful out-of-the-box defaults

Stage two:

- Some filesystem initialization is done when the overlay has been assembled:
  - 'root' and 'live' user accounts are created,
  - an initial environment for the accounts is configured,
  - the desktop environment is pre-configured for first use,
  - the liveslak scripts "makemod", "iso2usb.sh" and "upslak.sh" are copied to "/usr/local/sbin/" in the ISO for your convenience,
  - the "setup2hd" script and the Slackware installer files are copied to "/usr/local/sbin" and "/usr/share/liveslak" respectively.
  - slackpkg is configured,
  - a locate database is created,
  - etc…
- All these modifications are written to the writable filesystem that was created in the previous section. This filesystem will also be stored on the ISO as a squashfs module and when the Live OS boots, it will be mounted read-only just like all the other modules. Its name will be "0099-slackware_zzzconf-current-x86_64.sxz" or more generically "0099-slackware_zzzconf-${SLACKVERSION}-${ARCH}.sxz"

Configure the boot stage of the Live OS

Stage three:

- an initrd is generated, containing a modified "init" script (other Live distros call it a "linuxrc script") which re-assembles the overlay filesystem on boot and configures the Live OS according to the boot parameters (language, keyboard, timezone, runlevel, …)
- a slackware generic kernel plus the above initrd are added to a syslinux (for BIOS computers) and a grub2 (for UEFI computers) configuration.

Create the ISO image

Stage four:

- a bootable ISO file is created using mkisofs.
- the "isohybrid" command is run on the ISO so that you can "dd" or "cp" the ISO to a USB stick and thus create a bootable USB media.

Done! You can find the ISO file and its MD5 checksum in the /tmp directory.

iso2usb.sh

The second script:

The “iso2usb.sh” script's runtime usage is explained in detail in a previous paragraph “Transfering ISO content to USB stick”.

This section explains how the script modifies the ISO for the enhanced USB functionality.

Layout of the USB stick

The “iso2usb.sh” script wipes and re-partitions the USB stick unless the “-r” or *refresh* parameter is used. See section “Transfering ISO content to USB stick” for an explanation of all commandline switches.
The script will create 3 partitions:

- First partition: a small (1 MB in size) FAT partition which is not used for Slackware Live Edition. It can be used by an alternative bootloader if needed. You can also store your LUKS keyfile on it to unlock a LUKS-encrypted Slackware Linux computer (see the README_CRYPT.TXT file on your Slackware DVD for more information on LUKS keyfiles).
- Second partition: a 100 MB VFAT partition containing the kernel, initrd and all the other stuff required by syslinux and grub2 to boot Slackware Live Edition.
- Third partition: a Linux partition taking up all of the remaining space. It contains the actual liveslak modules, the persistent live storage and optionally your encrypted homedirectory. You can use the remainder of this Linux *ext4* filesystem's free space to store anything you like.

Note that this script is the only supported method of transfering the liveslak ISO content to a USB stick and make that USB stick into a persistent live OS. Several 3rd party tools (like multibootusb, rufus, unetbootin) that claim to be able to mix several Live OS'es on a single USB stick and make them all work in a multi-boot setup, are not currently supporting liveslak.

Mounting a filesystem in an encrypted container

The script will create a file of requested size in the root of the Live partition using the 'dd' command. The 'cryptsetup luksCreate' command will initialize the encryption, which causes the script to prompt you with “are you sure, type uppercase YES” after which an encryption passphrase must be entered three times (two for intializing, and one for opening the container). If the container is used for an encrypted /home, its filename will be “slhome.img”. The script will copy the existing content of the ISO's /home into the container's filesystem which will later be mounted on top of the ISO's /home (thereby masking the existing /home). The Live OS is instructed to decrypt the container and mount the filesystem. This is done by editing the file “/luksdev” in the initrd and adding a line: “/slhome.img”. The iso2usb.sh script only supports creating and configuring an encrypted /home, but you can create additional encrypted containers yourself and mount them on other locations in the ISO's filesystem. In order for this to work, you need to edit the “/luksdev” file and add a line “/your/container.img:/your/mountpoint” i.e. container path and the target mount directory on a single

line, separated by a colon. The Live init script will create the mount point if it is missing.

Using a container file for storing persistence data

A second type of encrypted container exists, which can be used for storing your persistence data. The Live init script will check if it needs to enable persistence in this order:

1. is the USB filesystem writable? If so,
    1. does a directory /persistence exist? If so, use that; if not,
    2. does a file /persistence.img exist? If so, mount the file and if it is an encrypted container, ask for a passphrase during boot.

Adding USB wait time

For slow USB media, the default 5 seconds wait time during boot are sometimes insufficient to allow the kernel to detect the partitions on your USB device. The script can optionally add more wait time. It does this by editing the file “wait-for-root” in the initrd and updating the value which is stored there (by default “5” is written there by the “make_slackware_live.sh” script).

makemod

The third script:

The “makemod” script allows you to create a Slackware Live module easily, with a Slackware package or a directory tree as its input parameter.

Usage:

```
# makemod <packagename|directory> modulename.sxz
```

- The first parameter is either the full path to a Slackware package, or else a directory.
    - If a packagename is supplied as first parameter, it will be installed into a temporary directory using Slackware's “installpkg”. The content of the temporary directory will be squashed into a module by the “squashfs” program.
    - If a directoryname is supplied, its content will be squashed into a module by the “squashfs” program..
- The second parameter is the full pathname of the output module which will be created.

You can copy the module you just created (minding the filename conventions for a Slackware Live module, see paragraph “Slackware Live module format”) to either the optional/ or to the addon/ directory of your Live OS. If you copy it to the optional/ or addon/ directory of the liveslak sources then “make_slackware_live.sh” will use the module when creating the ISO image.

setup2hd

The fourth script:

The “setup2hd” script is a modified Slackware installer, so you will be comfortable with the process. The 'SOURCE' section offers two types of choices: a regular Slackware network installation using a NFS, HTTP, FTP or Samba server, as well as a choice of installing the Live OS which you are running. The script knows where to find the squashfs modules, so the “Install Live OS” selection will not prompt further inputs.

- The Slackware network installation is identical to that of the official Slackware installation medium.
- If you chose to install the Live OS, then after you select the target partition(s), every active module of the Live OS variant (SLACKWARE, KTOWN, MATE, …) is extracted to the hard drive. After extraction has completed, the script summarizes how many modules have been extracted. It will also show an example command to extract any remaining inactive or disabled modules manually. The final step in the installation is again the stock Slackware installer which kicks off the Slackware configuration scripts.

pxeserver

The fifth script:

The `pxeserver` script works as follows:

- It requires a wired network; wireless PXE boot is impossible.
- The pxeserver script tries to find a wired interface; you can pass an explicit interfacename as parameter to the script (optional).
- If multiple wired interfaces are detected, a dialog asks the user to select the right one.
- A check is done for DHCP configuration of this wired interface;
    - If DHCP configuration is found then pxeserver will not start its own DHCP server and instead will rely on your LAN's DHCP server.
    - If no DHCP config is found, the script will ask permission to start its own internal DHCP server. Additionally the user will be prompted to configure an IP address for the network interface and IP range properties for the internal DHCP server.
- The script will then start the PXE server, comprising of:
    - dnsmasq providing DNS, DHCP and BOOTP;
    - NFS and RPC daemons;
- The script will detect if you have an outside network connection on another interface and will enable IP forwarding if needed, so that the PXE clients will also have network access.
- The Live OS booted via pxelinux is configured with additional boot parameters:

```
nfsroot=<server_ip_address>:/mnt/livemedia
luksvol=
nop
hostname=<distroname>
tz=<server_timezone>
locale=<server_locale>
kbd=<server_kbd_layout>
```

  Which shows that the configuration of the Live OS where the PXE server runs is largely determining the configuration of the PXE clients.

- Note that when networkbooting, the hostname of the Live OS will be suffixed with the machine's MAC address to make the hostname of every network-booted computer unique.

upslak.sh

The sixth script:

The “upslak.sh” script's runtime usage is explained in detail in a previous paragraph “Updating the kernel (and more) on a USB stick”.

This section explains how the script modifies the content of the Live USB stick.

When the script is started, it will do some sanity checks and then extracts the content of the initrd image. Some characteristics of the initrd will be recorded:

- existence of previously backed-up kernel modules is checked,
- template variables and their values are obtained from the init sctript,
- the current USB wait time is checked.

Depending on the parameters passed to the script, it will then perform one or more of the following actions:

Update the kernel and moules

You can provide a new kernel and its modules in two ways. The '-k' option accepts a kernel image file or else a Slackware package contaning a kernel. The '-m' option accepts a directory tree of modules below “/lib/modules/, or else a Slackware package containing kernel modules. If there is sufficient space on the Linux and EFI partitions, the script will make a backup of the current kernel and modules by renaming the kernel and the module directory with a ”.prev“ suffix. Sufficient space means that at least 10 MB of free space must remain on the partition(s) after making the backup and installing the new kernel plus modules. If space is an issue, you can skip making a backup by providing the '-b' parameter to the script (a possibly unsafe choice).

Restore backed-up kernel and modules

If a backup was made of kernel and modules, the upslak.sh script is able to restore these using the '-r' option, thereby removing the replacements. This comes in handy when the replacement kernel turns out to be non-functional.

Add network support modules

This should normally not be needed. By default, all liveslak ISO images have network support built-in. But customized Live ISO images may be shipped without network support initially. If you want your Live OS to be PXE-bootable you need network support in the kernel. Use the '-n' option.

Increase (or decrease) USB wait time

Similar to the functionality of the “iso2usb.sh” script, the “upslak.sh” script is able to alter the USB

wait time at boot using the '-w' option.

Replace the liveslak init script

The init script inside the initrd image is the core of liveslak. The init script prepares the Live filesystem and configures several run-time OS parameters. If you have made modifications to this init script you can easily replace the default init script with your own script using the '-i' option. The “upslak.sh” script is smart enough to recognize a iveslak template as input. The ”.tpl“ extension of some liveslak files means that these are templates. They are not usable as-is, because they contain placeholder strings like ”@VERSION@“ or ”@DISTRO@“ that first need to be replaced with real values. The “upslak.sh” script will take care of these substitutions.

Wrap persistence data into a new squashfs module

Persistence data will accumulate over time on the USB stick. That is perfectly OK, and you can wipe it on boot if that is needed. But sometimes you want to capture the packages you installed into the persistent storage, and create a new squashfs module out of them. The “upslak.sh” script is able to move your persistence data into a new squashfs module using the '-p' option. The new module will be created in the ”/liveslak/addons/“ directory so that it will be loaded into the Live OS everytime your USB Live boots up. After creating the new module, the persistence store will be re-initialized (i.e. its content will be erased on the next boot). The new module's name is time-stamped (/liveslak/addons/0099-slackware_customchanges-yyyymmddHHMMSS.sxz where yyyymmddHHMMSS is the timestamp) so that this action can be repeated as many times as you want.

Creating a Live ISO from scratch

Creating an ISO image of Slackware Live Edition requires that you are running Slackware 14.2 or newer (64-bit). Older releases of Slackware have a kernel that is too old to support liveslak's use of the “overlayfs” kernel functionality, and are lacking the squashfs tools. Likewise, a Slackware Live Edition can only be created for Slackware 14.2 or newer.

You also need the “liveslak” script collection which can be downloaded from any of the links at the bottom of this page.

Liveslak is a directory tree containing scripts, bitmaps and configuration files. Only 6 scripts are meant to be run by you, the user. These scripts (“make_slackware_live.sh”, “iso2usb.sh”, “makemod”, “setup2hd”, “pxeserver” and “upslak.sh”) are explained in more detail in the section ”Scripts and tools“ higher up. When creating a Live ISO from scratch, you only need to run the “make_slackware_live.sh” script.

Liveslak sources layout

The toplevel 'liveslak' directory contains the following subdirectories:

- EFI/ - contains the skeleton for boot support on UEFI computers. Some of the UEFI configuration files are dynamically generated by the “make_slackware_live.sh” script.

- README.txt - this documentation.
- addons/ - squashfs modules placed in this directory will be loaded into the Live filesystem when the OS boots.
- contrib/ - contributed scripts that are not used directly for the creation and usage of a Live ISO.
- graphics/ - squashfs modules for proprietary GPU support (Nvidia) can be placed here. The module(s) will be copied to addons/ by the “make_slackware_live.sh” script for any Live Desktop Environment (except pure Slackware) that might profit from proprietary driver support.
- local64/ , local/ - these directories can contain Slackware packages considered 'local' i.e. not belonging to any repository. The package(s) will be converted to squashfs module(s) by the “make_slackware_live.sh” script, copied to the “addons/” subdirectory in the ISO and loaded into the Live filesystem when the OS boots.
- media/ - scripts and images that are specific to a Live variant.
- optional/ - squashfs modules placed in this directory will not automatically be loaded into the Live filesystem when the OS boots. You need to pass “load=[mod]” boot parameter to load any of these.
- patches/ - patches for Slackware scripts that need modifications to run inside a Live OS.
- pkglists/ - definition files of 3rd party repositories (*.conf) and the package lists to be used from those repositories (*.lst) must be placed in this directory.
- setup2hd/ - script templates used by the `setup2hd` disk installer.
- skel/ - contains compressed tarballs (whose filenames must match wildcard “skel*.txz”). These files will be extracted to the ”/etc/skel“ directory in the Live filesystem.
- syslinux/ - contains the skeleton for boot support on BIOS computers. Some of its files are dynamically generated by the “make_slackware_live.sh” script.
- xdm/ - graphical Slackware theme for the XDM graphical session manager for those ISO variants which do not ship with GDM, KDM or SDDM.

The toplevel 'liveslak' directory contains the following files:

- blueSW-128px.png , blueSW-64px.png - these are bitmaps of the Slackware “Blue S” logo, used for the “live” user icon and in the XDM theme.
- grub.tpl - the template file which is used to generate the grub menu for UEFI boot.
- iso2usb.sh - this script creates a bootable USB version wih persistence from a Slackware Live ISO.
- languages - this file contains the input configuration for language support. One language per line contains the following fields: “code:name:kbd:tz:locale:xkb”. Example: “nl:nederlands:nl:Europe/Amsterdam:nl_NL.utf8:”
  - code = 2-letter language code
  - name = descriptive name of the language
  - kbd = name of the console keyboard mapping for this language
  - tz = timezone for the language's native country
  - locale = the locale used in the country
  - xkb = optional custom X keyboard variant for the language
- liveinit.tpl - this is the template for the “init” script which is copied into the initrd image for the Live OS. Together with the Slackware generic kernel, the initrd is what boots the computer. The “init” script assembles the Live filesystem from its squashfs modules.
- make_slackware_live.conf - the configuration file for the “make_slackware_live.sh” script. You can define defaults for many script parameters here so that you do not have to edit the script itself.
- make_slackware_live.sh - the script that generates the Live ISO.
- makemod - this script creates a squashfs module out of a Slackware package (or out of a directory tree).

- menu.tpl - template which is used to generate the syslinux boot menu for BIOS computers.
- pxeserver.tpl - template to generate the script that starts a PXE server allowing other computers to boot Slackware Live over the network.
- setup2hd.tpl - template to generate the script you use to install your Slackware Live to a harddisk.
- setup2hd.local.tpl - here a developer of a custom Live OS can override the default post-installation routine by (re-)defining the function “live_post_install()” in the `setup2hd` script.
- upslak.sh - a script which allows you to tweak the content of a USB Live stick.

Generate the ISO

The liveslak's “make_slackware_live.sh” script accepts optional parameters to tweak the process of Live OS generation:

```
The script's parameters are:
 -h                  This help.
 -a arch             Machine architecture (default: x86_64).
                     Use i586 for a 32bit ISO, x86_64 for 64bit.
 -c comp             Squashfs compression (default: xz).
                     Can be any of 'gzip lzma lzo xz zstd'.
 -d desktoptype      SLACKWARE (full Slack), LEAN (basic Plasma5/XFCE),
                     DAW (Digital Audio Workstation), XFCE (basic XFCE,
                     stripped), KTOWN (ktown Plasma5 replacement), MATE
                     (Gnome2 fork replaces KDE), CINNAMON (fork of Gnome3
Shell
                     replaces KDE), DLACK (Gnome3 replaces KDE).
 -e                  Use ISO boot-load-size of 32 for computers
                     where the ISO won't boot otherwise (default: 4).
 -f                  Forced re-generation of all squashfs modules,
                     custom configurations and new initrd.img.
 -l <localization>   Enable a different default localization
                     (script-default is 'en').
 -m pkglst[,pkglst] Add modules defined by pkglists/<pkglst>,...
 -r series[,series] Refresh only one or a few package series.
 -s slackrepo_dir   Directory containing Slackware repository.
 -t <none|doc|mandoc|bloat>
                     Trim the ISO (remove man and/or doc and/or bloat).
 -v                  Show debug/error output.
 -z version          Define your Slackware version (default: current).
 -C                  Add RAM-based Console OS to boot menu.
 -G                  Generate ISO file from existing directory tree
 -H <hostname>       Hostname of the Live OS (default: darkstar).
 -M                  Add multilib (x86_64 only).
 -O <outfile>        Custom filename for the ISO.
 -R <runlevel>       Runlevel to boot into (default: 4).
 -S privkey:cert     Enable SecureBoot support and sign binaries
                     using the full path to colon-separated
                     private key and certificate files.
 -X                  Use xorriso instead of mkisofs/isohybrid.
```

The script uses package repositories to create a Live ISO. The packages will be installed into a temporary directory.

In order to create a Live ISO for any of these variants, the package repositories that are required must be available as a local directory (this can be a network-mounted directory). If you have not mirrored them locally, then all packages of the Slackware repository as well as those you require from a 3rd party repository will be downloaded from a remote server as long as a rsync URL for the repository is configured in ./pkglists/*.conf.

When all pre-reqs are met, you issue a single command to generate the ISO. The following example will create a pure Slackware Live Edition:

```
# ./make_slackware_live.sh
```

Another example which creates a MATE variant, configuring runlevel '3' as default and specifying a custom path for the Slackware package repository root (note that the script will look for a subdirectory “slackware64-current” below this directory if you are generating this ISO for slackware64-current):

```
# ./make_slackware_live.sh -d MATE -R 3 -s ~ftp/pub/Slackware
```

An example on how to create a DAW Live ISO which supports UEFI SecureBoot (since liveslak 1.5.0 and only for 64-bit), is compressed using 'zstd' instead of the default 'xz' and is generated using xorriso instead of mkisofs. You need to provide the full path to a SSL private key and certificate file:

```
# ./make_slackware_live.sh -d DAW -c zstd -X -S
/root/liveslak.key:/root/liveslak.pem
```

If you want to know what package sets are included in any of these Desktop Environments, run the following command:

```
# grep ^SEQ_ make_slackware_live.sh
```

for MATE, you will find:

```
SEQ_MSB="tagfile:a,ap,d,e,f,k,l,n,t,tcl,x,xap,xfce,y pkglist:slackextra,mate
local:slackpkg+"
```

Which means that most of the Slackware package series (excepting kde and kdei) will be installed from their tagfiles, and on top of that two package lists are installed from the pkglists/ subdirectory: slackextra and mate. Lastly, “slackpkg+” will be installed from a local directory.

Using the Customization Features of the Live OS

Master configuration file

You can create your own custom Live OS by changing its characteristics in the configuration file ”`make_slackware_live.conf`“. Among the things you can change are:

- The name of the Desktop variant (the script itself knows "*SLACKWARE*", "*KTOWN*", "*DAW*", "*XFCE*", "*MATE*", "*CINNAMON*", "*STUDIOWARE*" and "*DLACK*"),
- The list(s) of packages used for your custom distribution,
- The full name of the user (by default that is "*Slackware Live User*"),
- The name of the useraccount (by default that is "*live*"),
- The name of the distribution (by default that is "*slackware*"),
- And finally you can define a function "`custom_config()`" where you can add all your costom post-installation steps that are not covered in the "`make_slackware_live.sh`" script itself.

This is the section in `make_slackware_live.conf` which deals with these customizations. Two variables are required if you want to create your own custom Live OS: "**`LIVEDE`**" and "**`SEQ_CUSTOM`**", the rest is optional (but useful nevertheless):

```
# REQUIRED:
# Define a new name for your own variant of Slackware Live Edition:
#LIVEDE="CINELERRA"

# REQUIRED:
# Define your own custom package sequence for a custom Live ISO.
# In this example you would need to create two files
"pkglists/cinelerra.conf"
# and "pkglists/cinelerra.lst" defining the package location and package
list
# respectively):
#SEQ_CUSTOM="min,noxbase,x_base,xapbase,xfcebase,cinelerra"

# OPTIONAL:
# Use something else than the name "min",
# for the package list containing the generic kernel:
#MINLIST="min"

# OPTIONAL:
# Your custom distro name (will reflect in boot screen & filenames):
#DISTRO="cinelerra"

#OPTIONAL:
# Name of the 'live' user account in the Live image:
#LIVEUID="live"

# OPTIONAL:
# Marker used for finding the Slackware Live files:
#MARKER="CINELERRA"

# OPTIONAL:
# The filesystem label of the ISO:
#MEDIALABEL="CINELERRA"

# OPTIONAL:
# The ISO main directory:
#LIVEMAIN="cinelerra"
```

```
# OPTIONAL:
# Add your own Live OS customizations to the function custom_config() :
#custom_config() {
#  # Add your own stuff here which is not covered in the main script:
#}
```

Customizing the list of used packages

Any liveslak ISO variant contains a specific set of Slackware packages, as defined in the various SEQ_* variables used in the `make_slackware_live.sh` script. Your customized Live OS will be using variable "SEQ_CUSTOM".

Let's breakdown the definition of such a variable to explain how to customize the package set for your own live ISO.

The list of packages in the MATE ISO for instance, is defined by the SEQ_MSB variable (*MSB* stands for *Mate Slack Build*). Its value is as follows:

```
# grep ^SEQ_MSB make_slackware_live.sh
SEQ_MSB="tagfile:a,ap,d,e,f,k,l,n,t,tcl,x,xap,xfce,y pkglist:slackextra,mate
local:slackpkg+"
```

Three keywords can be identified in the value of a SEQ_* variable, and these determine where the packages to be installed are going to be searched for:

- tagfile - this is an Slackware tagfile for a complete package series. For instance, using "tagfile:ap" means: install all packages in the **AP** series.
- pkglist - this is a list of packages to be installed from the Slackware distro itself or from a Slackware-compatible 3rd-party repository. The file containing that package list is searched in the `./pkglists/` subdirectory of the liveslak toplevel directory. For instance, using "pkglist:mate" means: install all packages mentioned in the file `./pkglists/mate.lst`. If there is no matching `./pkglists/mate.conf` file then the packages are assumed to be present in the Slackware distro directory. Else the ".conf" file is parsed and the variables that are defined in the ".conf" file will be used while generating the ISO. Most importantly, "SL_REPO_URL" will contain the rsync URI pointing to the 3rd-party repository where the requested packages can be downloaded.
- local - some packages can not be found in Slackware-compatible repositories. The "local" keyword alows you to install packages from a subdirectory of the liveslak toplevel directory. For instance, using "local:slackpkg+" means: install all packages found in subdirectory `./local/slackpkg+/` or if you are generating a 64bit live ISO, install all packages found in directory `./local64/slackpkg+/`.

For the value of a SEQ_* variable, any combination of these keywords can be used. Every keyword is followed by a colon, and that is followed by a comma-separated list of relevant package definitions. They are all separated by spaces.

Custom background images

The Plasma5 based Live variants allow customization of the background image used for the login

greeter, the desktop wallpaper and the lock screen. The image you want to use for this purpose, must have a 16:9 aspect ratio and its dimensions should at least be 1920×1080 pixels. You must store the custom image inside the liveslak source tree: in the subdirectory `./media/<variant>/bg/` where "<variant>" is the lower-case name of the Live variant (variant 'KTOWN' equals directory 'ktown', 'DAW' becomes 'daw', etc).

The "make_slackware_live.sh" script will look there for a file named either "background.jpg" or "background.png". If you want, that file can be a symlink to the actual bitmap file. The image will be converted into a set of wallpaper images of different aspect ratios and sizes. The different aspect ratios like 16:9, 16:10 and 4:3 will be achieved by cropping the images if needed, to avoid distortion. The image set will be installed as a Plasma5 wallpaper called "Slackware Live", and configured to be the default Live OS background.

Internals of Slackware Live Edition

Overlayfs, squashfs

Overlayfs and squashfs are doing the real magic here. As explained earlier, the squashfs program takes a directory structure and complresses this into a single archive file. It does this in a special way, more like how mkisofs does this than how tar creates an archive. The resulting module can be loop-mounted to access the filesystem inside.

When you have several of these loop-mounted squashfs modules, each containing a fraction of the filesystem of the OS, you are going to stack these fractional filesystems on top of each other and thus assemble the complete filesystem (much like Tintin did in The Secret of the Unicorn when he overlayed several translucent pieces of parchment and looked through them to see the complete picture). Overlayfs is the driver that performs this 'overlaying' of the fractional filesystems. Overlayfs can work with many read-only fractional filesystems and exactly one writable filesystem. The writable filesystem is what gives your Live OS the appearance that it is writing to a hard drive - the writes to the overlay filesystem are however done to RAM (in that case you have a real Live OS) or to a 'persistence' filesystem which is stored somewhere on a writable medium (a USB stick with 'persistence' is an example of this case).

The initrd and its init script

The initrd used for the Slackware Live Edition is a standard Slackware initrd created with Slackware's "mkinitrd" command, with just one modification: its "init" script. The correct name for an 'initrd' nowadays is 'initramfs' by the way, short for "initial ram filesystem" because it contains the initial file system that gets loaded into kernel memory when your computer boots. The init script of an initrd is what prepares the root filesystem even before the actual OS starts. When the OS does start, it finds a root filesystem all ready to use. An example case is of course the LUKS-encrypted root filesystem. Another is a root filesystem stored on logical volumes (LVM). Some advance work is required to make the root filesystem accessible and start the real "init" program of the OS (PID 1). Just like the previous examples, you need a script in an initrd to assemble the root filesystem of a Live OS. Slackware's initrd does not support a Live environment, so the stock init script was expanded for use in liveslak.

What does the 'liveslak' init script do?

- It parses any boot parameters you entered (or were passed by syslinux/grub) and tries to make sense of them.
- It does some initialization just like the Slackware init (start udev, wait a bit for USB media to settle, load kernel modules, load custom keyboard mapping, initialize RAID etc) before it gets to the Slackware Live section.
- A RAM based filesystem is created which forms the base of all the rest.
- Using tools like 'blkid' and 'mount', the init script tries to locate the Live media. It uses blkid to search for the Live media's default filesystem label. If that fails (because you changed the label) it will use blkid to find all filesystem partitions available and mount them mount these one by one until the Live partition is found.
- With the Live media located, the next step is to loop-mount the squashfs modules one by one and add them to the overlay filesystem in the correct order. If you specified the 'toram' boot parameter, then a module will first be copied into RAM before loop-mounting it. This allows you to remove the boot media after booting since the complete OS will run from RAM.
- Modules will be loaded in order:
    - first the system modules (core modules in the system/ subdirectory)
    - then the addon modules (in the addon/ directory). If you do not want an addon to be loaded, you can specify “noload=modulename” on the syslinux/grub boot commandline
    - last, the optional modules (in the optional/ subdirectory). By default an optional module is not loaded unless you force it by adding “load=modulename” to the boot commandline.
- Next, persistence will be configured if the Live OS was booted from a writable media such as a USB stick. The init script will first look for a directory in the root of the Live partition of the USB stick called “persistence” and use that to store persistent changes to the Live filesystem. If that directory does not exist but a file “persistence.img” is found, then that file will be loop-mounted and persistent changes to the Live filesystem will be written to this container file. The “persistence.img” container can be LUKS-encrypted in which case the init script will ask you for a passphrase to unlock it before mounting.
- The overlay filesystem is then finalized by adding the writable toplevel directory structure (either persistent or volatile).
- The complete RAM filesystem which underpins the overlay is made available to the user of the Live OS as ”/mnt/live“
- The filesystem of the Live media is made available to the user of the Live OS as ”/mnt/livemedia“. If the media is a USB stick then you will have write access to ”/mnt/livemedia“.
- With the root filesystem assembled, the Live OS is configured before it actually boots:
    - if a OS-specific configuration file (by default `/liveslak/slackware_os.cfg`) exists, its contents will be parsed. Values of the variables defined in this file overrule any default *liveslak* or boot command-line values.
    - if you specified “swap” on the boot commandline, any available swap partition will be added to ”/etc/fstab“ in the Live OS.
    - if you specified a custom keyboard layout for the console (and optionally another for X) by using the “kbd” and “xkb” boot parameters then these will be confifured in ”/etc/rc.d/rc.keymap“ and ”/etc/X11/xorg.conf.d/30-keyboard.conf“ in the Live OS.
    - Same for any custom locale which was specified with the “locale” parameter, this will get added to ”/etc/profile.d/lang.sh“.
    - If timezone and hardware clock were specified in the “tz” parameter, these will be configured in ”/etc/localtime“ and ”/etc/hardwareclock“.
    - The boot parameters “livepw” and “rootpw” allow you to specify custom passwords for the 'live' and 'root' users; the defaults for these two are simply 'live' and 'root'. This is achieved by running the “chpasswd” command in the chrooted overlay so that a plain text password can be given as input.

    - The “hostname” boot parameter can be used to change the Live OS' hostname from its default “darkstar”. Configuration is written to ”/etc/HOSTNAME“ and ”/etc/NetworkManager/NetworkManager.conf“.
    - If the “blacklist” boot parameter was specified, then the kernel modules mentioned as argument(s) will be added to a modprobe blacklist file ”/etc/modprobe.d/BLACKLIST-live.conf“.
    - The ”/var/lib/alsa/asound.state“ file in the Live OS is removed to allow correct sound configuration on any computer where the Live media is booted.
    - The complete content of the /liveslak/rootcopy directory on the Live partition (may be empty) is copied to the filesystem root of the Live OS, potentially 'overwriting' files in the Live OS. Use the /liveslak/rootcopy to add customization to your Live OS when you run it off a USB stick.
    - And lastly but very importantly, any LUKS-encrypted container files are unlocked (init will ask you for the passphrase) and the filesystem(s) contained therein will be mounted in the Live OS. Currently, a LUKS-encrypted /home is supported. The files ”/etc/fstab“ and ”/etc/crypttab“ will be updated so that the Live OS will do the mounting and unmounting.
    - The init script will end by telling the kernel to swith to our new root filesystem (the overlay) and start the Slackware init program (PID 1, /sbin/init).
- From this moment onward, you are booting a 'normal' Slackware system and the fact that this is actually running in RAM and not from your local harddisk is not noticeable.

OS configuration file for persistent media

If present, the liveslak init will load a OS config file from a persistent Live medium such as a USB stick. In the case of *Slackware Live Edition* this file is called `/liveslak/slackware_os.cfg` - i.e. is placed in the ”`liveslak`“ directory of your USB drive. For custom non-Slackware Live OS-es based on liveslak, the filename may be different.
This file contains one or more “VARIABLE=value” lines, where VARIABLE is one of the following variables that are used in the live init script:

- BLACKLIST, KEYMAP, LIVE_HOSTNAME, LOAD, LOCALE, LUKSVOL, NOLOAD, RUNLEVEL, TWEAKS, TZ, XKB.

Values for the variables defined in this configuration file override the values already set via liveslak's own defaults or via boot-up command-line parameters.

When booting your persistent *Slackware Live Edition*, the optional boot-time parameter “cfg” deals with this OS configuration file. The “cfg” parameter understands two possible argument values:

- “cfg=write” will (over)write the OS configuration file to your USB drive, using the values for all of the above variables that are valid for that particular boot. So if your timezone is ”PST“ then one of the lines in that file will read ”TZ=PST“.
- “cfg=skip” will skip processing of an existing ”`/liveslak/slackware_os.cfg`“ file.

The OS configuration file is not present by default. You either create it at boot-time using ”`cfg=write`“ (which is a persistent change) or you create it manually using an ASCII text editor, after mounting the USB partition on a computer. As an example, here is the content of ”`/liveslak/slackware_os.cfg`“ on my own USB stick:

```
KEYMAP=nl
LIVE_HOSTNAME=zelazny
```

```
LOCALE=nl_NL.utf8
TWEAKS=tpb,syn
TZ=Europe/Amsterdam
```

Slackware Live module format

A Slackware Live module contains a directory tree, which has been 'squashed' into a compressed archive file by the program “squashfs”. The compression algorithm used is “xz” which explains the choice of the module's file extension ”.sxz“ meaning “squashed with xz”.

Slackware Live Edition expects its modules to adhere to a particularly loose filename convention:

- The filename format is “NNNN-modname-*.sxz”, where 'N' is a digit and 'modname' must not contain a dash '-'.
- The “modname” part is what you must specify in the boot parameters “load” and “noload”.
- Anything may be part of the '*' but most commonly used is “${VERSION}-${ARCH}”. The core modules in Slackware Live use the Slackware release as ${VERSION} and the Slackware architecture as ${ARCH}. For the modules in addons/ and optional/ subdirectories, ${VERSION} would commonly be the version of the program that is being made available in the module.
- The four digits of a modulename have a meaning. Some ranges are claimed by the core OS, so please do not use them. Their prefixes are based on the package source:

```
    0000 = contains the Slackware /boot directory
    0005 = Console OS modules when explicitly enabled for a regular ISO
installed otherwise from Slackware tagfiles
    0010-0019 = packages installed from a Slackware tagfile (a,ap,d,
... , y series)
    0020-0029 = packages installed from a package list as found in the
./pkglists subdirectory of the liveslak sources (min, noxbase, x_base,
xapbase, xfcebase etc)
    0030-0039 = a 'local' package, i.e. a package found in subdirectory
./local or ./local64 (depending on architecture)
    0099 = liveslak configuration module (contaning all the
customizations that change the installed packages into a usable Live
OS)
```

- Other ranges are free to be used. Note that order in which the filesystem of the Live OS is assembled by overlaying the squashed directory trees depends on the module numbering. So if you have modules that you want to have loaded in specific order, just ensure that their filenames have ascending numbers.

1. Example of a core module: 0099-slackware_zzzconf-14.2-x86_64.sxz
2. Example of an optional module: 0060-nvidia-352.79_4.4.1-current-x86_64.sxz

Other Slackware based Live distros

Naturally, there have been many who went before me, and since I started as a n00b in Linux Live land, I have learnt a lot about how a Live distro works from playing with these other Slackware-based Live distros. Allow me to name them, and show respect:

SLAX

Website: https://www.slax.org/

SLAX was the original Live variant of Slackware. The linux-live scripts which are used to create a SLAX ISO were generalized so that they can create a Live version of any OS that is already installed to a harddrive. SLAX development stalled a couple of years ago.
In 2017 a new release of SLAX became available, however Slackware is no longer its parent OS. New SLAX releases are based on Debian.

The Live functionality of SLAX is based on aufs and unionfs which requires a custom-built kernel with aufs support compiled-in. It is small and has its boot scripts tweaked for startup speed.

Porteus

Website: http://www.porteus.org/

Porteus was created as a fork of SLAX by the SLAX community when the development of SLAX seemed to have ended. Porteus has an active user community where it's “all about the modules”. The use of aufs instead of overlayfs allows Porteus (like SLAX) to add and remove squashfs modules in the running Live system on the fly, which sparked the development of a lot of community modules. It looks like the next generation of Porteus will be based on Arch Linux instead of Slackware: this has to do with the original Porteus developer leaving the team.

Salix Live

Website: http://www.salixos.org/download.html

Salix is a distribution based on Slackware with its own philosophy of “one tool per task” reducing the number of packages a lot, compared to its parent Slackware distro. Salix has implemented dependency checking in its package management tool. Live editions of Salix are available in several editions, each built around and focused on a different Desktop Environment or Window Manager. Live editions are available for XFCE and MATE.

Slackel

Website: http://www.slackel.gr/

Slackel is a Greek distro based on both Slackware and Salix. It comes in three flavors, each of which has a Live variant: KDE4, Openbox and Fluxbox. The Live scripts are a modification of Salix-Live.

SlackEX

Website: http://slackex.exton.net/

A website offering Live versions based on many regular Linux distributions. The SlackEX version is

loosely based on Slackware with a custom kernel and some tools that are not part of Slackware itself. I was unable to find the sources for this live distro. Its creator stopped SlackEX development in December 2017.

Liveslak Sources

Slackware Live Edition is created by the 'liveslak' scripts developed and maintained by Eric Hameleers aka Alien BOB alien@slackware.com.

- Git repository: git://git.liveslak.org/liveslak.git
- Git repository (browsable): http://git.liveslak.org/liveslak/
- Download mirror: http://www.slackware.com/~alien/liveslak/

Sources

- Original source: https://git.slackware.nl/liveslak/tree/README.txt
- Project landing page: https://liveslak.org/
- ISO downloads: https://download.liveslak.org/

- Originally written by Eric Hameleers

slackware, live, overlayfs, squashfs, author alienbob