Install Slackware on an online.net Dedibox BareMetal Server

This tutorial explains on how to install and boot Slackware Linux on online.net Dedibox BareMetal Server Start Family. It is focused on servers that you don't have the physical access to and those that don't support remote exposure of the hardware (i.e. no KVM over IP). The setup of this kind of servers is possible through a Web interface. Fortunately or not, this interface doesn't natively support Slackware installation. No worries, we will manage nonetheless.

Although the very first parts of the tutorial are Dedibox server specific, the rest is more generic. This means that the information provided here applies equally to other hosting offers, which just must provide similar rescue OS (more on the rescue OS down below). Conversely, the first parts apply well to Linux distributions other than Slackware. If you need fine grained control over the installation process on the Dedibox server, you're in good place.

1. Dedibox rescue OS

The rescue OS is an operating system that you can boot your server into using the Web interface. This OS allows you to perform maintainability tasks, should your main operating system fail to boot or should you need to access the server, while bypassing your main OS. There are multiple versions of the rescue OS to choose from; we will be using Linux based one.

You can (and should) connect to the rescue OS over SSH. One of the characteristics of the rescue OS is that it is volatile, meaning that changes made to it are lost after reboot. But more importantly, the SSH host keys are regenerated every time the rescue OS is booted, which results in the SSH host's key fingerprint being changed too, whenever you reboot. This makes checking the authenticity of the server a bit cumbersome.

One of the ways to verify the host's authenticity is to open online.net technical support ticket, asking to provide you with the rescue OS host's key fingerprint. And because the fingerprint changes with every reboot, it is not desirable to reboot the machine during Slackware installation. No big drama, it is possible to successfully install and then boot Slackware into its full glory with only a single reboot at the end of the journey. You can also connect without ensuring the host's authenticity, have a play and even trial installation. Once you are familiar with the environment, you can reboot again into the rescue OS, ask the technical support for the fingerprint, cleanup the hard drive and perform the final installation and setup. Just beware that without verifying the host's authenticity you are susceptible to MITM attacks. Although the rescue filesystem is volatile and you can wipe the disk content to make sure it is clean and safe, still, the hardware itself might be the target of the attack. And nowadays, hardware is actually running software (firmware) more often than not.

Fortunately, there is an alternative method that does not involve the engagement of the technical support. You can read the full details on LinuxQuestions.org: Verifying host authenticity (SSH) after logging in, over then accessible secure serial terminal. In the tutorial itself, I will only focus on how to get the thing done.

2. Serial console

The Web interface provides a serial console option, which allows you to interact with the server hardware. It is of somehow limited use with the rescue OS (i.e. no login possible), but can be fully utilised with the main OS. If properly configured, it will allow you to see the boot process of the main OS and also to have a terminal login, should the SSH connection be not available for whatever reason. (But don't expect anything fancy, it's just a serial console after all).

It turns out that, it is actually possible to log in into the rescue OS using the serial console. At least it works when Ubuntu 16.04 amd64 is selected as rescue OS. Unfortunately, this tutorial was written with the assumption that it was not possible (which actually was the case!). For that reason, some steps are more complex than they would be if the serial console was available from the beginning. This applies especially to the server's SSH authenticity verification step. Unfortunately, I won't update the tutorial to accommodate for this discovery.

3. Making the rescue OS available

If you've just purchased a fresh server, it should come with no operating system pre-installed. Unfortunately, it means also that the rescue OS is not available yet.

To enable the rescue OS: using the Web interface, go to your server management page: *Server → Server list → (server name) Manage*. If the only option you see is `[INSTALL]`, then the rescue OS is not available and you have to first install one of the offered operating systems. (Purist's note: choose the operating system wisely, as the icon associated with this OS is going to represent your Slackware system thereafter). Once the OS installation is complete, you are going to be presented with more options: `[REBOOT]`, `[RESCUE]`, `[SERIAL_CONSOLE]` and `[INSTALL]`. You're ready to go now. Just don't boot the rescue OS yet.

4. Preparing rescue OS access over SSH

Be careful not to overwrite your workstation's SSH keys in ~/.ssh

We are now about to create two SSH keys pairs. One pair is going to be used to login into the rescue OS and the other (only public key part) for rescue OS authentication. Type the following on your workstation:

```
$ mkdir dedibox_rescue_os_keys
$ cd dedibox_rescue_os_keys
```

```
$ ssh-keygen -t rsa -f login_key -N ''
$ ssh-keygen -t rsa -f auth_key -N ''
```

Using the Web interface, go to your SSH keys management page: *logged in as (username) → SSH keys* and add the *auth_key.pub* and *login_key.pub* public keys. Note that after the successful addition, the page is not automatically refreshed and to actually see the keys, you have to click *SSH keys* link on the left hand side.

Uploading the keys through the Web interface has the effect of making them available to the rescue OS. When the rescue OS boots, the keys are simply appended to the *~/.ssh/authorized_keys* file for the particular user.

These keys might be safely removed from the Web interface once you are done with the Slackware installation.

5. Booting and connecting to the rescue OS

Using the Web interface, go to your server management page: *Server → Server list → (server name) Manage*. Enable the serial console by clicking on the `[SERIAL_CONSOLE]` button and follow the guide. The console will be opened in a new browser tab. Note that the console connection has expiration time, so it won't stay there forever. Also, there are not many messages appearing related to the rescue OS booting, but it's still better than nothing. Rescue OS takes some time to boot and being able to see it can let you calm down a bit (just keep in mind that there is some time between the moment the messages stop appearing and the moment you can actually connect over SSH). The reason you need to start `[SERIAL_CONSOLE]` before `[RESCUE]` is that, the `[SERIAL_CONSOLE]` button disappears once you boot with the rescue OS.

It is still possible to access the serial console, even if the `[SERIAL_CONSOLE]` button is not visible. You just need to go to https://console.online.net/en/server/state/XXXXX/bmc address, replacing *XXXXX* part with the actual number of your server.

Now go back to server management page and click on `[RESCUE]`. When prompted for the operating system selection, choose `[Ubuntu_16.04_amd64]` and then click on `[CLICK_HERE_TO_LAUNCH_THE_RESCUE_SYSTEM]`. After a while, you will be presented with the details needed to connect to the rescue OS over SSH. You can also switch to the `[SERIAL_CONSOLE]` output window to monitor what is happening.

Once the rescue OS is fully booted, connect to it from your workstation:

```
$ ssh username@x.y.z.w -i ./login_key
```

When asked *Are you sure you want to continue connecting (yes/no)?*, answer yes.

It might be a good idea to temporarily add the rescue OS SSH host's key fingerprint to the ~/.ssh/known_hosts file on your workstation. This will allow you to re-login back (e.g. in the case of a broken connection) into the rescue OS without the need for repeating the authentication procedure described below. Just remember to remove the fingerprint after you are done or if the authentication procedure described below fails.

Once logged in, to authenticate the rescue OS, on the server side type the following:

```
$ cat ~/.ssh/authorized_keys
```

and then compare the public keys printed on the terminal with *auth_key.pub* and *login_key.pub* public keys you generated previously. If both the keys between the server and the workstation match, you are secure to go. (Again, if you want to understand the details, refer to the aforementioned LinuxQuestions.org thread).

Now, on the server side, I suggest you to first start the *screen* program and then log in as root (the password is given on the rescue OS connection details page):

```
$ screen
$ sudo su -
```

(Being paranoid, I change the user and root passwords provided by *online.net*).

I will not go into details of *screen* program, but the reason we want to use it, is its ability to maintain the remote terminal opened, even if the SSH (or rather network) connection breaks or you accidentally close the terminal window on your side. Normally, such an event would break the installation process. If that happens to you, and you use *screen*, then you can regain the remote terminal (with all the started commands still executing!), by simply connecting over SSH and then re-attaching to the so-called *screen* session:

```
$ screen -r
$ # Sometimes, detaching the session first is needed:
$ screen -rd SESSION_PID
```

All of the above means also that you cannot stop remote command execution by just closing the local terminal window; *screen* session will be maintained on the remote end until you explicitly close it. BTW, you can also use this functionality to lower the network traffic during installation phase, i.e. once the packages started installing and need no attention, you can detach from the session and then re-attach some time later to check the status. You use Ctrl+AD keyboard combination to detach from *screen* session. Check the Internet on how to use the *screen* program (the man page is actually enormously long).

6. Setting-up Slackware installer

environment (chroot)

The main motivation behind this HOWTO is the fact that *online.net* does not provide direct method of Slackware installation on their Dedibox servers. And that is great! The point is, if such a method existed, it would be something unfamiliar to a Slackware user: a GUI/Web based installer. Whereas being dropped to the Slackware's installer shell allows you to install and configure the system the way you want (TM). All the advanced options are available without much hassle. And we're going to use them all. :-^

The procedure detailed below is known and common. We're going to *chroot* into the unpacked Slackware *initrd* image and run *setup* from within there.

Our Slackware *chroot* will be running out of Ubuntu rescue OS, which is running out of RAM based filesystem. The size of the RAM filesystem is ~8.0 GiB (that depends on the total amount of system RAM). Hint: 8.0 GiB is more than enough to hold the full Slackware packages tree, if need be (i.e. if you want to download the packages before running *setup*).

Now, let's setup the Slackware installer *chroot*:

```
$ mkdir -p ~/slackware-chroot
$ cd ~/slackware-chroot
$ wget https://slackware.osuosl.org/slackware64-14.2/isolinux/initrd.img
$ gunzip -cd initrd.img | cpio -dvim
$ mount --bind /proc proc
$ mount --bind /sys sys
$ mount --bind /dev dev
$ mount --bind /dev/pts dev/pts
$ mount --bind /run run
$ touch etc/resolv.conf
$ mount --bind /etc/resolv.conf etc/resolv.conf
```

And run the chroot:

```
$ chroot ~/slackware-chroot /bin/bash --login
$ cd
```

Mounting /etc/resolv.conf provides DNS to the *chroot*.

Once you *chrooted*, you might want to play with the *TERM* environment variable, which is going to influence the way that *dialogs* are displayed. By default, *TERM=linux* and it does not work well if you use *screen* or are connecting from terminal emulator running under X.

For best results with *screen*, use the following:

```
$ export TERM=screen
```

and if you do not use *screen*, but are connecting from within X:

```
$ export TERM=xterm
```

Finally, for the bare virtual terminal (VT), leave the default:

```
$ export TERM=linux
```

```
Welcome to the Slackware Linux installation disk!
```

7. Partitioning

You can now partition the disk to your liking, with two exceptions. In this tutorial, I'm going to use separate partition to be mounted on */boot* directory. This directory will hold the kernel, initrd and bootloader config file. Which brings us to the second exception: as the bootloader, I'm going to use *syslinux* installed to MBR, which means the disk has to use MBR label type. If you need GPT, you're on your own.

The filesystem of the */boot* partition has to be supported by *syslinux*. *ext4* will do the job just fine. And when it comes to the size, 128 MiB is sufficient.

The following instructions will destroy the data on the disk.

You can use your favourite partitioning tool, e.g. *fdisk*, *cfdisk*, etc. I'm going to use *parted*. Note that the disk should be in “unmanaged” state, that is, services like LVM2 or software RAID (mdadm) should be deactivated. I had a lot of headache releasing the disk from the control of LVM2, when I was playing with it (but I managed).

By the way, the disk is already going to contain the partition table, which was created when we were installing one of the stock OS-es in order to enable the rescue OS. The partition table can be wiped with the following commands:

```
$ dd count=1 bs=512 conv=notrunc if=/dev/zero of=/dev/sda
$ partprobe
```

The following parted commands will create the MBR label and the /boot partition:

```
$ parted /dev/sda mklabel msdos
$ # Start at 1 MiB in the hope of a correct alignment:
$ parted -a optimal /dev/sda mkpart primary 1MiB 129MiB
$ # Set bootable flag:
$ parted /dev/sda set 1 boot on
```

Having the */boot [/dev/sda1]* partition in place, you can partition the remaining space in the way that suits your needs. I'm going to use LVM2 to manage the disk, so I create one big partition [*/dev/sda2*] that is going to be passed to LVM2. The setup procedure for enabling LVM2 is described in Appendix A. The following parted command will create the required partition:

```
$ # Passing "-a optimal" automatically aligns at the last sectors of the
disk.
$ # The start and end offsets have to be given explicitly:
$ parted -a optimal /dev/sda mkpart primary 129MiB 100%
$ # Set the 'lvm' flag only if you plan to use LVM2 for managing the disk.
$ parted /dev/sda set 2 lvm on
```

Let the kernel know about the partition changes:

```
$ partprobe
```

Let's verify the result:

```
$ parted /dev/sda print
Model: ATA SAMSUNG MZ7LN256 (scsi)
Disk /dev/sda: 256GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End    Size   Type     File system  Flags
 1      1049kB  135MB  134MB  primary  ext4         boot
 2      135MB   256GB  256GB  primary               lvm
```

8. Slackware installation (setup program)

If you prefer to download the required packages yourself, instead of letting the *setup* program do it for you (I actually prefer it that way), now is the time.

I want to use *rsync* to download the packages beforehand. Since Slackware *chroot* does not provide *rsync* command, we have to leave Slackware *chroot* for a moment, use *rsync* from Ubuntu rescue OS and then come back:

```
$ # Exit from Slackware chroot:
$ exit
$ mkdir -p ~/slackware-chroot/packages
$ cd ~/slackware-chroot/packages
$ # Download only: a, ap, d, l, n packages series (I don't need GUI apps),
$ # pay !attention! to the "." at the end of command line:
$ rsync -vaz
rsync://rsync.osuosl.org/slackware/slackware64-14.2/slackware64/{a,ap,d,l,n}
.
$ wget
```

```
https://slackware.osuosl.org/slackware64-14.2/slackware64/CHECKSUMS.md5
$ # Verify the checksums of the downloaded packages:
$ grep -P "\./(a|ap|d|l|n)/" CHECKSUMS.md5 | md5sum -c --quiet
$ mkdir -p ~/slackware-chroot/patches
$ cd ~/slackware-chroot/patches
$ Download patches, pay !attention! to the "." at the end of command line:
$ rsync -vaz
rsync://rsync.osuosl.org/slackware/slackware64-14.2/patches/packages .
$ wget https://slackware.osuosl.org/slackware64-14.2/patches/CHECKSUMS.md5
$ # Verify the checksums of the downloaded patches:
$ grep -P "\./packages" CHECKSUMS.md5 | md5sum -c --quiet
$ # Return to Slackware chroot:
$ chroot ~/slackware-chroot /bin/bash --login
$ cd
$ # Remember to update TERM as described earlier:
$ export TERM=screen
```

Having all the partitions in place, it's now time to run the *setup* program and perform the installation as you know it. Just remember to format and mount the */boot* partition when prompted by *setup*. When prompted, skip *LILO* installation, as we're going to use *syslinux* instead. If you downloaded the packages beforehand, point the *setup* to the pre-mounted */packages* directory. Otherwise, use *setup* to download the packages for you.

Don't reboot the machine when the setup program offers to at the end of the installation.

9. Freshly installed Slackware chroot

Yes, one more *chroot* to deal with. The configuration of the freshly installed system is best performed from the actual system itself. And we're going to do exactly that. When the *setup* program finished its job, it has left the Slackware root filesystem (and some additional ones) mounted on */mnt*. And there is nothing preventing us from *chrooting* into this directory, meaning that we can actually “log in” into the freshly installed system without rebooting.

This system is somehow limited (i.e. there are no services running), but has all the tools needed to perform the final configuration steps before rebooting the server.

To avoid any unpleasant surprises, we mount some possibly needed filesystems, before *chrooting*:

```
$ cd /mnt
$ mount --bind /run run
$ mount --bind /dev/pts dev/pts
$ # Only needed if you downloaded the patches and want to apply them:
$ mkdir run/patches
$ mount --bind /patches/packages run/patches
```

```
$ # Enter Slackware Chroot (tm):
$ chroot /mnt /bin/bash --login
$ cd
$ # Remember to update TERM as described earlier:
$ export TERM=screen
```

10. Applying patches

The following set of commands will apply all the available patches and let you know of any *.new* files to deal with:

```
$ find /run/patches -name \*.txz -exec upgradepkg {} \;
$ find /etc -name \*.new
```

11. Bootloader (syslinux)

The following set of commands will install the *syslinux* bootloader:

```
$ extlinux --install /boot
$ dd count=1 bs=440 conv=notrunc if=/usr/share/syslinux/mbr.bin of=/dev/sda
```

Then, create the *syslinux* config file:

```
$ cat << EOF > /boot/syslinux.cfg
PROMPT  0
TIMEOUT 0
DEFAULT vmlinuz-generic
SERIAL  1 9600

LABEL vmlinuz-generic
  KERNEL vmlinuz-generic
  APPEND console=ttyS1,9600 printk.time=0 quiet ipv6.disable=1 ro
  INITRD initrd-generic.gz
EOF
```

This configuration will enable the messages to appear on the serial console. I also specify some kernel parameters (*printk.time=0 quiet*) to considerably silence its output (error messages would still appear). As I do not want to bother with IPv6, I disable it at kernel level (*ipv6.disable=1*). As you can see, we will be using the generic kernel with initrd. This is the only way (that is, by means of initrd) the LVM2 can be made functional.

Note that the kernel and initrd paths specified in *syslinux.cfg* have to be relative to the */boot* directory. This is because *syslinux* is unable to read from LVM2 based root partition, so something like */boot/vmlinuz-generic* would not work (/ is on LVM2 partition in my case).

12. Initial RAM disk (initrd)

Create the *initrd* config file:

```
$ cat << EOF > /etc/mkinitrd.conf
# mkinitrd.conf
# See "man mkinitrd.conf" for details on the syntax of this file
#
#SOURCE_TREE="/boot/initrd-tree"
#CLEAR_TREE="0"
OUTPUT_IMAGE="/boot/initrd-generic.gz"
KERNEL_VERSION="$( readlink /boot/vmlinuz-generic | rev | cut -f1 -d- | rev
)"
#KEYMAP="us"
MODULE_LIST="ext4"
#LUKSDEV="/dev/sda2"
#LUKSKEY="LABEL=TRAVELSTICK:/keys/alienbob.luks"
ROOTDEV="/dev/vg0/rootfs"
ROOTFS="ext4"
#RESUMEDEV="/dev/sda2"
#RAID="0"
LVM="1"
#UDEV="1"
#MODCONF="0"
#MICROCODE_ARCH="/boot/intel-ucode.cpio"
WAIT="0"
EOF
```

As the comment says, refer to the *mkinitrd.conf* man page for details.  In particular, make sure your *MODULE_LIST*, *ROOTDEV* and *ROOTFS* are defined correctly. If you don't need LVM2 support, you can set *LVM="0"* (or comment it out).

The notable thing is how the *KERNEL_VERSION* is automatically derived, not for the running kernel, but rather for the installed kernel (which might be newer or older than the running one). */etc/mkinitrd.conf* is sourced by the */sbin/mkinitrd* script, so it is possible to use shell commands within the config file. On standard Slackware installation, you will find that */boot/vmlinuz-generic* is actually a symlink:

```
$ ls -l /boot/vmlinuz-generic
lrwxrwxrwx 1 root root 22 Dec 13 00:44 /boot/vmlinuz-generic -> vmlinuz-
generic-4.4.38
```

So the following code:

```
KERNEL_VERSION="$( readlink /boot/vmlinuz-generic | rev | cut -f1 -d- | rev
)"
```

will simply extract the version number from the *installed* kernel image.

And finally, to create the *initrd*, run the following command:

```
$ mkinitrd -c -F
```

Note that it is not necessary to run any *syslinux* related commands after creating or updating the *initrd* image. This is different from *LILO*, where you have to run *lilo* command after changing the *initrd* image.

13. Enabling serial console access

As of now, the serial console configuration in */boot/syslinux.cfg* allows for interacting with the bootloader and also to see the kernel messages, but it does not allow for root login over serial port. If you want to enable it, then uncomment the following line in */etc/inittab*:

```
s2:12345:respawn:/sbin/agetty -L ttyS1 9600 vt100
```

and the following line in */etc/securetty*:

```
ttyS1
```

You might also want to comment out the follwing lines in */etc/inittab*:

```
#c1:12345:respawn:/sbin/agetty --noclear 38400 tty1 linux
#c2:12345:respawn:/sbin/agetty 38400 tty2 linux
#c3:12345:respawn:/sbin/agetty 38400 tty3 linux
#c4:12345:respawn:/sbin/agetty 38400 tty4 linux
#c5:12345:respawn:/sbin/agetty 38400 tty5 linux
#c6:12345:respawn:/sbin/agetty 38400 tty6 linux
```

and the following lines in */etc/securetty*:

```
#tty1
#tty2
#tty3
#tty4
#tty5
#tty6
```

tty[1-6] are for the standard VT login prompts, but since we have no keyboard and no display, we cannot make any use of them.

14. Finalising the installation

We're done with the installation and initial configuration of the Slackware Linux. You can now prepare the system for reboot and, well, reboot. Before doing that, you might also consider looking at

the Appendix B, where I explain how to prepare the SSH stuff, so that after rebooting, you can connect to the server with SSH right away. (Otherwise, you will have to log in over serial console to perform the other configuration tasks).

First, prepare the hard disk to safely survive the reboot phase. Note that, I only (u)mount */mnt/boot* and */mnt* partitions as these are the only hard disk partitions I have. If you have more mounted disk partitions, you should umount them too:

```
$ # Exit freshly installed Slackware chroot:
$ exit
$ umount -v /mnt/boot
$ mount -v -o remount,ro /mnt
$ # This never hurts:
$ sync
$ # Only needed if LVM2 is used:
$ vgchange -an --ignorelockingfailure
$ # This never hurts again:
$ sync
$ # Shouldn't be needed, but just in case:
$ sleep 3
```

Now, go to the server management page and press `[BOOT_IN_NORMAL_MODE]` button. You can observe the reboot process on the serial console.

A. Setting up LVM2 disk management

The following instructions will destroy the data on the disk.

Before continuing to LVM2 partitioning, if the disk is already under LVM2 control, it has to be first deactivated. I use the following set of commands to do so:

```
$ lvscan
$ ( cd /dev/mapper && lvchange -an $(pvs --noheadings -o vg_name) )
$ vgscan
$ vgchange -an $(pvs --noheadings -o vg_name)
$ pvscan
$ pvremove -ffy $(pvs --noheadings -o pv_name)
$ partprobe
```

To find out more about LVM2, go to https://wiki.archlinux.org/index.php/LVM

Remember that the disk has already been partitioned using MBR in the “Partitioning” chapter and */dev/sda2* already exists. The following set of commands will activate LVM2 on */dev/sda2* and create

the partitions (the LVM2 partitions are going to sit on top of */dev/sda2* partition):

```
$ pvcreate /dev/sda2
$ pvdisplay
$ vgcreate vg0 /dev/sda2
$ vgdisplay
$ # Create the partitions (logical volumes):
$ lvcreate -L 12G vg0 -n rootfs
$ lvdisplay
$ vgchange -ay
```

NOTE:

- The good thing about LVM2 is that you can easily add more partitions later on.
- I have chosen partition sizes that suit my current needs, leaving significant free space. LVM2 can easily grow the sizes later on if needed.
- I haven't created the swap partition. The server has more than enough of RAM. But if needed, it can be easily added later on.

B. SSH server configuration (before rebooting)

You have to be in the *chroot* of the freshly installed Slackware system to perform the configuration steps detailed below.

If you enabled the *sshd* service during *setup*, it'll be automatically started the next time the Slackware system boots. Unfortunately, you won't be able to connect to it for two reasons:

1. the host keys are not generated yet, so you won't be able to verify host's authenticity and of course you don't want to connect without being able to verify it,
2. user's public key authentication is not set up and of course you don't want to be logging in using password authentication.

To solve the first issue, we need to manually perform the task that would normally be done by the Slackware init scripts when the system boots for the first time. Generating the host keys basically boils down to the following command:

```
$ ssh-keygen -A
```

And then to obtain the host's key fingerprint (I stick to RSA):

```
$ ssh-keygen -l -f /etc/ssh/ssh_host_rsa_key
```

Setting up public key authentication is a bit more cumbersome, but still far from being rocket science. :-^ First, you need to upload your public key from your workstation to the server. Run the following command on the workstation:

```
$ scp -i ./login_key ~/.ssh/id_rsa.pub user@x.y.z.w:~
```

NOTE: The above command uploads *~/.ssh/id_rsa.pub* public key, but for the transfer authentication, it uses the very same key you uploaded earlier using the Web interface.

Now, back to the server, create the required *~/.ssh* directory:

```
$ mkdir ~/.ssh
```

I told you to use the *screen* program at the beginning, right? Now we will make use of it. The public key that you uploaded above has been placed in the *user* home directory of the Ubuntu rescue OS. We need to rename it to the *authorized_keys* file in the *~/.ssh* directory of the fresh Slackware installation:

```
$ # Detach from screen session, you'll be dropped to Ubuntu rescue OS:
(keyboard) Ctrl+a d
$ # Login as root:
$ sudo su -
$ mv /home/user/id_rsa.pub /root/slackware-
chroot/mnt/root/.ssh/authorized_keys
$ # Exit root login:
$ exit
$ Re-attach to screen session:
$ screen -r
```

Ensure correct ownership and permissions, otherwise *sshd* won't let us in:

```
$ chown root:root ~/.ssh
$ chown root:root ~/.ssh/authorized_keys
$ chmod 0700 ~/.ssh
$ chmod 0600 ~/.ssh/authorized_keys
```

NOTE:

1. If you haven't used *screen*, you would just open second SSH connection to perform the above task. Alternatively, you could exit all the *chroots* and then run them again, but who would want to do that?
2. Remember that the correct server's network configuration has to be in place for you to be able to connect over SSH after reboot.

At this point, all the pieces should be in place and you should be able to successfully login to your fresh Slackware installation after the server is rebooted.

NOTE: I know I allow for root login over SSH. I have to live with that. :-^

Sources

- Originally written by Andrzej Telszewski

howtos, author atelszewski