Diskless Workstation

Aus wiki.archlinux.de

„Diskless-Workstation“ bezeichnet einen Computer, der über keine eigenen Festplatten verfügt und sowohl Betriebssystem als auch Daten von einem oder mehreren Servern über ein Netzwerk bezieht. (Wikipedia)

Um die in der Wikipedia Einleitung genannte Funktionalität zu erreichen wird u. a. von der Technik (**P**)reboot e(**X**)ecution (**E**)nvironment (*PXE*) Gebrauch gemacht.
In dem *PXE*-Artikel wird die *Funktionsweise* beschrieben.

Der Artikel nennt auch Einsatzmöglichkeiten:

- Ein Notfallsystem auszuliefern
- Ein Installationsmedium auszuliefern
- **Eine komplette Systemumgebung über das Netzwerk auszuliefern**

Um eine „Diskless-Workstation“ zu betreiben wird die letztgenannte Möglichkeit genutzt. Ebenso müssen mehrere Dienste auf dem Server gestartet und konfiguriert werden. Auf Grund des Wechsels von Sysvinit zu systemd und Einführung von netctl sind einige Artikel, die sich meist ausführlicher mit hier genannten Programmen oder Diensten beschäftigen, nicht mehr aktuell. Dieser Artikel trägt dem Umstand Rechnung.

Die folgenden Punkte sollten vor einer Installation beachtet werden:

Hinweis: Für die Umsetzung dieses Artikel sollte beim Anwender eine größere Erfahrung mit Linux vorhanden sein.

- **NFSv4**: Bei dem Artikel **Mkinitcpio** unter dem Punkt #Using_net (https://wiki.archlinux.org/index.php/Mkinitcpio#Using_net) bringt dieses Ergebnis:

Achtung: NFSv4 is not yet supported.

- (Stand: 28.06.2013)
  Somit bleibt dieser Punkt unberücksichtigt, bis eine Änderung eingetreten ist. Wenn jemand *NFSv4* einsetzen will, auch wenn das Paket *mkinitcpio-nfs-utils* für `net` installiert ist, wird spätestens beim Bau von *initramfs-linux.img* und *initramfs-linux-fallback.img* mit *Mkinitcpio* Probleme bekommen.

- **mkarchroot** wird genauer erläutert. Von diesem Befehl gibt es keine manpage nur die „**-h**“ Hilfeausgabe.

- **pacstrap**: Hier gilt das gleiche wie für *mkarchroot*.

- **Verzeichnisstruktur**: Beim Anlegen sollte folgendes beachtet werden:
  „**Viele ziehen es vor, das Boot-System in `/var/lib/tftpboot` anzulegen, und unter `/srv/nfs` anschließend die Dateisysteme der Ziel-Rechner anzulegen, allerdings hat dies den Nachteil, das vom Clientsystem aus anschließend kein Kernelupdate durchgeführt werden kann. Der Vorteil ist eine erhöhte Sicherheit, da man via TFTP auf sämtliche Unterverzeichnisse Zugriff hat! Dies ist abzuwägen anhand der eigenen Paranoia. Sollten in einem Netz viele Fremde unterwegs sein, ist es im Interesse der Sicherheit anzuraten, nicht `/srv/nfs` auch für den Bootloader zu verwenden!**“
  Das Zitat stammt aus dem **PXE**-Artikel

- **Rechner-Architektur**: Für jede (i686, x86_64, PowerPC, AMD ....) empfiehlt es sich ein eigenes Verzeichnis für die Ziel-Rechner anzulegen. Es wird gezeigt wie dies mit einer entsprechend angepassten pacman.conf erreicht werden kann. Aufgrund der obigen Informationen wird das **Boot**-System im Verzeichnis `/srv/tftp` installiert. Die Dateisysteme für die Ziel-Rechner werden im **Root**-Verzeichnis `/srv/arch/linux_i686` für die i686-Architektur installiert.

Inhaltsverzeichnis

Die Server Konfiguration

Zunächst einmal muss man die folgenden Komponenten installieren:

Einen DHCP Server um den Ziel-Rechnern IP Adressen zuweisen zu können.

Einen TFTP Server um die Bootimages zu den Ziel-Rechnern übertragen zu können. (Eine optionale Voraussetzung für alle PXE ROMs).

Eine Art von Netzwerkspeicher (entweder NFS oder NBD) um die Arch Installation zu den Ziel-Rechnern exportieren zu können.

Hinweis: `dnsmasq (https://www.archlinux.de/?page=Packages&architecture=&search=dnsmasq)` kann gleichzeitig sowohl als DHCP- und als TFTP-Server eingesetzt werden. Weitere Informationen sind im Artikel dnsmasq zu finden.

DHCP Server

ISC `dhcp (https://www.archlinux.de/?page=Packages&architecture=&search=dhcp)` muss auf dem Server installiert und konfiguriert werden.

```
# vim /etc/dhcpd.conf
```

```
allow booting;
allow bootp;

authoritative;

option domain-name-servers 10.0.0.1;

option architecture code 93 = unsigned integer 16;

group {
    next-server 10.0.0.1;

    if option architecture = 00:07 {
        filename "/grub/x86_64-efi/core.efi";
    } else {
        filename "/grub/i386-pc/core.0";
    }

    subnet 10.0.0.0 netmask 255.255.255.0 {
        option routers 10.0.0.1;
        range 10.0.0.128 10.0.0.254;
    }
```

Hinweis: `next-server` sollte die IP-Adresse des TFTP-Servers sein; alles andere sollte entsprechend geändert werden, damit es zum eigenen Netzwerk passt.

Hinweis: `filename "/grub/i386-pc/core.0";` Diese Zeile kann sich in Bezug auf den Boot-Loader #Pxelinux noch verändern.

RFC 4578 definiert die „Art der System Architektur des Ziel-Rechners“ in der DHCP Option. In der obigen Konfiguration, sollte der PXE Ziel-Rechner ein x86_64-efi Bootimage (Typ 0x7) anfordern, wird ihm das passende übertragen, ansonsten wird auf ein i686 Bootimage zurück gegriffen. Dies erlaubt beides, sowohl einem UEFI Ziel-Rechner als auch einem Ziel-Rechner mit älterem BIOS simultan über das selbe Netzwerksegment zu booten.

Man muss den ISC DHCP per systemd starten.

```
# systemctl start dhcpd4.service
```

Um zu überprüfen, ob der Service erfolgreich gestartet werden konnte wird folgende Eingabe benötigt:

```
# systemctl status dhcpd4.service
```

Hier sollte alles im grünen Bereich sein. (Wie das Beispiel zeigt:)

```
# systemctl status dhcpd4.service
dhcpd4.service - IPv4 DHCP server
   Loaded: loaded (/usr/lib/systemd/system/dhcpd4.service; disabled)
   Active: active (running) since Fr 2013-06-28 23:03:06 CEST; 17s ago
  Process: 419 ExecStart=/usr/bin/dhcpd -4 -q -pf /run/dhcpd4.pid (code=exited, status=0/SUCCESS)
 Main PID: 420 (dhcpd)
   CGroup: name=systemd:/system/dhcpd4.service
           └─420 /usr/bin/dhcpd -4 -q -pf /run/dhcpd4.pid
```

TFTP-Server

`tftp-hpa (https://www.archlinux.de/?page=Packages&architecture=&search=tftp-hpa)` muss auf dem Server installiert und konfiguriert werden.

Der TFTP-Server wird gebraucht um den Bootloader, den Kernel, und das initramfs zum Ziel-Rechner zu übertragen.

Die Datei sollte entsprechend diesem Muster erstellt werden:

```

[Unit]
Description=hpa's original TFTP daemon

[Service]
ExecStart=/usr/bin/in.tftpd -s /srv/tftp/
StandardInput=socket
StandardOutput=inherit
StandardError=journal
```

Die erstellte Datei wird auf dem Server im Pfad `/etc/systemd/system/` unter dem Namen `tftpd.service` gespeichert.

Der TFTP-Server muss per systemd gestartet werden.

```
# systemctl start tftpd.socket tftpd.service
```

Eine Überprüfung lohnt sich immer:

```
# systemctl status tftpd.socket
# systemctl status tftpd.service
```

Das ganze sollte so ausssehen:

```
tftpd.socket
Loaded: loaded (/usr/lib/systemd/system/tftpd.socket; disabled)
        Active: active (listening) since Fr 2013-06-28 23:09:01 CEST; 27s ago
        Listen: [::]:69 (Datagram)
```

```
tftpd.service - hpa's original TFTP daemon
Loaded: loaded (/etc/systemd/system/tftpd.service; static)
        Active: active (running) since Fr 2013-06-28 23:09:01 CEST; 14s ago
        Main PID: 425 (in.tftpd)
        CGroup: name=systemd:/system/tftpd.service
                └─425 /usr/bin/in.tftpd -s /srv/arch/boot
```

Netzwerkspeicher

Der primäre Unterschied zwischen der Verwendung von NFS und NBD ist, während es mit beiden in der Tat möglich ist, das mehrere Ziel-Rechner die selbe Installation nutzen können, mit NBD jedoch (aufgrund der direkten Art der Manipulation des Dateisystems) wird der `copyonwrite` Modus benötigt, um dies zu erreichen, was zum Verwerfen jeglicher Schreibzugriffe führt, wenn sich der Ziel-Rechner abmeldet. In einigen Situationen könnte dies jedoch gewünscht sein.

NFS Server

`nfs-utils (https://www.archlinux.de/?page=Packages&architecture=&search=nfs-utils)` muss auf dem Server installiert und konfiguriert werden.

NFSv3

Das Root-Verzeichnis der jeweiligen arch-Installation muss in die **NFS** exports-Datei eintragen werden.

```
# vim /etc/exports
```

```
/srv/arch/linux_i686 *(rw,no_root_squash,no_subtree_check,sync)
```

Hinweis: Ist man nicht über Datenverlust im Falle von Netzwerk- und / oder Server-Ausfällen besorgt, ersetzt man sync durch async --zusätzliche Optionen finden sich im *NFS* Artikel. Die Option sync bietet sich an wenn man bei Schreibzugriffen sicher stellen möchte, das die Daten korrekt gespeichert werden, diese Einstellung geht jedoch zu Lasten der Geschwindigkeit. Die Option async sollte bei ausschließlich lesenden Zugriffen benutzt werden, da dies die Geschwindigkeit steigert, die Sicherheit kann in diesem Fall vernachlässigt werden.

Als Nächstes muss man den NFSv3 (https://wiki.archlinux.org/index.php/NFSv3) Server starten:

```
# systemctl start rpcbind.service
# systemctl start nfs-server.service
# systemctl start rpc-statd.service
```

Die Überprüfung:

```
# systemctl status rpcbind.service
# systemctl status nfs-server.service
# systemctl status rpc-statd.service
```

Das Ergebnis:

```
● rpcbind.service - RPC bind service
```

```
Loaded: loaded (/usr/lib/systemd/system/rpcbind.service; indirect; vendor preset: disabled)
   Active: active (running) since Ta Jahr-Mo-Ta YY:ZZ:VV CEST; 2h 33min ago
 Main PID: 446 (rpcbind)
   CGroup: /system.slice/rpcbind.service
           └─446 /usr/bin/rpcbind -w

Mon XX YY:ZZ:VV server.local systemd[1]: Starting RPC bind service...
Mon XX YY:ZZ:VV server.local systemd[1]: Started RPC bind service.
```

```
● nfs-server.service - NFS server and services
```

```
Loaded: loaded (/usr/lib/systemd/system/nfs-server.service; enabled; vendor preset: disabled)
   Active: active (exited) since Ta Jahr-Mo-Ta YY:ZZ:VV CEST; 2h 40min ago
 Main PID: 452 (code=exited, status=0/SUCCESS)
   CGroup: /system.slice/nfs-server.service

Mon XX YY:ZZ:VV server.local systemd[1]: Starting NFS server and services...
Mon XX YY:ZZ:VV server.local systemd[1]: Started NFS server and services.
```

```
● rpc-statd.service - NFS status monitor for NFSv2/3 locking.
```

```
Loaded: loaded (/usr/lib/systemd/system/rpc-statd.service; static; vendor preset: disabled)
   Active: active (running) since Ta Jahr-Mo-Ta YY:ZZ:VV CEST; 2h 43min ago
 Main PID: 448 (rpc.statd)
   CGroup: /system.slice/rpc-statd.service
           └─448 /usr/sbin/rpc.statd --no-notify

Mon XX YY:ZZ:VV server.local systemd[1]: Starting NFS status monitor for NFSv2/3 locking....
Mon XX YY:ZZ:VV server.local rpc.statd[448]: Version 1.3.2 starting
Mon XX YY:ZZ:VV server.local rpc.statd[448]: Flags: TI-RPC
Mon XX YY:ZZ:VV server.local rpc.statd[448]: Running as root. chown /var/lib/nfs to choose different user
Mon XX YY:ZZ:VV server.local systemd[1]: Started NFS status monitor for NFSv2/3 locking..
```

NBD

`nbd (https://www.archlinux.de/?page=Packages&architecture=&search=nbd)` muss auf dem Server installiert und konfiguriert werden.

```
# vim /etc/nbd-server/config
```

```
[generic]
```

```
    user = nbd
    group = nbd
[arch]
    exportname = /srv/arch.img
    copyonwrite = false
```

Hinweis: Man muss `copyonwrite` auf „true“ setzen falls mehrere Ziel-Rechner das NBD Verzeichnis gleichzeitig nutzen sollen; man gebe `man 5 nbd-server` ein für mehr Details zur Ausgabe auf der Konsole.

nbd muss per systemd gestartet werden.

```
# systemctl start nbd.service
```

Die Prüfung:

```
# systemctl status nbd.service
```

Das Ergebnis:

```

(Wer NBD benutzt trage bitte hier die entsprechende Konsolenausgabe nach den obigen Beispielen ein.)
```

Bisher wurden die Server/Dienste erst ein Mal „nur“ gestartet. Damit diese beim Booten aktiviert werden muss man „start“ durch „enable“ ersetzen. Um dieses Setup zu testen reicht ein starten zunächst aus. Läuft alles wie gewünscht können die Server/Dienste dann beim Booten aktiviert werden.

Zielrechner Installation / Konfiguration

Als nächstes muss eine komplette Arch Linux Installation in einem Unterverzeichnis auf dem Server einrichten werden. Während der Startphase wird der Ziel-Rechner eine IP-Adresse vom DHCP-Server erhalten, dann wird von diesem mit **PXE** ein Kernel gebootet und mountet diese Installation als **Root**-Verzeichnis.

Root-Verzeichnis für Ziel-Rechner einrichten

Informationen zum nachfolgend genannten btrfs (https://wiki.archlinux.org/index.php/Btrfs) Dateisystem gibt es auf der entsprechenden Site.

Demnach werden viele Anwender dieses Dateisystem zuerst auf Ihren Systemen installieren müssen, da es erst ab Anfang 2013 in den Standard-Kernel eingeflossen ist. Wenn man gerne dieses neu in ArchLinux eingeführte Dateisystem nutzen möchten.

Man muss eine Sparse-Datei (https://de.wikipedia.org/wiki/Sparse-Datei) von mindestens 1 Gigabyte erstellen, und diese dann mit einem **btrfs** Dateisystem formatieren.

(Es kann natürlich auch ein echtes Block-Device oder ein LVM einrichtet werden, wenn dies gewünscht wird).

Des weiteren soll hier von der Einrichtung auf einem Block-Device ausgegangen werden.

```
# truncate -s 1G /srv/arch.img
# mkfs.btrfs /srv/arch.img
# export root=/srv/arch/linux_i686
# mkdir -p "$root"
# mount -o loop,discard,compress=lzo /srv/arch.img "$root"
```

Hinweis: Das Erstellen eines separaten Dateisystems ist für **NBD** zwingend nötig aber optional für **NFS** und kann übersprungen / ignoriert werden.

Wie oben unter **Rechner-Architektur** erwähnt, soll das **Root**-System im Verzeichnis `/srv/arch/linux_i686` installiert werden.

Bootstrapping Installation

Es müssen die `devtools (https://www.archlinux.de/?page=Packages&architecture=&search=devtools)` und `arch-install-scripts (https://www.archlinux.de/?page=Packages&architecture=&search=arch-install-scripts)` Pakete installiert werden und `mkarchroot` ausgeführt werden.

Es folgt die Ausgabe des Befehls mkarchroot **-h**:

```
# mkarchroot -h
```

```
Usage: mkarchroot [options] working-dir [package-list | app]
 options:
    -C <file>     Location of a pacman config file
    -M <file>     Location of a makepkg config file
    -c <dir>      Set pacman cache
    -h            This message
```

Hier ist die Option -C bedeutend: Damit lässt sich eine alternative `pacman.conf` angeben. In dieser kann eine für den oder die Ziel-Rechner passende Architektur angeben werden. Gerade wenn die Ziel-Rechner eine andere Architektur als der Server aufweist Die jeweilige Zeile ist farblich hinterlegt.

Ein Ausschnitt aus der pacman.conf:

```
#
# /etc/pacman.conf
#
# See the pacman.conf(5) manpage for option and repository directives

#
# GENERAL OPTIONS
#
[options]
# The following paths are commented out with their default values listed.
# If you wish to use different paths, uncomment and update the paths.
#RootDir     = /
#DBPath      = /var/lib/pacman/
#CacheDir    = /var/cache/pacman/pkg/
#LogFile     = /var/log/pacman.log
#GPGDir      = /etc/pacman.d/gnupg/
HoldPkg     = pacman glibc
#XferCommand = /usr/bin/curl -C - -f %u > %o
#XferCommand = /usr/bin/wget --passive-ftp -c -O %o %u
#CleanMethod = KeepInstalled
#UseDelta    = 0.7
Architecture = auto

# Pacman won't upgrade packages listed in IgnorePkg and members of IgnoreGroup
#IgnorePkg   =
#IgnoreGroup =

#NoUpgrade   =
#NoExtract   =
[...]
```

Hier ist die Zeile mit `Architecture = auto` relevant. Mit dem Wert `auto` wird die selbe Architektur verwendete wie auf dem Server auch. Die Datei wird in einen Editor geladen und die Architektur für den Ziel-Rechner eingetragen. Im Falle einer i686-Architektur lautet der Eintrag: `Architecture = i686`. Die Datei sollte unter dem Name abspeichern werden, der die Architektur trägt, z. B. `pacman_arch_i686.conf`.

Hier die angepasste Datei:

Der Ausschnitt aus der pacman.conf:

```
#
# /etc/pacman.conf
#
# See the pacman.conf(5) manpage for option and repository directives

#
# GENERAL OPTIONS
#
[options]
# The following paths are commented out with their default values listed.
# If you wish to use different paths, uncomment and update the paths.
#RootDir     = /
#DBPath      = /var/lib/pacman/
#CacheDir    = /var/cache/pacman/pkg/
#LogFile     = /var/log/pacman.log
#GPGDir      = /etc/pacman.d/gnupg/
HoldPkg     = pacman glibc
#XferCommand = /usr/bin/curl -C - -f %u > %o
#XferCommand = /usr/bin/wget --passive-ftp -c -O %o %u
#CleanMethod = KeepInstalled
#UseDelta    = 0.7
Architecture = i686

# Pacman won't upgrade packages listed in IgnorePkg and members of IgnoreGroup
#IgnorePkg   =
#IgnoreGroup =

#NoUpgrade   =
#NoExtract   =
[...]
```

Dann muss man in das Verzeichnis `/srv/arch/` wechseln.

Der Aufruf von `mkarchroot` lautet:

```
# mkarchroot -C pacman_arch_i686.conf linux_i686
```

Mit diesem Befehl wird das Verzeichnis `linux_i686` angelegt und die Pakete aus der Gruppe „**base**“ für die Architektur i686 installiert. Diese Gruppe ist der default Wert für `mkarchroot`. Sollte das Verzeichnis bereits existieren wird der Befehl mit einer entsprechenden Meldung abgebrochen. Es ist auch folgender Aufruf möglich in Anbetracht der obigen Zeile `export root=/srv/arch/linux_i686`:

```
# mkarchroot -C pacman_arch_i686.conf "$root"
```

Je nachdem von welchem Verzeichnis aus man gerne installiert.

Zunächst einige Infos zu dem Befehl `pacstrap`:

```
# pacstrap -h
```

```
usage: pacstrap [options] root [packages...]

  Options:
    -C config      Use an alternate config file for pacman
    -c             Use the package cache on the host, rather than the target
    -d             Allow installation to a non-mountpoint directory
    -G             Avoid copying the host's pacman keyring to the target
    -i             Avoid auto-confirmation of package selections
    -M             Avoid copying the host's mirrorlist to the target

    -h             Print this help message

pacstrap installs packages to the specified new root directory. If no packages
are given, pacstrap defaults to the "base" group.
```

Da die Pakete aus der Gruppe „*base*“ schon mit dem Befehl `mkarchroot` installiert wurde, sollte sie ausgelassen werden. Auch hier, wie beim vorherigen Befehl ist es möglich mit der Option -C eine Alternative pacman.conf an zu geben. Somit kann die vorhandene Datei `pacman_arch_i686.conf` wieder verwendet werden.

Der Befehl setzt sich wie folgt zusammen:

```
# pacstrap -C pacman_arch_i686.conf -d "$root" mkinitcpio-nfs-utils nfs-utils
```

Es wird die genannte `*.conf` Datei verwendet, die dafür sorgt das Pakete für die i686-Architektur herunter geladen werden, damit diese dann in ein nicht gemountetes Verzeichnis installiert werden. Zum Schluss werden die beiden Pakete genannt. Sollte das Installationsverzeichnis schon gemountet sein, entfällt die Option -d.

> **Hinweis:** In allen Fällen ist das Paket `mkinitcpio-nfs-utils (https://www.archlinux.de/?page=Packages&architecture=&search=mkinitcpio-nfs-utils)` weiterhin erforderlich. `ipconfig` wird im frühen Bootprozess nur verwendet wenn dies vorgesehen ist.

Nun wird das initramfs aufgebaut.

NFSv3

Die Datei `mkinitcpio.conf` muss konfiguriert werden.

```
# vim "$root/etc/mkinitcpio.conf"
```

```
MODULES="nfsv3"
HOOKS="base udev autodetect net filesystems keyboard"
BINARIES=""
```

> **Hinweis:** Außerdem müssen die entsprechend benötigten module (https://wiki.archlinux.org/index.php/Network#Device_Driver) für den Ethernet-Controller dem `MODULES` Feld hinzufügen werden.

Die Datei `mkinitcpio.conf` mit den entsprechenden beispielhaften Einträgen. Die relevanten Bereiche sind farblich hervorgehoben.

Äußerst wichtig: Die Module, die hinter `MODULES=` eingetragen werden, sind für den Ziel-Rechner und nicht für den Server!!!

```
mkinitcpio.conf
```

```
# vim:set ft=sh
# MODULES
# The following modules are loaded before any boot hooks are
# run.  Advanced users may wish to specify all system modules
# in this array. For instance:
#     MODULES="piix ide_disk reiserfs"
```

```
MODULES="nfsv3 8139too ipw2200" # bitte entsprechend anpassen.

# BINARIES
# This setting includes any additional binaries a given user may
# wish into the CPIO image. This is run last, so it may be used to
# override the actual binaries included by a given hook
# BINARIES are dependency parsed, so you may safely ignore libraries
BINARIES=""

# FILES
# This setting is similar to BINARIES above, however, files are added
# as-is and are not parsed in any way. This is useful for config files.
FILES=""

# HOOKS
# This is the most important setting in this file. The HOOKS control the
# modules and scripts added to the image, and what happens at boot time.
# Order is important, and it is recommended that you do not change the
# order in which HOOKS are added. Run 'mkinitcpio -H ' for
# help on a given hook.
# 'base' is _required_ unless you know precisely what you are doing.
# 'udev' is _required_ in order to automatically load modules
# 'filesystems' is _required_ unless you specify your fs modules in MODULES
# Examples:
##   This setup specifies all modules in the MODULES setting above.
##   No raid, lvm2, or encrypted root is needed.
#    HOOKS="base"
#
##   This setup will autodetect all modules for your system and should
##   work as a sane default
HOOKS="base udev autodetect net filesystems keyboard"
#
##   This setup will generate a 'full' image which supports most systems.
##   No autodetection is done.
#    HOOKS="base udev block filesystems"
#
##   This setup assembles a pata mdadm array with an encrypted root FS.
##   Note: See 'mkinitcpio -H mdadm' for more information on raid devices.
#    HOOKS="base udev block mdadm encrypt filesystems"
#
##   This setup loads an lvm2 volume group on a usb device.
#    HOOKS="base udev block lvm2 filesystems"
#
##   NOTE: If you have /usr on a separate partition, you MUST include the
#    usr, fsck and shutdown hooks.
#    HOOKS="base udev autodetect modconf block filesystems keyboard fsck"

# COMPRESSION
# Use this to compress the initramfs image. By default, gzip compression
# is used. Use 'cat' to create an uncompressed image.
#COMPRESSION="gzip"
#COMPRESSION="bzip2"
#COMPRESSION="lzma"
#COMPRESSION="xz"
#COMPRESSION="lzop"

# COMPRESSION_OPTIONS
# Additional options for the compressor
#COMPRESSION_OPTIONS=""
```

Das initramfs wird nun installiert. Der einfachste Weg dies zu tun ist mit dem Befehl `arch-chroot`.

```
# arch-chroot "$root" /bin/bash
(chroot) # mkinitcpio -g /srv/tftp/linux_i686.img
(chroot) # exit
```

Wie oben schon unter **Rechner-Architektur** erwähnt soll das **Boot**-System im Verzeichnis `/srv/tftp/` installiert werden.

```
Hier eine mögliche Ausgabe des build-Prozesses:
```

```
==> Building image from preset: /etc/mkinitcpio.d/linux.preset: 'default'
  -> -k /srv/tftp/vmlinuz-linux_i686 -c /etc/mkinitcpio.conf -g /srv/tftp/initramfs-linux_i686.img
==> Starting build: 3.9.7-1-ARCH
  -> Running build hook: [base]
  -> Running build hook: [udev]
  -> Running build hook: [autodetect]
  -> Running build hook: [net]
  -> Running build hook: [filesystems]
  -> Running build hook: [keyboard]
==> Generating module dependencies
==> Creating gzip initcpio image: /srv/tftp/initramfs-linux_i686.img
==> Image generation successful
==> Building image from preset: /etc/mkinitcpio.d/linux.preset: 'fallback'
  -> -k /srv/tftp/vmlinuz-linux_i686 -c /etc/mkinitcpio.conf -g /srv/tftp/initramfs-linux-fallback_i686.img -S autodetect
==> Starting build: 3.9.7-1-ARCH
  -> Running build hook: [base]
  -> Running build hook: [udev]
  -> Running build hook: [net]
==> WARNING: Possibly missing firmware for module: bna
==> WARNING: Possibly missing firmware for module: b43legacy
==> WARNING: Possibly missing firmware for module: b43
==> WARNING: Possibly missing firmware for module: atmel
==> WARNING: Possibly missing firmware for module: at76c50x_usb
==> WARNING: Possibly missing firmware for module: prism54
==> WARNING: Possibly missing firmware for module: zd1201
```

```
==> WARNING: Possibly missing firmware for module: zd1211rw
==> WARNING: Possibly missing firmware for module: orinoco_usb
==> WARNING: Possibly missing firmware for module: wl1251
==> WARNING: Possibly missing firmware for module: rtl8723ae
==> WARNING: Possibly missing firmware for module: ipw2100
==> WARNING: Possibly missing firmware for module: ipw2200
==> WARNING: Possibly missing firmware for module: p54pci
==> WARNING: Possibly missing firmware for module: p54usb
  -> Running build hook: [filesystems]
  -> Running build hook: [keyboard]
==> Generating module dependencies
==> Creating gzip initcpio image: /srv/tftp/initramfs-linux-fallback_i686.img
==> Image generation successful
```

NBD

mkinitcpio-nbd (https://aur.archlinux.org/packages/mkinitcpio-nbd)[AUR] muss auf dem Ziel-Rechner installiert werden. Man muss es mit **makepkg** bauen und installieren:

```
# pacman --root "$root" --dbpath "$root/var/lib/pacman" -U mkinitcpio-nbd-0.4-1-any.pkg.tar.xz
```

Hinweis: Die Pfad-Variable `"$root"` muss hier auf `/srv/arch/linux_i686` gesetzt sein.

Dann muss `nbd` in das `HOOKS` Feld nach `net` hinzufügt werden; `net` wird das Netzwerk konfigurieren, aber nicht versuchen ein NFS-Mount zu machen, wenn das `nfsroot` nicht in der Kernel-Zeile angegeben wird.

Zielrechner Konfiguration (auf dem Server)

Zusätzlich zu dem hier genannten Setup sollte

- hostname (https://wiki.archlinux.org/index.php/HOSTNAME#Set_the_host_name),
- timezone (https://wiki.archlinux.org/index.php/Timezone#Time_Zone),
- locale,
- keymap (https://wiki.archlinux.org/index.php/KEYMAP),

konfiguriert werden, man befolge alle anderen relevanten Teile der Arch Install Scripts.

Eine ausführlichere Alternative ist der Installation Guide (https://wiki.archlinux.org/index.php/Installation_Guide#Connect_to_the_internet).

Ebenfalls ist die `fstab` zu erwähnen. Diese sollte so ähnlich aussehen:

/etc/fstab

```
#
# /etc/fstab: static file system information
#
# <file system>         <dir>                         <type>  <options>                     <dump>  <pass>
tmpfs                   /tmp                          tmpfs   nodev,nosuid                  0       0

proc                    /proc                         proc    defaults                      0       0

none / none
             # <-- hier ist die Leerzeile wichtig!
```

Mit dem Befehl `genfstab` sollte man in diesem Fall äußerst vorsichtig sein. Dies kann einer der Gründe sein warum der Ziel-Rechner nicht bootet. Dazu sieht man am #Ende des Wikis nach, genauer bei „Im Fehlerfall...“.

Nun ist eine gute Gelegenheit zu erwähnen, dass die Konfiguration für **eine** Architektur beendet ist. Sollen noch weitere Ziel-Rechner mit einer anderen Architektur von diesem Server booten, sollte dafür ein eigenes **Root**-Verzeichnis angelegt werden und die dazu passenden Kernel mit aussagekräftigen Namen im **Boot**-Verzeichnis gespeichert werden. Dazu ist es notwendig zu dieser Stelle #Zielrechner Installation / Konfiguration zurück zu gehen, um die Schritte dafür zu wiederholen.

Die verschiedenen Bootloader

GRUB

Obwohl schlecht dokumentiert, unterstützt **GRUB** auch das Laden über *PXE*. Aufgrund eines Fehlers in Endian (https://savannah.gnu.org/bugs/?36755) in der `grub-core/net/tftp.c` Datei der nicht in der bzr Revision 4548 (http://bzr.savannah.gnu.org/lh/grub/trunk/grub/revision/4548) behoben wurde (`grub-gfx (https://www.archlinux.de/?page=Packages&architecture=&search=grub-gfx)` 2.00 ist Revision 4542 (http://bzr.savannah.gnu.org/lh/grub/trunk/grub/revision/4542)), muss man das burg-bios-bzr (https://aur.archlinux.org/packages/burg-bios-bzr)[AUR] aus dem AUR selber bauen; es macht am meisten Sinn, dies in der Ziel-Rechner-

Installation zu tun. Darüber hinaus ist es aufgrund eines Fehlers in `grub-core/net/drivers/efi/efinet.c` der nicht behoben wurde bis zur bzr Revision 4751 (http://bzr.savannah.gnu.org/lh/grub/trunk/grub/revision/4751), muss burg-efi-x86_64-bzr (https://aur.archlinux.org/packages/burg-efi-x86_64-bzr)[AUR] ebenfalls erstellt werden. Der Arch-User-Repository (https://wiki.archlinux.org/index.php/Arch_User_Repository) Artikel beschreibt, wie diese Pakete gebaut werden.

```
# pacman --root "$root" --dbpath "$root/var/lib/pacman" -U grub-bios-bzr-4751-1-x86_64.pkg.tar.xz
```

Hinweis: Die Pfad-Variable `"$root"` muss hier auf `/srv/arch/linux_i686` gesetzt sein.

Es soll die `burg-bios-bzr` Installation auf dem Ziel-Rechner genutzt werden, daher macht man ein `arch-chroot` in die Ziel-Rechner Installation damit man den Befehl `grub-mknetdir` benutzen kann.

```
# arch-chroot /srv/tftp grub-mknetdir --subdir=grub
```

Als nächstes wird zu burg-efi-x86_64-bzr (https://aur.archlinux.org/packages/burg-efi-x86_64-bzr)[AUR] gewechselt das eben gebaut wurde, und es wird `grub-mknetdir` ausgeführt, genau so wie oben, nur zum zweiten Mal.

```
# pacman --root "$root" --dbpath "$root/var/lib/pacman" -U grub-bios-bzr-4751-1-x86_64.pkg.tar.xz
```

Hinweis: Die Pfad-Variable `"$root"` muss hier auf `/srv/arch/linux_i686` gesetzt sein.

Nun erstelle man eine einfache GRUB-Konfiguration:

```
# vim "/srv/tftp/grub/grub.cfg"
```

```
menuentry "Arch Linux" {
    linux tftp/vmlinuz-linux_i686 quiet add_efi_memmap ip=:::::eth0:dhcp nfsroot=10.0.0.1:/srv/arch/linux_i686
    initrd tftp/initramfs-linux_i686.img
}
```

GRUB wird das Root-Verzeichnis `root=(tftp,10.0.0.1)` automatisch eintragen, so das der Kernel und das initramfs per TFTP übertragen werden, ohne zusätzliche Angaben zur Konfiguration, jedoch sollte man diese ausdrücklich angeben, wenn man weitere Nicht-TFTP Menü-Einträge machen möchtest.

Hinweis: Die Kernel Zeile sollte man anpassen wie es erforderlich ist, in Bezug auf #Pxelinux (https://wiki.archlinux.org/index.php/PXE) für NBD-relevante Optionen.

Pxelinux

Hinweis: *Syslinux* besitzt derzeit keinen UEFI-Netzwerk-Stack, so dass man nicht in der Lage sein wird syslinux-firmware-git (https://aur.archlinux.org/packages/syslinux-firmware-git)[AUR] zu benutzen. Es ist noch immer möglich mit #GRUB und TFTP den Kernel und das initramfs zu übertragen; *pxelinux* funktioniert gut mit älteren PXE Boot ROMs.

Pxelinux wird bereitgestellt durch `syslinux (https://www.archlinux.de/?page=Packages&architecture=&search=syslinux).`

Der *pxelinux* Bootloader (aus dem Paket *syslinux*) wird in das Boot-Verzeichnis des Ziel-Rechners kopiert.

```
# cp /usr/lib/syslinux/pxelinux.0 "/srv/tftp"
# mkdir "/srv/tftp/pxelinux.cfg"
```

Zugleich legt man auch das pxelinux.cfg Verzeichnis an, wo **pxelinux** standardmäßig nach seinen Konfigurationsdateien sucht. Es soll nicht zwischen verschiedenen Host-MAC-Adressen unterscheiden werden, daher erstellen wir die Standardkonfiguration.

```
# vim "/srv/tftp/pxelinux.cfg/default"
```

```
DEFAULT linux

LABEL linux
  MENU LABEL Arch Linux i686
  KERNEL vmlinuz-linux_i686
  APPEND initrd=initramfs-linux_i686.img quiet ip=:::::eth0:dhcp nfsroot=10.0.0.1:/arch
```

NFSv3 Mountpoints sind relativ zum Root des Servers nicht `fsid=0`. (`fsid=0` wird nur bei dem noch nicht unterstützten NFSv4 angegeben.)

Wenn *NFSv3* verwendet wird, ist es notwendig `10.0.0.1:/arch` durch `10.0.0.1:/srv/arch/linux_i686` bei `nfsroot=` zu ersetzen.

Oder wenn *NBD* verwendet wird, wird die folgende „append“-Zeile benötigt:

```
append ro initrd=initramfs-linux_i686.img ip=:::::eth0:dhcp nbd_host=10.0.0.1 nbd_name=arch/linux_i686 root=/dev/nbd0
```

Hinweis: Es muss `nbd_host` und/oder `nfsroot` geändert werden, auf die jeweils passende Netzwerk-Konfiguration (die Adresse des NFS / NBD-Servers)

Die *pxelinux* Konfigurationssyntax ist identisch mit der von *syslinux*; man ziehe die obige Dokumentation für weitere Informationen zu Rate.

Die einzelnen Parameter werden in dem *PXE*-Artikel beschrieben.

Der Kernel und das initramfs wird per TFTP übertragen, so dass die Pfade zu diesen relativ sein werden, bezogen auf das TFTP Root-Verzeichnis. Andernfalls wird das Root-Dateisystem zum NFS mount selbst werden, also relativ zum Root-Verzeichnis des NFS-Servers.

Um tatsächlich *pxelinux* zu laden, ersetzt man `filename "/grub/i386-pc/core.0";` in der Datei `/etc/dhcpd.conf` durch `filename "/pxelinux.0";`

zusätzliche Mountpunkte

NBD Root-Verzeichnis

Im späteren Bootvorgang ist es nötig das Root-Dateisystem zu beidem zu ändern, um es sowohl schreibbar zu machen `rw` und die Aktivierung von `compress=lzo` für eine deutlich verbesserte Festplatten-Performance im Vergleich zu *NFS*.

Hinweis: Die Pfad-Variable `"$root"` muss hier auf `/srv/arch/linux_i686` gesetzt sein.

Das Ergebnis

Bei einem ersten Test des obigen Setups sollte eine ähnliche Ausgabe zu sehen sein:

Der ganz normale Konsolen-Login.

```
Arch Linux 3.9.8-1-ARCH (tty1)

(Rechner Name) login:
```

Im Fehlerfall gibt es eine solche Ausgabe:

```
(MAC- und IP-Adressen geändert)
```

```
Intel UNDI, PXE-2.0 (build 082)
Copyright (C) 1997-2000 Intel Corporation
For Realtek RTL8139(X)/8130/810X PCI Fast Ethernet Controller u2.13 (020326)
CLIENT MAC ADDR: 00 00 00 00 00 00 GUID: (gelöscht)
CLIENT IP: 192.000.000.000 MASK: 255.255.255.0 DHCP IP: 192.000.000.000
GATEWAY IP: 192.000.000.000
PXELINUX 4.06 2012-10-23 Copyright (C) 1991-2012 H. Peter Anvin et al
!PXE entry point found (we hope) at 9E1A:00F9 via plan 0
UNDI code segment at 9E1A len 132B
UNDI data segment at 9877 len 5A30
Getting cached packet 01 02 03
My IP address seems to be (gelöscht) 192.000.000.000
ip=192.000.000.000:192.000.000.000:192.000.000.000:255.255.255.0
BOOTIF=01-(gelöscht)
SYSUUID=(gelöscht)
TFTP prefix:
Trying to load: pxelinux.cfg/default                                   ok
early console in decompress_kernel
Decompressing Linux... Parsing ELF... done.
Booting the kernel.
IP-Config: eth0 hardware address 00:00:00:00:00:00 mtu 1500 DHCP
IP-Config: eth0 guessed broadcast address 192.000.000.000
IP-Config: eth0 complete (from 192.000.000.000):
address: 192.000.000.000 broadcast: 192.000.000.000 netmask: 255.255.255.0
gateway: 192.000.000.000 dns0     : 0.0.0.0         dns1   : 0.0.0.0
```

```
rootserver: 192.000.000.000 rootpath:
filename: pxelinux.0
NFS-Mount: 192.000.000.000:/srv/arch/linux_i686
[ 16.635244] irq 18: nobody cared (try booting with the "irqpoll" option)
[ 16.635786] handlers:
[ 16.635865] [] usb_hcd_irq [usbcore]
[ 16.635967] Disabling IRQ #18
Welcome to emergency mode! After logging in. type "journalctl -xb to view
system logs. "systemctl reboot" to reboot, "systenctl default" to try again
to boot into default node.
Giue root password for maintenance
(or type Control-D to continue):
```

Um die Zeilen mit dem IRQ #18 zu eliminieren wird die angegebene Option `irqpoll` der passendem Zeile beim jeweiligen Bootloader eingefügt.

Bei *GRUB* ist dies:

```
linux tftp/vmlinuz-linux_i686 quiet add_efi_memmap ip=:::::eth0:dhcp irqpoll nfsroot=10.0.0.1:/srv/arch/linux_i686
```

also der Kernel-Zeile.

Bei *Pxelinux* ist dies:

```
append initrd=initramfs-linux_i686.img quiet ip=:::::eth0:dhcp irqpoll nfsroot=10.0.0.1:/arch
```

also die „append“-Zeile.

In diesem konkreten Fall lag der Fehler in einer falschen `fstab`. Siehe dazu: #Zielrechner Konfiguration (auf dem Server). Kann diese Möglichkeit ausgeschlossen werden ist es ratsam man loggt sich im „emergency mode“ ein und schaut sich gründlich das Log an welches man durch Eingabe, wie oben zu sehen, `journalctl -xb` erhält. Dieses Log hatte eine Größe von ca. 72K, somit recht umfangreich, so das eine Fehlerbehebung möglich sein sollte.

Siehe auch

kernel.org: Mounting the root filesystem via NFS (nfsroot) (https://www.kernel.org/doc/Documentation/filesystems/nfs/nfsroot.txt)

syslinux.org: pxelinux FAQ (http://www.syslinux.org/wiki/index.php/PXELINUX)

Von „https://wiki.archlinux.de/index.php?title=Diskless_Workstation&oldid=18474“

Kategorie: Installation
