Arch Linux

Hier sind die Themen zu Arch Linux hinterlegt.

Linux allgemein

In diesem Kapitel dokumentiere ich Informationen rund um Linux, die eher allgemeiner Natur sind und nichts im speziellen mit Arch Linux zu tun haben.

Warnmeldung "kauditd_printk_skb: .. callbacks suppressed" im Protokoll

Diese Warnmeldung gibt die Anzahl der Unterdrückten Warnmeldungen, die vom Netzwerk kommen, aus.

Das ist ein Mechanismus des Kernels um das System u.a. vor "Denial-of-Service"-Angriffen zu schützen.

```
sudo sysctl -a | grep net.core.message_
```

Es gibt dazu zwei Parameter, die dieses Mechanismus steuern:

```
net.core.message_burst
net.core.message_cost
```

Erläuterungen dazu:

Diese Parameter werden verwendet, um die Warnmeldungen einzuschränken, die vom Netzwerkcode in das Kernelprotokoll geschrieben werden. Sie erzwingen eine Ratenbegrenzung, um einen Denial-of-Service-Angriff unmöglich zu machen. Ein höherer ***message_cost*** *-Faktor führt zu weniger Nachrichten, die geschrieben werden. **Message_burst** steuert, wann Nachrichten verworfen werden. Die Standardeinstellungen begrenzen Warnmeldungen auf eine alle fünf Sekunden.*

Die aktuellen Werten können wie folgt ermittelt werden:

```
sudo sysctl -a | grep net.core.message_
```

Möchte man die Werte ändern geht das mit folgenden Befehlen:

```
sudo sysctl -w net.core.message_cost=0
```

```
sudo sysctl -w net.core.message_burst=0
```

Achtung: In einer produktiven Umgebung sollte man diese Werte nicht anpassen!!

Private Key und Zertifikat mit openssl erstellen

Für die verschlüsselte Übertragung von Syslog-Nachrichten an einen zentralen Log-Server (Graylog) werden private Schlüssel und Zertifikate benötigt.

Diese können wie folgt erstellt werden:

Privater Schlüssel

```
openssl genrsa -des3 -out privat.key 4096
```

Es wird dazu ein Passwort abgefragt, das später beim Erzeugen von Zertifikaten benötigt wird.

Zertifikat

```
openssl req -x509 -new -nodes -key private.key -sha256 -days 9999 -out zertifikat.pem
```

Eigene IP-Adresse(n) ermitteln

Unter Linux kann man mit folgendem Befehl die eigene(n) IP-Adresse(n) ermitteln:

```
ifconfig
```

Speicherverbrauch auf Datenträgern analysieren

Auf einem Server ohne GUI

Auf Systemen ohne GUI kann man zu Ermittlung der Belegung der Datenträger das Tool Ncdu nutzen. Es wird im Terminal wie folgt ausgeführt:

```
sudo ncdu
```

Direkt nach dem Start scannt das Tool ab dem aktuellen Verzeichnis sämtliche Dateien im aktuellen und in den Unterverzeichnissen und stellt das Ergebnis im Terminal dar.

Man kann mit den Cursor-Tasten und der Enter-Taste durch die Verzeichnisse navigieren.

Auf Systemen mit Plasma-Desktop (KDE)

Auf Systemen mit Plasma-Desktop kann das Tool Filelight dazu genutzt werden, Datenträger hinsichtlich Speicherverbrauch zu analysieren.

Eigenes Script mittels systemd starten

Es ist relativ einfach ein eigenes Script mit dem systemd beim Systemstart starten zu lassen. Dieses geschieht mit dem bekannten Befehl "sytemctl start".

Zunächst muss eine Service-Datei für den Dienst angelegt werden:

```
[Unit]
Description=Wireguard Verbindung WG1 aktiv halten

Wants=network.target
After=syslog.target network-online.target

[Service]
Type=simple
ExecStart=python3 /home/thomas/scripts/python/wg1_active.py
Restart=on-failure
RestartSec=10
KillMode=process

[Install]
WantedBy=multi-user.target
```

Mit dieser Service-Datei wird ein von mir geschriebenes Python-Script beim Systemstart gestartet.

Mit dem Script wird eine Wireguard-Verbindung "am Leben" gehalten.

Die Service-Datei darf nicht ausführbar sein.

Ort für die Service-Datei (unter Arch Linux):

```
/etc/systemd/system
```

Quelle: https://secinfinity.net/so-fuhren-sie-ein-linux-programm-beim-start-mit-systemd-aus/

Standard-Shell festlegen

Möchte man z.B. zsh als Standard-Shell festlegen, gibt man folgenden Befehl ein (zsh muss natürlich installiert sein):

```
chsh -s /usr/bin/zsh
```

Da das eine Einstellung für einen Benutzer ist, muss der Befehl von dem Benutzer ausgeführt werden, der die Standard-Shell ändern möchte.

SSD-Temperatur loggen

An einem RaspberryPI 4+ ist eine NVME-SSD über einen USB-Adapter angeschlossen. Es die Temperatur der SSD in eine Logdatei geschrieben werden.

Die folgende Lösung hat die Google-KI erstellt:

```
#!/bin/bash
# Pfad zur Logdatei (anpassen!)
LOGFILE="/home/DEIN_BENUTZERNAME/ssd_temp.log"
DEVICE="/dev/sda"

# Zeitstempel
DATE=$(date "+%Y-%m-%d %H:%M:%S")

# Extrahiert nur die Zahl aus der Zeile "Temperature: 46 Celsius"
TEMP=$(sudo smartctl -a $DEVICE | grep "^Temperature:" | awk '{print $2}')

# In Datei schreiben
echo "$DATE - SSD Temp: $TEMP°C" >> "$LOGFILE"
```

Das erstellte Bash-Script muss ausführbar gemacht werden:

```
chmod +x ssd_temp_log.sh
```

Automatisierung via cron-job.

Crontab öffnen:

```
sudo crontab -e
```

Zeile am Ende einfügen, damit die Temperatur alle 5 Minuten in die Logdatei geschrieben wird:

```
*/5 * * * * /pfad/zu/deinem/ssd_temp_log.sh
```

Aktuelle Temperatur im Terminal alle 5 Sekunden anzeigen (immer nur der aktuelle Wert)

```
watch -n 5 "sudo smartctl -a /dev/sda | grep '^Temperature:'"
```

SSH-Zugriffe (Port 22) nur aus dem internen Netz zulassen

Ich habe eine VPN-Verbindung zwischen einem externen Server und einem Raspberry Pi in meinem Heimnetz. Dadurch kann man sich vom Server aus per SSH mit dem Raspberry Pi verbinden (natürlich müssen Benutzer und Passwort ebenfalls bekannt sein).

Um dies zu verhindern kann man mit Hilfe der IPTables-Firewall eine Regel auf dem Raspberry Pi erstellen, die diese verhindert.

Erster Schritt: Zugriff vom internen Netz zulassen:

```
iptables -A INPUT -s 192.168.0.0/16 -p tcp -m tcp --dport 22 -j ACCEPT
```

Zweiter Schritt: Sämtliche Zugriffe von Außen sperren:

```
iptables -A INPUT -p tcp -m tcp --dport 22 -j DROP
```

Diese Befehle ändern die Einstellungen zunächst temporär. Damit dieses permanent aktiv ist muss unter Ach Linux noch folgender Befehl eingeben werden:

```
iptables-save -f /etc/iptables/iptables.rules
```

Quellen:

1. https://www.gutefrage.net/frage/wie-kann-ich-meinen-ssh-port-von-aussen-blockieren-mit-iptables
2. https://wiki.archlinux.org/title/iptables

Kommandozeilen Befehle

Um den geeigneten Befehl für die Kommandozeile zu finden, kann folgende Seite sehr gut helfen:

CMD Generator

Systemd Service-Datei anpassen

Eine Systemd Service-Datei soll nicht direkt angepasst werden.

Es gibt dafür eine Override-Möglichkeit. Mit dieser lassen sich Inhalte in der Service-Datei überschreiben bzw. anpassen.

Mit folgendem Kommando erzeugt und editiert man eine entsprechende Override-Datei (Beispiel für zoraxy):

```
sudo systemctl edit zoraxy
```

Die Override-Datei wird mit dem Inhalt der aktuellen Service-Datei angelegt. Die Zeilen sind aber alle auskommentiert.

Es ist nicht mehr zulässig an den aus der Service-Datei übernommen Inhalten direkt etwas anzupassen. Stattdessen soll oberhalb des auskommentierten Bereiches der Block mit den angepassten Zeilen eingefügt werden. Beispiel:

```
### Editing /etc/systemd/system/zoraxy.service.d/override.conf
### Anything between here and the comment below will become the contents of the drop-in file

[Service]
ExecStart=
ExecStart=/usr/bin/zoraxy -log /var/log/zoraxy -port=:49999 -fastgeoip=true 2>&1 | logger &

### Edits below this comment will be discarded



### /usr/lib/systemd/system/zoraxy.service
# [Unit]
# Description=Zoraxy Reverse Proxy Server
#
# # start not befor network is online
# After=network-online.target
```

```
# Wants=network-online.target
#
# [Service]
# Type=simple
#
# # start zoraxy as root
# User=root
# Group=root
#
# # folder where zoraxy config & runtime data are located
# WorkingDirectory=/usr/lib/zoraxy
#
# # use absolute path for zoraxy always
# # options:
# # -fastgeoip=true               faster GeoIP-lookup, but more RAM ...
# #                               faster GeoIP-lookup, but uses more RAM ...
# # -log /var/log/zoraxy          log to file / dont create local log folder
# # -port=:8008                   use port instead of defalut (8000)
#
# ExecStart=/usr/bin/zoraxy -log /var/log/zoraxy -port=:8000 2>&1 | logger &
# ExecStop=/usr/bin/kill "$MAINPID"
#
#
# [Install]
# WantedBy=multi-user.target
```

Mit Zeile 4 und 5 passe ich die Zeile 35 entsprechend meinen Vorstellungen an.

Im Standard ist der Editor zum anpassen "nano". Mir persönlich gefällt der Editor "micro" (https://micro-editor.github.io/ besser. Nachdem man "micro" installiert hat, kann man mit folgendem Aufruf auch "mirco" als Editor nutzen:

```
sudo env EDITOR=micro systemctl edit zoraxy
```

AUR-Pakete

In diesem Kapitel geht es um das erstellen oder verändern von AUR-Paketen.

AUR-Paket anpassen inklusive Erstellung Patch

Da sich z.B. Bibliotheken geändert haben und dadurch die Abhängigkeiten nicht mehr auflösen lassen, muss ein existierendes AUR-Paket angepasst werden.

In diesem Beispiel geht es konkret um das Pakte urbackup2-client, welches wegen der Umstellung der wxgtk-Pakete (https://archlinux.org/news/wxwidgets-32-update-may-need-manual-intervention/) angepasst werden muss.

Dieses AUR-Paket wird über GitHub (https://github.com/funilrys/aur-urbackup2-client) verwaltet. Die Beschreibung der Änderungen am Paket sind aber auch allgemein gültig.

Zur GitHub-Anbindung nutze ich Visual Studio Code.

Git-Repository clonen

Wie das mit VS-Code geht ist hier beschrieben.

So sieht es dann aus:

hier folgt noch ein Bild...

.SRCINFO-Datei erstellen

```
$ makepkg --printsrcinfo > .SRCINFO
```

Patch erstellen

Zunächst "src"-Verzeichnis löschen, sofern es schon da ist.

Dann mit

```
makepkg --nobuild
```

das Source-File downloaden. Außerdem wird der Sourcecode entpackt.

Dann erstellt man sich von dem Verzeichnis, in dem eine Datei gepatcht werden muss zwei Kopien an einem anderen Ort.

Z.B. kopiert man den Ordner "src" nach "src_ori" und "src_neu".

Dann führt man die notwendigen Änderungen im Ordner "src_neu" durch. Im übergeordnetem Ordner führt man dann zunächst

```
diff --unified --recursive --text src_ori src_neu --color
```

aus. Das ist ein Testlauf der die gefunden Änderungen farblich im Terminal dargstellt.

Ist man zufrieden muss noch

```
diff --unified --recursive --text src_ori src_neu > neuerpatch.patch
```

ausführen, was die eigentliche Patchdatei mit dem Namen "neuerpatch.pach" erstellt.

Anschließend kopiert man die Datei "neuerpatch.pach" in das Verzeichnis "src_ori" und fürht dort folgendes aus:

```
patch --strip=1 < ../neuerpatch.patch
```

Die Änderungen sollten nun auch im Verzeichnis "src_ori" durchgeführt sein. Dieses sollte man überprüfen.

Wenn der Patch funktioniert wird dieser in das eigentliche Paket-Verzeichnis kopiert. Dort muss dann die PKGBUILD-Datei entsprechend angepasst. Dazu diese editieren. Hier ein Beispiel:

```
pkgname=urbackup2-client
pkgver=2.4.11
pkgrel=4
pkgdesc="Client Server backup system"
arch=('i686' 'x86_64' 'armv5' 'armv6h' 'armv6' 'armv7h' 'armv7' 'aarch64')
url="http://www.urbackup.org/"
license=("GPL")
makedepends=('gcc-libs' 'gcc' 'make')
depends=('wxwidgets' 'crypto++' 'zlib')
```

```
conflicts=('urbackup2-client-no-gui' 'urbackup-client-no-gui' 'urbackup-client')
source=(
    "https://www.urbackup.org/downloads/Client/${pkgver}/urbackup-client-${pkgver}.0.tar.gz"
    'btrfs_create_filesystem_snapshot'
    'btrfs_remove_filesystem_snapshot'
    'dattobd_create_filesystem_snapshot'
    'dattobd_remove_filesystem_snapshot'
    'defaults_client'
    'lvm_create_filesystem_snapshot'
    'lvm_remove_filesystem_snapshot'
    'md5-bytes.patch'
    'double_language.patch'
)
sha512sums=('a0cae5dad805d11e2764b2635dd37a4e24e0d029bae1eefc471f2b87262ac2448b3d123c2ffb5deb2eccab1baed0e31d3711e8837d66b93edf736fb113145ef1'

'416fb8f5f3687a3c369cc2b199d4c8b4170494f0a119566a91ac6a0c2f202dc5049804c10508b66ba657011b39be5ddd055091cd531a665b4398899f404086ca'

'860021ce5b8d92ff58e8286991162c7bab45493c3b9c87577a43764f6b416397448bb99b8fcb850c4c5853927cb0a8637792b75ff53ee7ee257da3f5d29ae3a7'

'fde5912b589a495dc03a26d174d7673ff746eed34d6b1ed64758b2dc2ec2ec53e02e6a28b04734a7112f16687b31d25123e99dbc69e9dcab48773675382ec582'

'a8b58bba1b8b0a6b70395f9fe4277eeee60a0ba534f4eddb999d719915c76b76facb54172e03b7b29b9f725a4d720e9b676b05e5081f7528570956e903fe59bd'

'238c286d451474a8721292f7e98b4f13600cb430c16a27ceb9551cc83705b8268a3f1202785fb5b61523f372b4e7e804fd20b7db62677621983d79a271aa106b'

'a2d4ba03ae15582d2cd74ff68c38ff0f90d75a6eb5c241f9a022b0652fa2dc9b184439f6bda9a9538645925f739503ee7b3fc7bb232589583cdeb6dc27d74e5c'

'9bdfefccdd9d6e37a77975324a7c417f3de2aa59e6da0bfde3c318b8c6f3d7f4629f3a41eebee548b9c572b8ed39640434cc08bd020d25362fddffc4426438de'

'34e25c868cf4572414fbc6c693877127152f9a97edf8865b4263a55cf16f71a5045ba96b1a9af8244ed49c35cab56e3fdb44348d191e9f85e2efb66392907132'
```

```
'fb6c43d725d4ea201bdf91b1482473f205834c14f24906041d5c6ef22b1e0f4cec16d5e765a3fda6fed4d3b98f621
7b190dd5d1e84051525205e22747c5eefac')

CFLAGS="-march=native -O2 -pipe -fstack-protector-strong"
CXXFLAGS="${CFLAGS} -ansi -std=gnu++11"
CPPFLAGS="${CPPFLAGS} -DNDEBUG"
MAKEFLAGS="-j$(nproc)"

build() {
    sed  -i '/\#include \"cryptopp_inc.h\"/a #include "assert.h"' "${srcdir}/urbackup-client-
${pkgver}.0/cryptoplugin/AESGCMDecryption.h"

    patch -d"${srcdir}/urbackup-client-${pkgver}.0" -p0 < "${srcdir}/md5-bytes.patch"
    patch --forward --strip=1 --input="${srcdir}/double_language.patch"

    cd "${srcdir}/urbackup-client-${pkgver}.0"
    ./configure --prefix=/usr --sbindir=/usr/bin --localstatedir=/var --sysconfdir=/etc --
enable-embedded-cryptopp
    make
}

package() {
    cd "${srcdir}/urbackup-client-${pkgver}.0"
    make DESTDIR="${pkgdir}" install
    sed -i 's/\/usr\/local\/sbin/\/usr\/bin/gi' urbackupclientbackend-debian.service
    install -Dm644 urbackupclientbackend-debian.service \
    "${pkgdir}"/usr/lib/systemd/system/urbackupclientbackend.service
    install -Dm644 docs/urbackupclientbackend.1 \
    "${pkgdir}"/usr/share/man/man1/urbackupclientbackend.1

    cd "${srcdir}"
    install -Dm644 defaults_client "${pkgdir}/etc/default/urbackupclient"
    install -Dm700 btrfs_create_filesystem_snapshot "${pkgdir}/usr/share/urbackup"
    install -Dm700 btrfs_remove_filesystem_snapshot "${pkgdir}/usr/share/urbackup"
    install -Dm700 lvm_create_filesystem_snapshot "${pkgdir}/usr/share/urbackup"
    install -Dm700 lvm_remove_filesystem_snapshot "${pkgdir}/usr/share/urbackup"
    install -Dm700 dattobd_create_filesystem_snapshot "${pkgdir}/usr/share/urbackup"
    install -Dm700 dattobd_remove_filesystem_snapshot "${pkgdir}/usr/share/urbackup"
}
# vim: ts=2
```

Bei diesem Paket habe ich den Patch "double_language.patch" ergänzt. Dazu musste diese in diesem Abschnitt am Ende hinzugefügt werden:

```
source=(
    "https://www.urbackup.org/downloads/Client/${pkgver}/urbackup-client-${pkgver}.0.tar.gz"
    'btrfs_create_filesystem_snapshot'
    'btrfs_remove_filesystem_snapshot'
    'dattobd_create_filesystem_snapshot'
    'dattobd_remove_filesystem_snapshot'
    'defaults_client'
    'lvm_create_filesystem_snapshot'
    'lvm_remove_filesystem_snapshot'
    'md5-bytes.patch'
    'double_language.patch'
)
```

Damit der Patch auch ausgeführt wird benötigt es noch die Zeile

```
patch --forward --strip=1 --input="${srcdir}/double_language.patch"
```

im Abschnitt

```
build() {
    sed  -i '/\#include \"cryptopp_inc.h\"/a #include "assert.h"' "${srcdir}/urbackup-client-
${pkgver}.0/cryptoplugin/AESGCMDecryption.h"

    patch -d"${srcdir}/urbackup-client-${pkgver}.0" -p0 < "${srcdir}/md5-bytes.patch"
    patch --forward --strip=1 --input="${srcdir}/double_language.patch"

    cd "${srcdir}/urbackup-client-${pkgver}.0"
    ./configure --prefix=/usr --sbindir=/usr/bin --localstatedir=/var --sysconfdir=/etc --
enable-embedded-cryptopp
    make
}
```

Offiziell sollen solche Patch-Files in einem eigenen "prepare()"-Abschnitt eingefügt werden. Da in diesem Fall aber bereits ein andere Patch im "build()"- Abschnitt existierte, habe ich ihn dort eingefügt.

Nach den Änderungen muss noch mit

```
updpkgsums
```

die Checksumme ergänzt werden.

Checksumme manuell erstellen

```
sha512sum file1.txt
```

Checksummen automatisiert erstellen (für alle Dateien)

```
updpkgsums
```

Links:

https://wiki.archlinux.org/title/Patching_packages

https://wiki.archlinux.de/title/Arch_Build_System

AUR-Paket erstellen

https://aaron-fischer.net/rxy

AUR-Pakete veröffentlichen

Möchte man für das AUR Pakete veröffentlichen , benötigt man zunächst folgende Dinge:

1. AUR-Account
2. SSH-Schlüsselpaar

Das anlegen eines Accounts ist selbsterklärend. Daher gehe ich hier nicht weiter darauf ein.

SSH-Schlüsselpaar erzeugen

Dieses wird von ChatGPT wie folgt beschrieben:

```
ssh-keygen -t ed25519 -C "deinname@deinrechner"
```

- `-t ed25519` → moderner, sicherer Schlüsseltyp
- `-C "Kommentar"` → nur ein Hinweis (meist Benutzername@Rechnername), optional

Dann kommt:

```
Enter file in which to save the key (/home/thomas/.ssh/id_ed25519):
```

- Standard ist `~/.ssh/id_ed25519`.
- Falls du schon einen Key dort hast, kannst du ENTER drücken (überschreibt nicht automatisch), oder einen neuen Namen vergeben wie `~/.ssh/id_ed25519_aur`.

Dann wirst du nach einer **Passphrase** gefragt – das ist optional, aber empfohlen. Wenn du keine willst, einfach ENTER drücken.

Danach liegen zwei Dateien in `~/.ssh/`:

- **privater Schlüssel**: `id_ed25519`
  (gut aufbewahren, darf nie hochgeladen werden)
- **öffentlicher Schlüssel**: `id_ed25519.pub`
  (den musst du ins AUR eintragen)

Inhalt der Datei anzeigen:

```
cat ~/.ssh/id_ed25519.pub
```

- Den kompletten Inhalt (eine Zeile, beginnt mit `ssh-ed25519 ...`) kopieren.
- Auf aur.archlinux.or einloggen → **My Account** → **SSH Public Keys** → einfügen → speichern.

Jetzt prüfen, ob die Verbindung klappt:

```
ssh -T aur@aur.archlinux.org
```

Wenn alles passt, kommt so etwas wie:

```
Welcome to the AUR, <AUR-Benutzername>! Interactive shell is disabled.
```

Paket anpassen und veröffentlichen

Zunächst auf der Ordnerstruktur im persönlichen Ordner ein geeignetes Verzeichnis anlagen und darein wechseln und ein Terminal öffnen.

Dann folgendes ausführen um den aktuelle Stand des Paketes in den Ordner zu laden. Hier am Beispiel des Pakets "cockpit-navigator":

```
git clone ssh://aur@aur.archlinux.org/cockpit-navigator.git
```

Es ensteht der Ordner "cockpit-navigator". Darein wechseln:

```
cd cockpit-navigator
```

Darin befindet sich u.a. die Datei PKGBUILD. Diese mit einem Editor nach Bedarf anpassen. U.a. muss sich bei jeder Änderung mindestens die Paketversion ändern.

Hat man alle Änderungen durchgeführt und die Datei gespeichert, muss man die Checksumme(n) anpassen. Dazu im Terminal folgenden Befehl ausführen:

```
updpkgsums
```

Dabei wird auch der Sourcecode runtergeladen, damit die Checksumme gebildet werden kann.

Dann sollte die Datei .SRCINFO neu erstellt werden:

```
makepkg --printsrcinfo > .SRCINFO
```

Nun sollte die Erstellung und Installation des Paketes getestet werden:

```
makepkg -si
```

Läuft alles fehlerfrei durch, müssen jetzt noch die Änderungen dokumentiert und hochgeladen werden:

```
git add PKGBUILD .SRCINFO
git commit -m "einen passenden Kommmentar hinterlegen"


git push origin master
```

Jetzt wird man noch mal nach dem Passwort zu seinem SSH-Schlüssel gefragt. Nach der Eingabe erfolgt das Hochladen der Änderung und die Veröffentlichung.

Pacman

Paketcache aufräumen

Pacman behält alle heruntergeladenen Pakete in einem Cache (/var/cache/pacman/pkg/). Damit die benötige Menge an Speicherplatz nicht immer größer wird, kann man pacman anweisen, den cache aufzuräumen.

Zunächst sollte man sich den aktuell belegten Speicherplatz ausgeben lassen:

```
du -sh /var/cache/pacman/pkg
```

Mit diesem Befehl kann man dann veraltete Pakete löschen lassen:

```
sudo pacman -Sc
```

Möchte man den cache komplett leeren lassen, geht das mit diesem Befehl:

```
sudo pacman -Scc
```

Arbeitet man viel mit YAY und hat auch AUR-Pakete installiert, löscht folgender Befehl nicht nur den normalen Pakte-Cache, sondern auch den YAY-Cache für die AUR-Pakete:

```
yay -Scc
```

Lokales Paket installieren

Zum installieren eines lokalen Paketes, wird pacman wie folgt aufgerufen:

```
pacman -U <Paket-Dateiname>
```

Installierte Pakete suchen

Möchte man wissen, welche Pakete installiert sind kann man dieses wie folgt abfragen:

```
sudo pacman -Q
```

Möchte man nicht alle installierten Pakte aufgelistet bekommen, sondern interessiert sich nur z.B. für ein bestimmtes Paket, kann man mit Hilfe einer Abfrageoption auch noch einen *Regulärer_Ausdruck* hinzufügen:

```
sudo pacamn -Q -s <Regulärer_Ausdruck>
```

Ein konkrete Abfrage, welches MongoDB-Paket installiert ist, sieht dann so aus:

```
sudo pacman -Q -s mongodb
```

Link:

https://man.archlinux.org/man/pacman.8.de

Archlinux Pakete

Paket selber compilieren

Eine deutsche Anleitung zu diesem Thema gibt es hier:

https://wiki.archlinux.de/title/Arch_Build_System

Die Seite ist leider etwas veraltet. Z.B. wird nicht beschrieben, wie man ein mit einem PGP-Schlüssel signiertes Paket compiliert bekommt.

Die Prüfung der PGP-Signatur kann aber auch deaktiviert werden:

```
makepkg --skippgpcheck
```