

# Paket-Formate der Zukunft?

AppImage, Flatpak und Snap im Vergleich

# whoami

- > Christian Stankowic
- > Linux, Virtualisierung, IaC
- > Every Linux is awesome!\*
- > 🐘: [@stdevel@chaos.social](mailto:stdevel@chaos.social)
- > 📝: <https://cstan.io>
- > 🎙️: [FOCUS On: Linux](#)

\* auch [Hannah Montana Linux](#)



# Agenda

- > Motivation und Überblick
- > AppImage
- > Flatpak
- > Snapcraft
- > Ausblick

# Motivation

# Motivation

User wollen:

- > Stabilität
- > **Aktualität**, funktionale Updates
- > Benutzbarkeit
- > **one-click**-Experience



# Motivation

Entwickler:innen wollen:

- > Code schreiben\*
- > Fokus auf Funktionalität
- > Testen von Erwartungen
- > sich nicht mit Paketierung rumärgern

\* Nein! Doch! Ohh!



# Klassische Paketverwaltung

- > Die Auswahl ist groß:
  - > RPM (*yum, dnf, zypper*)
  - > DEB (*apt, apt-get*)
  - > Arch (*pacman*)
  - > slackpkg, apk, nix,...
- > Ergänzende quellenbasierte\* Paketmanager:
  - > **AUR** (*Arch User Repository*)
  - > CRUX Ports
  - > **Portage** (*Gentoo Linux*)



\* Paket wird vor Installation kompiliert

# Qual der Wahl: Linux-Distributionen

	Releases	Wartung	Paketformat	Pakete*	Fokus
Ubuntu	6m bzw. 2y (LTS)	9m bzw. 5-10y	DEB	~34k	Aktualität, Stabilität
Debian	~2y	3-5y	DEB	~32k	Stabilität
Fedora	~6m	~1y	RPM	~23k	Aktualität
RHEL	~3y	10-13y	RPM	~2.5k	Stabilität
Alma/Rocky Linux	~3y	10y	RPM	~2.5k	Stabilität
SLES	~3y (major), ~1y (SP)	10-13y	RPM	13k	Stabilität, Aktualität
openSUSE Leap	~3y (major), ~1y (SP)	18m	RPM	13k	Stabilität, Aktualität
openSUSE Tumbleweed	Rolling Release	-	RPM	14k	Bleeding Edge
Arch	Rolling Release	-	PKG	10k	Bleeding Edge
NixOS	~6m	~1y	Nix	~80k+	Bleeding Edge

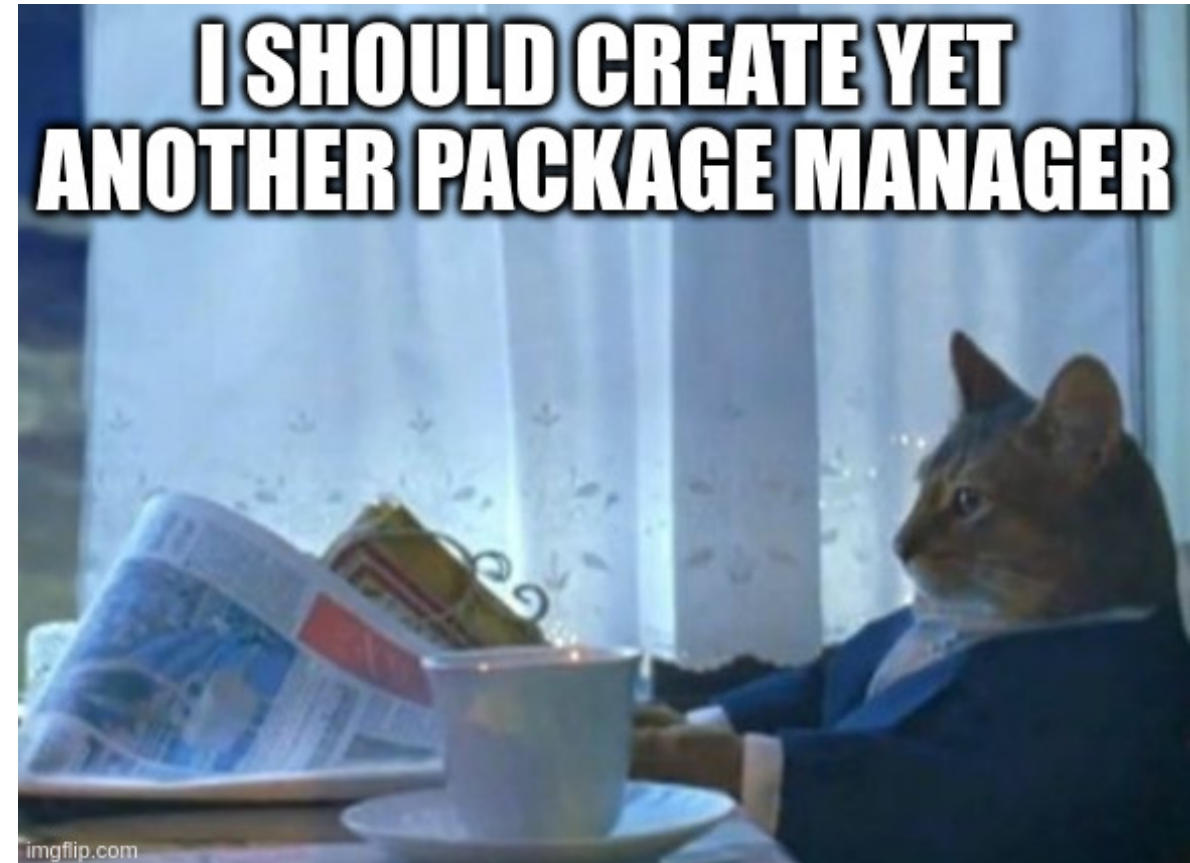
\*Anzahl der Pakete bezieht sich auf das Core OS ohne zusätzliche Repositories



# Status Quo

- > Linux ist für viele Entwickler:innen eine **zu komplexe** Plattform:
  - > Die Auswahl an Linux-Distributionen ist groß
  - > Zahlreiche Paket-Manager
  - > Es gibt zu unterschiedliche Versionsstände

Gäbe es doch nur eine Lösung dafür...



# "Neue" Paketmanager

- > Drei Frameworks zur Problemlösung:
  - > ApptImage
  - > Flatpak
  - > Snapcraft
- > werden parallel zum systemweiten Paketmanager verwendet
- > verstehen sich als **Ergänzung** und nicht als Ersatz



# AppImage

# ApplImage

- > 2004 von Simon Peter (*probonopd*) als **klik** gestartet
  - > Fokus auf Desktop-Anwendungen und -User
  - > benötigt keine Runtime
  - > **keine Installation** benötigt, einfach **ausführen**
    - > analog zu .exe, .dmg und PortableApps
  - > SquashFS-Format\*, keine Sandbox-Umgebung
  - > keine Rechteverwaltung
  - > ca. **1.400** Anwendungen im zentralen [ApplImageHub](#)
- \* v2, v1 [nutzte ISO](#)



# AppImages ausführen

Heruntergeladene Datei ausführbar machen und starten:

```
$ chmod u+x DogeApp.AppImage  
$ ./DogeApp.AppImage
```

Zur Interaktion mit dem Paketformat stehen verschiedene **Parameter** bereit:

- > `--appimage-help` (Hilfe anzeigen)
- > `--appimage-extract` (Paket entpacken)
- > `--appimage-portable-home` (abweichender Pfad für ~)
- > `--appimage-portable-config` (abweichender Pfad für ~/.config)

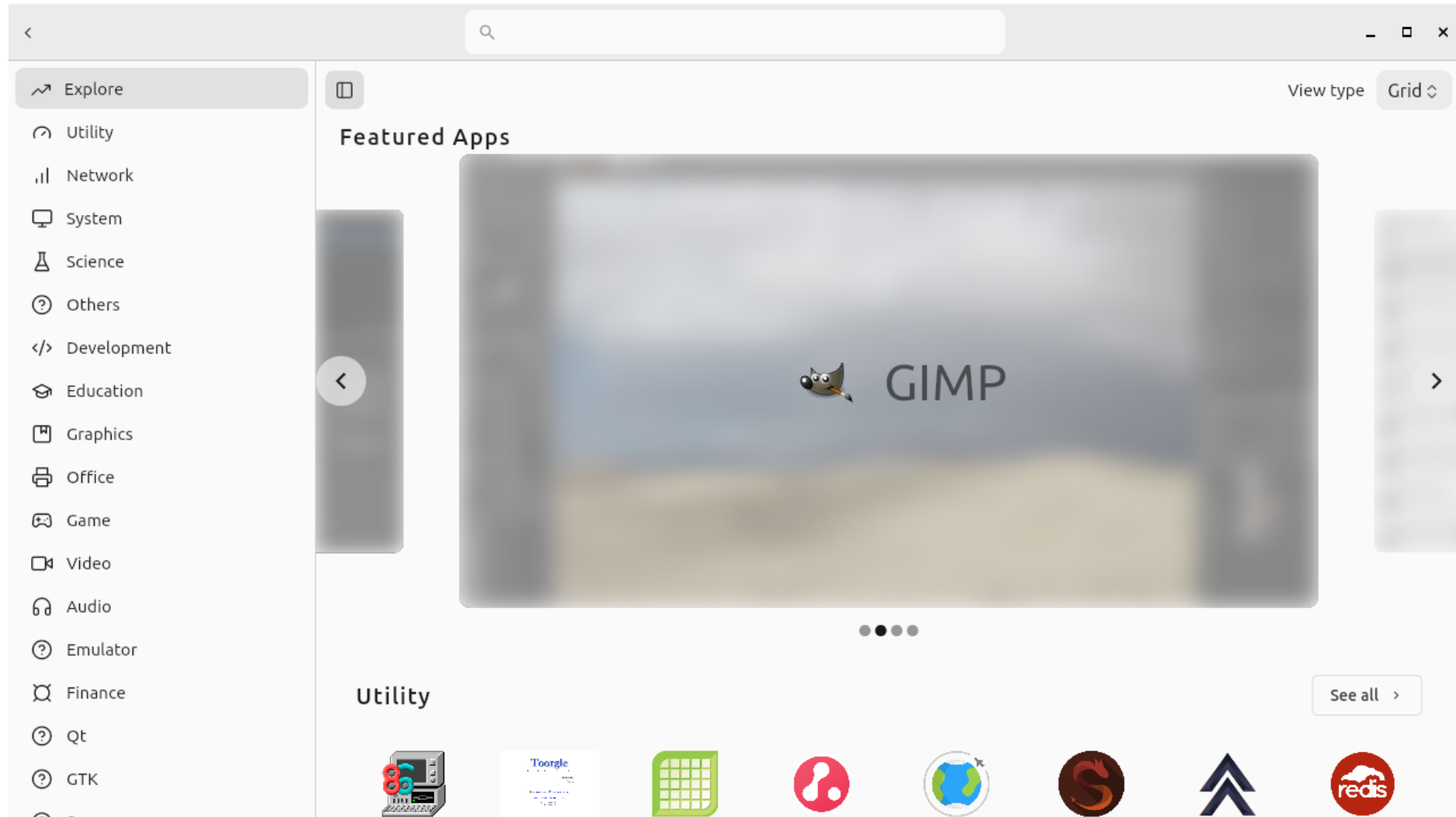
# Applimages verwalten

- > Da Applimages einfach heruntergeladen und ausgeführt werden, gibt es keine **zentrale Verwaltung**
  - > Philosophie: *one app = one file*

Abhilfe schaffen:

- > [ApplImageUpdate](#)
  - > offizielles Tool
  - > optionaler `appimaged`-Dienst, Rechtsklick im Panel zum Updaten
- > [ApplImage Pool](#)
  - > Flutter Store-UI

# AppImage Pool



# Exkurs: DBUS-Portals

- > **Desktop-Bus**, Bibliothek für Interprozesskommunikation
    - > Teil von freedesktop.org, Bestandteil moderner Linux-Distributionen
  - > API-Definitionen zur Rechteverwaltung von Anwendungen, z.B.
    - > `org.freedesktop.portal.Account` - Abfragen persönlicher Informationen
    - > `.Camera` - Zugriff auf Webcams
    - > `.Location` - Abfragen von Standort-Informationen
    - > `.Screenshot` / `.ScreenCast` - Bildschirmfotos/-videos erstellen
    - > `.FileChooser` - Dateien lesen/speichern
- \* vollständige Liste siehe [hier](#)



# Anatomie eines AppImages

```
HannahMontanaApp.AppDir/           # Anwendungsverzeichnis
HannahMontanaApp.AppDir/AppRun       # Start-Skript/-Programm
HannahMontanaApp.AppDir/myapp.desktop # Desktopverknüpfung
HannahMontanaApp.AppDir/myapp.png    # Desktop-Icon
HannahMontanaApp.AppDir/usr/bin/myapp # Anwendung(en)
HannahMontanaApp.AppDir/usr/lib/libfoo.so.0 # Bibliothek(en)
```

Auf GitHub existiert ein [AppRun](#)-Beispiel:

- > setzt verschiedene Umgebungsvariablen
  - > PATH, LD\_LIBRARY\_PATH, PYTHONPATH, XDG\_DATA\_DIRS, PERLLIB, GSETTINGS\_SCHEMA\_DIR, QT\_PLUGIN\_PATH
- > führt Anwendung (anhand .desktop-Datei) aus

# Erstellen eines AppImages

Es gibt verschiedene Optionen:

1. Open Build Service
2. Bestehende Binärpakete konvertieren
3. Travis CI benutzen
4. `linuxdeployqt` für Qt-Anwendungen
5. `electron-builder` für Electron-Anwendungen
6. Manuell Ordnerstruktur erstellen und konvertieren

# Open Build Service

- > empfohlen für Open Source-Projekte
- > Service zur automatisierten Paket-Erstellung
- > **openSUSE Build Service**: für OSS kostenlose Instanz
  - > kann auch [selbst gehostet](#) werden
- > **automatisches** Neubauen falls ebenfalls auf OBS gehostete Abhängigkeiten neue Versionen erstellen
- > **automatisches Signieren** mit User Key
- > bettet Update-Informationen ein um **Binärdelta-Updates** zu ermöglichen



# Binärpakete konvertieren

[pkg2appimage](#)-Skript erleichtert das Konvertieren von Binärpaketen:

- > lädt Binärdateien von Debian-Repositories oder Webseiten herunter
- > führt **Skripte** aus, um Daten anzupassen
- > **erstellt** anschließend ein AppImage-Abbild
- > Konfiguration via **YAML**-Datei, fertige Vorlagen (**Recipes**) auf [GitHub](#)
- > erfordert Kenntnisse über die Struktur des Pakets

# Beispiel: ZDoom

---

app: zdoom

union: true

ingredients:

dist: precise

sources:

- deb http://us.archive.ubuntu.com/ubuntu/ precise main universe multiverse
- deb http://us.archive.ubuntu.com/ubuntu/ precise-updates main universe multiverse
- deb http://debian.drdteam.org/ stable multiverse

script:

- wget -c https://github.com/freedoom/freedoom/releases/download/v0.11.1/freedoom-\*.zip
- unzip freedoom-\*.zip

script:

- mv -f opt/zdoom/\* usr/bin/
- mkdir -p usr/share/games/doom
- mv ../freedom-\*/usr/share/games/doom/

zdoom.yml - [Quelle](#)

# Beispiel: ZDoom

## Konvertieren des Pakets:

```
$ pkg2appimage zdoom.yml
...
Embedding ELF...
Marking the AppImage as executable...
Embedding MD5 digest
Success
...
-rwxr-xr-x 1 cstankow cstankow 23M Aug  4 17:50 ../out/ZDoom-2.8.1.glibc2.15-x86
```

# Manuelle Ordnerstruktur

- > Ordnerstruktur lässt sich manuell erstellen
- > Konvention: `$anwendung.AppDir`
- > benötigt [appimagetool](#) zum Erstellen des Images

# Beispiel

Entpacken einer Debian-Datei, Erstellen der Ordnerstruktur:

```
$ mkdir leafpad ; cd $_
$ curl -O http://ftp.de.debian.org/debian/pool/main/l/leafpad/leafpad\_0.8.18.1-5\_
$ mkdir leafpad.AppDir ; cd $_
$ dpkg -x ../leafpad*.deb .
$ cp usr/share/icons/hicolor/32x32/apps/leafpad.png .
$ cp usr/share/icons/hicolor/scalable/apps/leafpad.svg .
$ cp usr/share/applications/leafpad.desktop .
$ curl -O https://raw.githubusercontent.com/AppImage/AppImageKit/master/resources
$ tree
```

```
.
├── AppRun
├── leafpad.desktop
├── leafpad.png
├── lib
│   ├── ...
└── usr
    ├── ...
```



# Beispiel

## Erstellen des Abbilds:

```
$ cd ..  
$ appimagetool leafpad.AppDir  
appimagetool, continuous build (commit 729a1a6), build <local dev build> built on  
Using architecture x86_64  
...  
Embedding ELF...  
Marking the AppImage as executable...  
Embedding MD5 digest  
Success
```

# Generelle Tipps

- > AppImages sollten alle benötigten Bibliotheken und Abhängigkeiten enthalten, die **nicht** zur Grundinstallation gehören
- > Auf dem ältesten System kompilieren, das unterstützt werden soll\*
  - > insbesondere `glibc`-Abhängigkeiten können Probleme machen
- > Auf unterstützten Systemen ausgiebig testen
- > keine hardkodierten Pfade verwenden
- > siehe auch Dokumentation im [AppImage-Wiki](#)

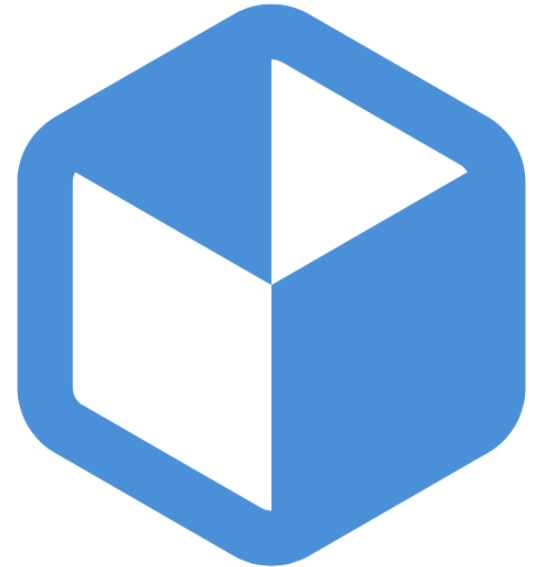
\* ältestes supportetes LTS-Release oder zumindest `$current-1`

# Flatpak

# Flatpak

- > 2007 erstmalig als **Glick** vorgestellt\*
- > seit 2016 als Flatpak bekannt
- > Fokus auf Desktop-Anwendungen
- > benötigt Runtime
- > Installation benötigt
- > startet Anwendungen in **Sandbox**-Umgebung
  - > Rechteverwaltung via **XDG-Portals**
  - > Konfiguration über DE oder [Flatseal](#)

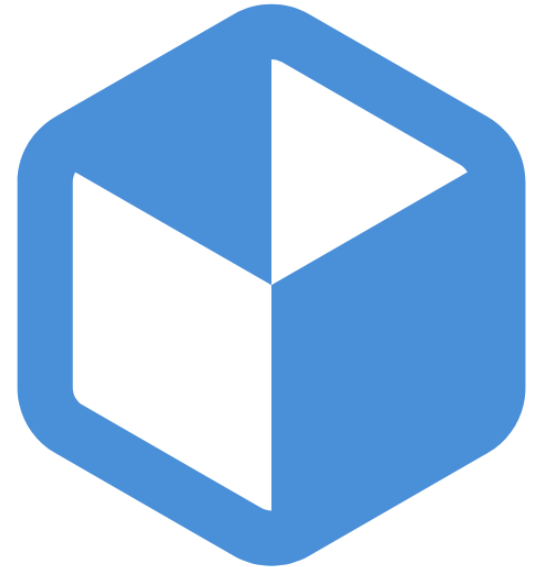
\* spätere Iterationen hießen *Glick 2*, *bundler* und *xdg-app*



## Flatpak

# Flatpak

- > verwendet **OSTree**- oder OCI-Images
- > ca. 2.300 Anwendungen auf [Flathub](#)
  - > **dezentrale** Stores vorhanden (elementaryOS, Pop!\_OS)
- > Updates als neues Images
- > derzeit auf [36 Distributionen](#) unterstützt
  - > auf einigen schon vorinstalliert



## Flatpak

# Flatpak einrichten

Flatpak-Runtime installieren:

```
# yum install flatpak  
# apt-get install flatpak  
# zypper install flatpak  
# pacman -S flatpak
```

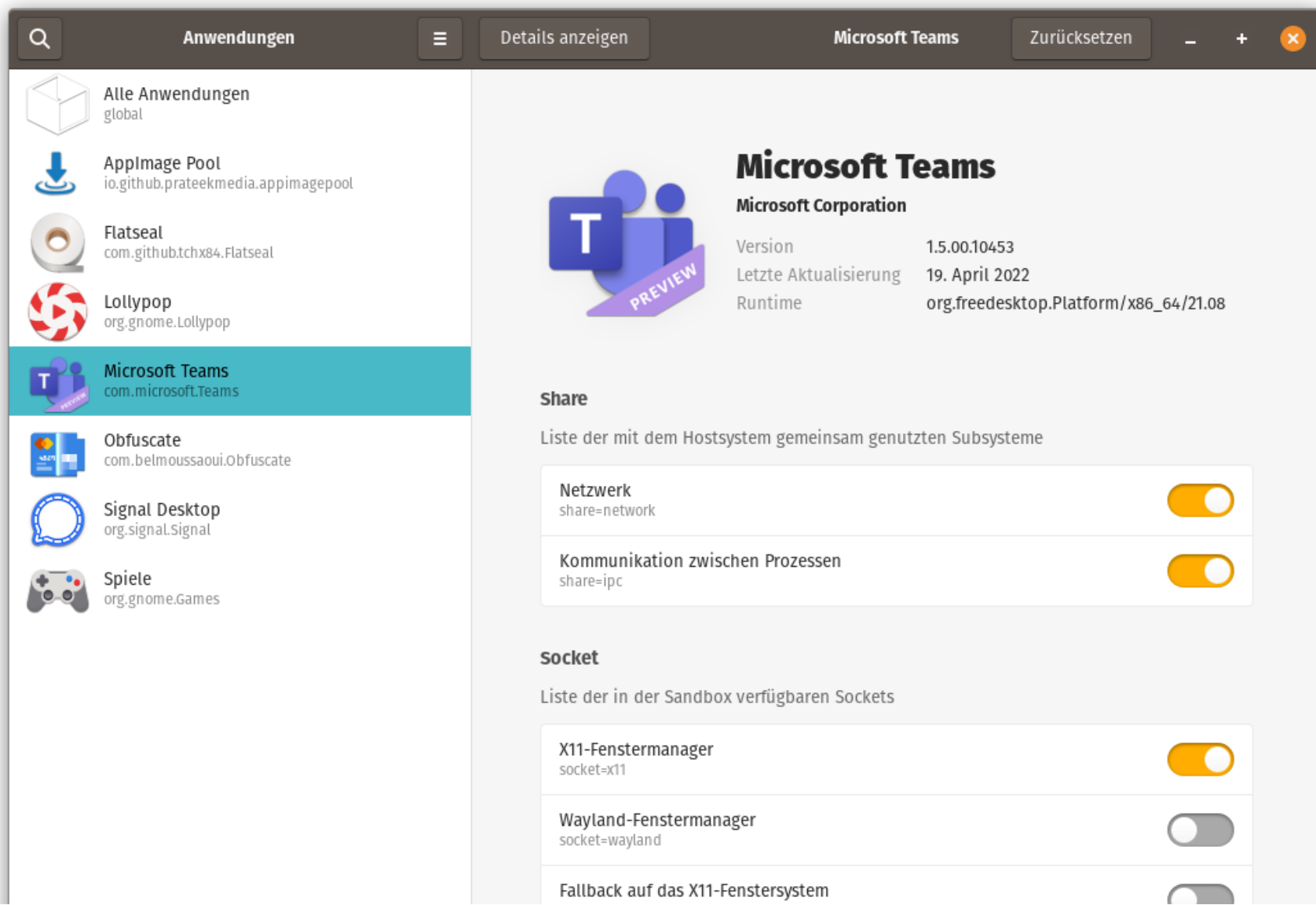
Store hinzufügen:

```
$ flatpak remote-add --if-not-exists flathub https://flathub.org/repo/flathub.flatpakrepo
```

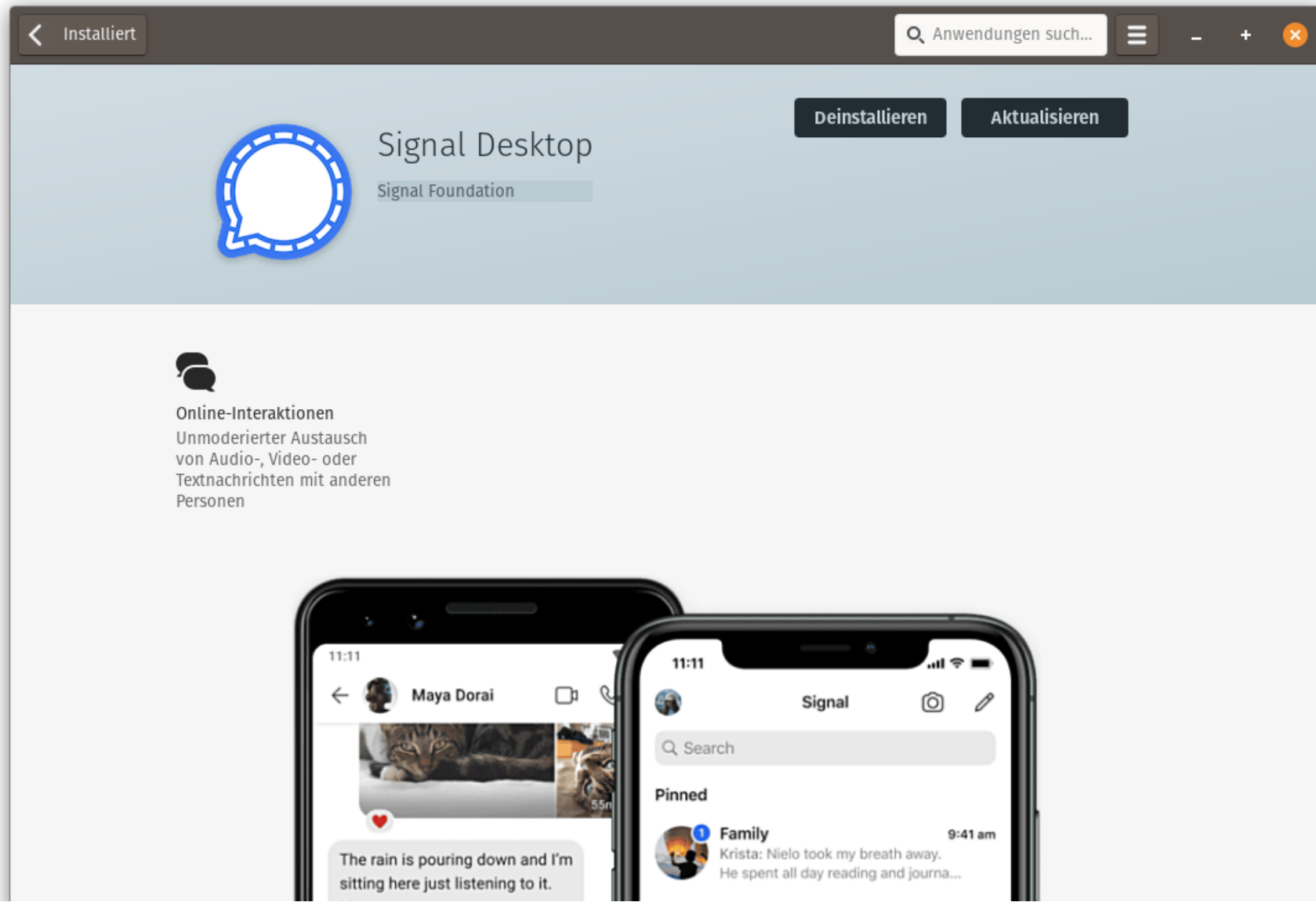
Anwendungen installieren:

```
$ flatpak install flathub com.github.tchx84.Flatseal
```

# Flatseal



# Integration in Pop!\_Shop





# Flatpak erstellen

- > SDK mit `flatpak-builder` benötigt
  - > kann via Flatpak installiert werden
- > **YAML**-Manifest definiert Quellen (u.a. `archive`, `git`, `file`, `patch`) und Build-Module
- > u.a. [unterstützte Buildsysteme](#):
  - > `autotools`, `cmake`, `meson`, `qmake`,...

# Flatpak erstellen

- > SDK mit `flatpak-builder` benötigt
  - > kann via Flatpak installiert werden
- > **YAML**-Manifest definiert Quellen (u.a. `archive`, `git`, `file`, `patch`) und Build-Module
- > u.a. [unterstützte Buildsysteme](#):
  - > `autotools`, `cmake`, `meson`, `qmake`,...
- > Erstellen und **Testen** des Pakets
- > Paket in **Repository** hinterlegen
- > Repository zu lokalen Quellen hinzufügen und Paket **installieren**

# Beispiel

Installieren des SDK:

```
$ flatpak install org.freedesktop.Sdk
```

Erstellen der Anwendung und des Manifests:

hello.sh:

```
#!/bin/sh  
echo "Ohai Kielux"
```

# Beispiel

`org.flatpak.Hello.yml:`

```
app-id: org.flatpak.Hello
runtime: org.freedesktop.Platform
runtime-version: '21.08'
sdk: org.freedesktop.Sdk
command: hello.sh
modules:
  - name: hello
    buildsystem: simple
    build-commands:
      - install -D hello.sh /app/bin/hello.sh
    sources:
      - type: file
        path: hello.sh
```

# Beispiel

Erstellen des Pakets:

```
$ flatpak-builder build-dir org.flatpak.Hello.yml
```

```
Downloading sources
```

```
Initializing build dir
```

```
Committing stage init to cache
```

```
Starting build of org.flatpak.Hello
```

```
=====
```

```
Building module hello in /home/cstankow/Dokumente/Lab/Flatpak/.flatpak-builder/b
```

```
=====
```

```
Running: install -D hello.sh /app/bin/hello.sh
```

```
Committing stage build-hello to cache
```

```
Cleaning up
```

```
Committing stage cleanup to cache
```

```
Finishing app
```

```
Please review the exported files and the metadata
```

```
Committing stage finish to cache
```

```
Pruning cache
```

# Beispiel

Anschließend existiert ein neuer Ordner `build-dir`:

```
$ tree build-dir/  
build-dir/  
├── export  
├── files  
│   ├── bin  
│   │   └── hello.sh  
│   └── manifest.json  
├── metadata  
└── var  
    ├── lib  
    ├── run -> /run  
    └── tmp
```

# Beispiel

## Testen der Anwendung:

```
$ flatpak-builder --user --install --force-clean build-dir org.flatpak.Hello.yml
Emptying app dir 'build-dir'
Downloading sources
Starting build of org.flatpak.Hello
...
Installing app/org.flatpak.Hello/x86_64/master
Pruning cache
```

```
$ flatpak run org.flatpak.Hello
Ohai Kielux
```

# Beispiel

## Anwendung in Repository veröffentlichen:

```
$ flatpak-builder --repo=repo --force-clean build-dir org.flatpak.Hello.yml
...
Exporting org.flatpak.Hello to repo
Commit: 3217d6e09036fbd0165014daae6fc3c8ead90868294d2d56e57a633714829fe4
```

## Repository einhängen und Anwendung installieren:

```
$ flatpak --user remote-add --no-gpg-verify tutorial-repo repo
$ flatpak --user install tutorial-repo org.flatpak.Hello
Installation complete.
$ flatpak run org.flatpak.Hello
Ohai Kielux
```



# Snapcraft

# Snapcraft

- > 2014 von Canonical vorgestellt
- > Fokus auf Desktop-Anwendungen, Serverdienste und Drucker-Stack
  - > ursprünglich für **IoT** entwickelt
  - > nutzt systemd-Features wie **Socket Activation**
- > **Runtime** benötigt, Installation notwendig
  - > Hintergrunddienst `snapped` aktualisiert Snaps automatisch
  - > lässt sich **nicht** deaktivieren, maximal um 60 Tage verzögern



# Snapcraft

- > Apps werden in **Sandbox** ausgeführt
  - > Rechte via **XDG Desktop Portals** und AppArmor
  - > kein SELinux-Support bisher\*
  - > auch Themes müssen dediziert verteilt werden
- > viele Anwendungen im **proprietären** [Snap Store](#)
  - > automatisches Testen der Apps auf Malware
  - > trotzdem gab es [2018 Apps mit Cryptominer](#)

\* EPEL bietet [snapd-selinux](#)



# Snapcraft

- > SquashFS-Image mit `.snap`-Dateiendung
  - > enthält Anwendung und benötigte Bibliotheken
  - > Image wird eingehängt, Dateien on-demand entpackt
  - > unterstützt **XZ** (Standard, geringere Größe) und **LZO** (größer aber schneller zu entpacken)
- > nicht nur für Ubuntu, auch für RHEL- und Debian-artig, sowie Fedora, openSUSE, Solus und ArchLinux
- > bietet **transaktionale Updates**



# Snap! - Marketing is a dancer

- > **Snap** - Anwendung + Abhängigkeiten, damit diese ohne Anpassungen auf verschiedenen Plattformen läuft
- > **snapd** - Hintergrunddienst, verwaltet Snaps automatisch
- > **Snap Store** - Online-Store, zentral, proprietär
- > **Snapcraft** - Framework/Anwendung zum Bauen von Anwendungen
- > **Channel** - Veröffentlichungskanal: `<track>/<risk>/<branch>`
  - > `track`: ein unterstütztes Release, z.B. `latest`, `insider`
  - > `risk`: Stabilität; `stable`, `candidate`, `beta` oder `edge`
  - > `branch`: Entwicklungszweig; z.B. `fix-bug-1337`
  - > Beispiele: `latest/stable`, `insider/edge`

# Neverending Story: Snap + Firefox

- > ab Ubuntu 21.10 liefert Ubuntu Firefox **primär** als Snap anzubieten
  - > seit Ubuntu 22.04 wird Firefox **ausschließlich** als Snap angeboten
- > **Vorteil:** Mozilla paketiert Firefox, weniger Verantwortung für Canonical

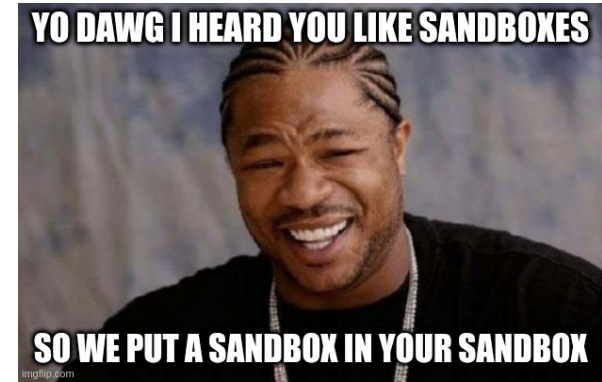
# Neverending Story: Snap + Firefox

- > ab Ubuntu 21.10 liefert Ubuntu Firefox **primär** als Snap anzubieten
  - > seit Ubuntu 22.04 wird Firefox **ausschließlich** als Snap angeboten
- > **Vorteil:** Mozilla paketiert Firefox, weniger Verantwortung für Canonical
- > Nachteil: deutlich **langsamere** Firefox-Startzeiten\*
  - > Kaltstart: 21s vs. 8s
  - > Normaler Start: 8.5s vs 2.9s
  - > Jetstream2-Benchmark: 64.804 vs 67.563
  - > inzwischen (**2023**) etwas besser geworden

\* getestet auf i7-8850H @ 2 vCPUs, 2 GB RAM; siehe auch [Blogpost](#)

# Neverending Story: Snap + Firefox

- > Canonical rechtfertigte die Entscheidung
  - > zusätzliche Sicherheit durch weitere Sandbox
- > Paket entpackt beim ersten Start **alle 98** verfügbaren Sprachpakete
- > Es wurde die **XZ**-Kompression verwendet
- > Das Firefox 100.0-Snap aktivierte die Compilerflags **PGO** und **LTO** aktiviert
  - > leicht schnellere Lauf-/Startzeiten
- > Bei RPi/AMD wurden die falschen GPU-Treiber erkannt  $\Rightarrow$  ineffizientes **Software rendering**





# Neverending Story: Snap + Firefox

- > Canonical und Mozilla haben Snaps überarbeitet
  - > Firefox lädt nur noch benötigte Übersetzungen herunter
  - > GTK-Snaps nutzen nun **LZO** statt XZ\*
- > ⇒ **50%** schnellere Kaltstarts auf RPi, **29-42%** auf x86

# Neverending Story: Snap + Firefox

- > Canonical und Mozilla haben Snaps überarbeitet
  - > Firefox lädt nur noch benötigte Übersetzungen herunter
  - > GTK-Snaps nutzen nun **LZO** statt XZ\*
- >  $\Rightarrow$  **50%** schnellere Kaltstarts auf RPi, **29-42%** auf x86
- > Weitere geplante Optimierungen:
  - > SquashFS Dekomprimierung ist unter Ubuntu **single-threaded**
  - > Anpassung der Kernelmodul-Konfiguration notwendig
  - > **Pre-Caching** für GTK-Anwendungen geplant

\* auch andere Snaps sollten dadurch profitieren

# Snap-Pakete erstellen

- > Metadaten werden in **YAML**-Datei definiert
  - > Name des Pakets
  - > Version und Beschreibung
  - > Berechtigungsmodell (`strict`, `devmode`, `classic`)
- > Erfordert installiertes [Multipass](#)\*
  - > erstellt Ubuntu VMs unter Linux, macOS und Windows
- > Erstellt eine `.snap`-Datei

\* Poor man's [Vagrant](#)?

# Beispiel

```
name: test-offlineimap-dummy
version: '1.0'
summary: OfflineIMAP
description: |
  OfflineIMAP is software that downloads your email mailbox(es) as local
  Maildirs. OfflineIMAP will synchronize both sides via IMAP.
confinement: devmode
base: core18
parts:
  test-offlineimap-dummy:
    plugin: python
    python-version: python2
    source: https://github.com/snapcraft-docs/offlineimap.git
    stage-packages:
      - python-six
apps:
  test-offlineimap-dummy:
    command: bin/offlineimap
```

# Beispiel

Zur Erstellung muss die Datei `snapcraft.yaml` in einen sinnvoll benannten Unterordner liegen:

```
$ cd test-offlineimap-dummy  
$ snapcraft
```

Hierbei wird eine Ubuntu-Instanz via **LXD** gestartet.

Die fertige Datei kann dann auf Systemen installiert werden:

```
# snap install --devmode --dangerous *.snap
```

# Ausblick

# Zusammenfassung

	AppImage	Flatpak	Snapcraft
Erschienen	2004	2007	2014
Autor:in	Simon Peter, Community	Flatpak-Team	Canonical
Fokus	Endanwender:innen	Desktop	Desktop, Dienste, Drucker
Runtime benötigt?	Nein	Ja	Ja
Installation notwendig?	Nein	Ja	Ja
Sandbox	Nein	Ja	Ja
Format	SquashFS	OSTree/OCI	SquashFS
Rechteverwaltung	Nein	Ja (XDG)	Ja (XDG, AppArmor)
Store	<a href="#">AppImageHub</a>	<a href="#">Flathub</a>	<a href="#">Snapcraft</a>
Angebot	ca. 1.400 Apps	ca. 2.300 Apps	~\(\ツ)/~
Updates	Neues Image bzw. Binary Delta	Neues Image	Transaktionale Updates

# Fazit

- > AppImage ist für Desktop-Apps die **schlankste** und einfachste Lösung
- > Flatpaks sind weit verbreitet, integrieren sich in **Software-Shops**
- > Snapcraft kämpft mit technischen Details und Akzeptanz
- > Canonical hätte **proaktiver**/schneller auf Feedback eingehen müssen
- > Quelloffen + Dezentral > Proprietäre Shops
- > Umweg über Multipass/LXD unnötig **komplex**
- > Web-Browser sind kritische Anwendungen, die weiterhin vom Distributor ausgeliefert werden sollten





# Links

- > [ApplImage-Wiki zur Paketierung](#)
- > [ApplImageHub](#)
- > [Interview mit Simon Peter](#)
- > [SUSECON 2017-Vortrag über ApplImage](#)
- > [Flathub](#)
- > [Flatpak-Dokumentation](#)
- > [Flatpak-Tutorial](#)
- > [Snapcraft](#)
- > [Snapcraft-Dokumentation](#)
- > [Snap Hello World](#)

# FOCUS ON: Linux

Themen wie diese könnt ihr hier alle 2 Wochen hören:

- > News des Monats
- > Tooltipps
- > Thematische Sonderfolgen

Verfügbar via:

- > [RSS](#) / [fyyd](#)
- > [Apple Podcasts](#)
- > [Spotify](#)



# Danke für die Aufmerksamkeit (Fragen?)

