Menadžment i distribucija softverskih paketa na GNU/Linux platformi

Ptiček, Goran

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Goran Ptiček

MENADŽMENT I DISTRIBUCIJA SOFTVERSKIH PAKETA NA GNU/LINUX PLATFORMI

ZAVRŠNI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Goran Ptiček

Matični broj: 45917/17-R

Studij: Informacijski sustavi

MENADŽMENT I DISTRIBUCIJA SOFTVERSKIH PAKETA NA GNU/LINUX PLATFORMI

ZAVRŠNI RAD

Mentor :

Dr. sc. Miran Zlatović

Varaždin, kolovoz 2021.

Goran Ptiček

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

U svijetu postoji velik broj programa koje često nije lako korektno i/ili u zadanom vremenu postaviti. U radu se pokušava pronaći odgovor na navedeni problem korištenjem XBPS menadžera softverskih paketa, usporedbom istog s ostalim rješenjima, izraditi vlastite softverske pakete programa otvorenog i zatvorenog koda te u konačnici postaviti poslužitelj za izgradnju i distribuciju tih paketa.

Ključne riječi: linux; paket; xbps; void; docker; poslužitelj; administracija;

Sadržaj

1. Uvod 1

2. Metode i tehnike rada 2

3. GNU/Linux 3
3.1. Hijerarhija datotečnog sustava 3
3.2. Dodavanje novih programa 5

4. Menadžeri paketa i njihovo korištenje 7
4.1. Na razini distribucije 7
4.1.1. Apk 8
4.2. Neovisni o distribuciji 9
4.2.1. Flatpak 9
4.2.2. Snap 12
4.2.3. Appimage 13
4.3. Neovisni o operacijskom sustavu 14
4.3.1. Docker 14

5. XBPS 18
5.1. Korištenje 18
5.1.1. xbps-query 18
5.1.2. xbps-install 19
5.1.3. xbps-remove 21
5.1.4. xbps-pkgdb 21
5.1.5. xtools 22
5.1.5.1. xlocate 22
5.1.5.2. xdowngrade 23
5.2. Xbps-src 25
5.2.1. Postavljanje 25
5.2.2. Izgradnja paketa 26
5.2.3. Stvaranje novih paketa 28
5.2.3.1. Siege 30
5.2.3.2. Jaxx 34
5.3. Distribucija paketa 36
5.3.1. Postavljanje poslužitelja 36
5.3.2. Postavljanje repozitorija 37
5.4. Zašto xbps-src? 38

6. Zaključak 40

Popis literature 42

Popis slika 43

Popis tablica 44

1. Uvod

Danas postoji velik broj dostupnih programa koje korisnici mogu instalirati i koristiti na vlastitim računalima, no menadžment istih često je kaotičan i u potpunosti ovisan o samom stvaratelju tog programa.

Instaliranje dodatnog software-a klasično se vrši posjetom službene stranice željenog programa, preuzimanjem instalacijskog paketa te u konačnici njegove instalacije, a ona u većini slučajeva sadrži i vlastiti servis za provjeru dostupnih ažuriranja kako korisnik ne bi morao ručno preuzimati ažuriranja za svaku novu verziju tog programa. Da bi programeri ubrzali razvoj software-a, oni koriste vanjske biblioteke koje opet mogu koristiti druge biblioteke. Ako program nije statično kompajliran, te biblioteke moraju mu biti dostupne prilikom njegovog rada, a isto osigurava instalacijski paket.

Vlastiti načini pakiranja programa i njihovih pozadinskih servisa za provjeru dostupnih ažuriranja dovode do manje efikasnosti, veće potrošnje računalnih resursa, doprinose fragmentaciji razvojnog okruženja te općenito čine proces razvoja, održavanja i distribucije programa težim.

Navedeni problemi mogu se rješiti korištenjem menadžera paketa koji su nastali standardizacijom postupka pakiranja i distribucije programa, a uglavnom ovise o operacijskom sustavu, no neki su agnostični o platformi jer koriste neki oblik virtualizacije tako da rade na svakoj platformi (npr. Docker). Na taj način, svakom programu osigurana je dostupnost njemu potrebnih biblioteka i dijeljenje istih s drugim programima kako bi se uštedjelo na pohrambenom prosturu i radnoj memoriji te jasno definiralo gdje se one pohranjuju. Nadalje, korištenje centraliziranog repozitorija svih tako pakiranih programa omogućava korisniku preuzimanje istih s jednog mjesta što pridonosi jednostavnosti, ali i sigurnosti jer administratori repozitorija mogu potpisati pakete koristeći asimetričnu kriptografiju pomoću koje menadžer paketa može verificirati autentičnost prenesenog programa. Dodatno, menadžer paketa programeru olakšava proces distribucije ažuriranja kojeg sad ne mora implementirati budući da ažuriranja svih korisničkih programa vrši menadžer paketa.

Cilj ovog završnog rada upoznati je čitatelje sa trenutnim stanjem menadžmenta paketa u GNU/Linux okruženju, pokazati najkorištenije menadžere paketa i način njihova korištenja te pokazati na primjeru kako izvršiti pakiranje i distribuciju programa otvorenog i zatvorenog koda koristeći XBPS menadžer paketa.

2. Metode i tehnike rada

Svaka GNU/Linux distribucija ima vlastiti repozitorij dostupnih programa koji su izgrađeni korištenjem njezinog menadžer paketa. U ovom radu primarno se razmatra XBPS menadžer paketa te se koristi Void Linux distribucija za provođenje iste. Većina praktične demonstracije interoperabilna je s ostalim distribucijama. Budući da Void Linux koristi chroot virtualno okruženje s vlastitim programima i bibliotekama kod izgradnje paketa, ista je moguća na bilo kojoj distribuciji.

U radu se uzima pristup inkramentalne demonstracije svakog koraka izvođenja neke radnje u blokovima koda. Svaki red u bloku koda koji započinje s *#* ili *$* znakom odnosi se na izvršavanje naredbe u naredbenom retku, a razlika je u tome što *#* indicira da su potrebne administrativne privilegije za izvršavanje, dok za *$* nisu. Dodatno, svaki readak koji započinje s ... indicira na skraćivanje ispisa izvršene naredbe.

Rad većinski koristi online resurse. Sama distribucija izgrađenih paketa provodi se na Raspberry Pi 3 B+ računalu.

3. GNU/Linux

Prema Stallman [1], Linux je kernel odnosno jezgra sustava koja omogućava interakciju između sklopovlja i svih programa, no on je sam po sebi beskoristan i dobiva smisao samo u konekstu operacijskog sustava. GNU je jedan od tih operacijskih sustava, Linux je jedan od njegovih jezgara, a zajedno oni čine GNU/Linux sustav.

Menadžer paketa je program koji omogućava instalaciju, nadogradnju i uklanjanje programa u operacijskom sustavu. Instalacija omogućava dodavanje programa operacijskom sustavu s lokalnog paketa ili udaljenog poslužitelja pri čemu se automatski verificira autentičnost i integritet preuzetih paketa. Menadžer paketa vodi popis potrebnih biblioteka i njihovih verzija za svaki program. [2, str. 542]

Odvojenosti kernela i samog operacijskog sustava temelj je inovacije u GNU/Linux sferi zbog koje bilo tko može dizajnirati vlastito okruženje oko istog kernela. Tako danas postoji velik broj distribucija s različitim stavovima u odnosu na izbor init sustava, menadžera paketa, initramfs generatora itd. Veći broj distribucija nudi veliku količinu izbora krajnjim korisnicima koji mogu odabrati sustav koji najbolje odgovara njihovim potrebama, no isto može i dovesti do problema; fragmentacije. Svaki menadžer paketa ima svoj način pakiranja programa pa se zbog toga isti program mora pakirati za više različitih platforma kako bi program bio svima dostupan. Npr., ako želimo učiniti naš program dostupnim korisnicim Ubuntu, Fedora i Void Linux distribucija, trebamo napraviti pakete za apt, dnf i xbps menadžere paketa. Potencijalno rješenje ovog problema adresirano je korištenjem univerzalnih menadžera paketa, opisano u sekciji 4.2.

Void Linux jedan je od GNU/Linux distribucija, no nije derivacija postojećih (npr. Manjaro je baziran na Arch Linux distribuciji). Void teži minimalizmu sustava i deverzitetu biblioteka C jezika (trenutno nudi gotovo sve programske pakete u musl i GNU libc izdanju), koristi runit init sustav i ima vlastiti menadžer paketa XBPS. [3]

3.1. Hijerarhija datotečnog sustava

Kako bi čitatelji mogli pratiti ostatak rada, potrebno je osvrnuti se na hijerarhiju datotečnog sustava kod GNU/Linux distribucija (eng. *File System Hierarchy*). GNU/Linux distribucije okvirno se pridržavaju standarda hijerarhije datotečnog sustava koji definira razmještaj datoteka prema njihovim kategorijama u različite direktorije.

/bin, /sbin, /usr/bin

Ovi direktoriji sadržavaju sve izvršne datoteke operacijskog sustava. Tradicionalno, najvažnije sustavske izvršne datoteke nalazile su se u `/sbin` direktoriju, ostale u `/bin` direktoriju, a ne esencijalne u `/usr/bin` direktoriju. Danas, kategorizacija izvršnih programa ovisi o korištenoj distribuciji. U slučaju Void Linux-a, sve izvršne datoteke nalaze se u `/usr/bin` datoteci, a `/bin` i `/sbin` su simbolične poveznice na istu. Neke izvršne datoteke nalaze se i u `/lib/libexec` jer nisu namijenjene izravnom korištenju od

strane korisnika, već budu pozivane od drugih programa koriteći IPC (eng. *Inter Process Communication*), a korištena implementacija IPC-a danas je Dbus.

/boot

Sadrži sve potrebne datoteke za pokretanje GNU/Linux sustava (sam Linux, bootloader i initramfs). Dodatno, u `/lib/modules` se nalaze relevantni moduli Linux kernela.

/dev

U GNU/Linux sustavima, sve komponente računala reprezentirane su u obliku datoteka koje se nalaze u ovom direktoriju.

/etc

Sve konfiguracijske datoteke programa mogu se pronaći u ovom direktoriju. Dodatno, većina programa podržava korisničku konfiguraciju unutra /home direktorija.

/home

Home direktorij sadrži sve korisničke datoteke.

/lib, /lib32, /lib64, /usr/lib, /usr/lib32, /usr/lib64

Ovi direktoriji sadržavaju sve biblioteke potrebne za funkcioniranje programa. Opet, njihov razmještaj ovisi o distribuciji. Void Linux sve 32 bitne biblioteke sprema u `/usr/lib32`, a 64 bitne u `/usr/lib`. Svi ostali direktoriji simbolične su poveznice na navedene direktorije.

/media, /mnt

Ovi direktoriji ne sadržavaju ništa, već se koriste kao pomoćni direktoriji za montiranje medija za pohranu podataka.

/opt

Opt direktorij je prazan i namijenjen je pohranjivanju ručno instaliranih programa (bez korištenja menadžera paketa).

/proc

Virtualna datoteka koja prikazuje različite informacije Linux kernela (npr. `/proc/cmdline` sadrži niz variajabla potrebnih za pokretanje operacijskog sustava).

/root

Root direktorij sadrži sve datoteke administrativnog računa.

/run

Run sadrži privremene datoteke sustava (identifikacijske brojeve sustavskih procesa, datoteke korisničke sjednice itd.).

/srv

Ovdje se pohranjuju datoteke web poslužitelja (web stranice). Alternativno, te datoteke se znaju nalaziti u `/var/www/html`.

/sys

Virtualna datoteka koja sadrži različite informacije o sklopovlju računala i priključenim uređajima.

/tmp

Sadrži privremene datoteke koje se brišu kod pokretanja računala (kod nekih distribucija, ovaj direktorij montiran je u radnoj memoriji pa brisanje nije potrebno). Dodatno, korisničke privremene datoteke mogu se pohranjivati u `/home/korisnik/.cache`.

/usr

Usr sadrži datoteke koje se ne mijenjaju nakon instalacije (izvršne datoteke, dokumentaciju programa dostupnih preko *man* naredbe, biblioteke itd.)

/var

Sadrži sve često mijenjane datoteke (sustavski zapisnik, cache menadžera paketa itd.).

3.2. Dodavanje novih programa

Operacijski sustavi generalno se distribuiraju sa različitim setom preinstaliranih programa za osnovne aktivnosti, no u većini slučajeva oni nisu dostatni za potrebe korisnika. Instalacija programa odnosi se se na uključivanje izvršnih datoteka, svih potrebnih biblioteka i dokumentacije u direktorije operacijskog sustava. Za instalaciju novih programa poželjno je koristiti menadžer paketa distribucije budući da je onda ažuriranje programa jednostavno i sigurnu u smislu praćenja potrebnih biblioteka. Loši menadžment dijeljenih biblioteka može dovesti do konflikta i prestanka rada programa. Npr. ako imamo programe *A* i *B* od kojih oboje trebaju biblioteku *C* za rad, ali program *A* treba noviju verziju biblioteke *C* koja nije kompatibilna s verzijom koju treba program *B*, dolazi do konflikta u kojem će jedan program raditi, a drugi ne.

Program može biti instaliran za sve korisnike ili za korisnika individualno. Kad se govori o menadžeru paketa, misli se na menadžment paketa za sve korisnike za koje su potrebne administrativne (eng. *root*) privilegije. Ukratko, većina direktorija GNU/Linux sustava zahtjeva administrativne privilegije za modificiranje i dodavanje sadržaja što doprinosi sigurnosti sustava, a u `/home` svaki korisnik ima vlastiti direktorij u kojem može dodavati, modificirati i brisati datoteke bez dodatnih privilegija.

Kad je program instaliran za sve korisnike, njegove izvršne datoteke sustavu su dostupne preko *PATH* varijable iz globalnog okruženja. *PATH* varijabla sadržava niz putanja do izvršnih datoteka koje su odvojene dvotočkom, pri čemu je važan njihov redoslijed. Kad bi korisnik pozvao naredbu *ls*, sustav bi tražio njezinu izvršnu datoteku redom u direktorijima specificiranih u PATH varijabli. Globalnu PATH varijablu može mijenjati samo administrator, no svaki korisnik može sam specificirati vlastitu unutar profila njegovog interaktivnog terminala. Ta značajka omogućava svakom korisniku da preuzme program s udaljenog poslužitelja i postavi ga u svoj direktorij koji je dodao svojoj PATH varijabli i na taj način pozivati taj program iz bilo kojeg drugog direktorija. Naravno, program se uvijek može pokrenuti u lokalnoj datoteci koristeći *./program* notaciju. Sve navedeno vrijedi za same izvršne datoteke, no korisnici ih ne pozivaju direktno kad koriste grafičko okruženje, već pozivaju pokretače s *.desktop* sufiksom koji onda pozivaju te izvršne datoteke. Ti pokretači isto mogu biti definirani od strane korisnika u `~/.local/share/applications` direktoriju. Korištenje program bez uključivanja u sustavsko okruženje ima svojih ograničenja jer, kad program bude kompleksniji i zahtjeva različite

vanjske biblioteke, često je kodiran da pronalazi iste u standardiziranim direktorijama sustava. Ovo se opet može izbjeći koristeći "chroot" okruženje što je van sklopa ove sekcije.

4. Menadžeri paketa i njihovo korištenje

U ovoj sekciji prikazat će se različiti menadžeri paketa i njihovo korištenje. U radu su menadžeri paketa podijeljeni u tri kategorije na temelju njihove dostupnosti na različitim operacijskim sustavima ili GNU/Linux distribucijama.

4.1. Na razini distribucije

Najveći broj menadžera paketa dizajnirani su za rad na određenoj distribuciji. Neki menadžeri paketa rade na samo jednoj distribuciji (APK za Alpine Linux), dok su drugi u korištenju kod više različitih distribucija (DNF koristi Fedora, RHEL, Centos). U tablici nisu navedeni svi derivati distribucija, već oni najrelevantniji. Naravno, postoji puno više menadžera paketa, no navedeni su izabrani jer su općenito najčešće korišteni.

U tablici se nalaze samo menadžeri paketa dizajnirani za GNU/Linux sustave, izuzev Alpine distribucije koje ne koristi GNU alate i C biblioteku, već busybox i musl zbog čega minimalna instalacija ne zauzima više od 8MB. Zbog tog razloga Alpine se često koristi u docker kontejnerima. [4]

Tablica 1: Osnovno korištenje menadžera paketa distribucije

	Alpine	Arch/Manjaro	Debian/Ubuntu	Fedora/Centos
Sinkroniziranje	apk update	pacman -Sy	apt update	dnf update
Instalacija	apk add	pacman -S	apt install	dnf install
Nadogradnja	apk upgrade	pacman -Su	apt upgrade	dnf upgrade
Deinstalacija	apk del	pacman -Rs	apt remove	dnf remove
Traženje	apk search	pacman -Ss	apt search	dnf search

Sinkroniziranje se odnosi na preuzimanje indeksa repozitorija koji sadrži popis svih paketa, njihovih verzija i njima potrebnih biblioteka na temelju čega se može vršiti instalacija, nadogradnja i uklanjanje paketa. U tablici su sve funkcije prikazene odvojeno za sebe iako se često zajedno kombiniraju. Npr. za instalaciju novog paketa na Arch distribuciji nije nužno prvo sinkronizirati lokalni indeks s indeksom repozitorija, već je dovoljno pozvati *pacman -Syu program*. Kod apt menadžera paketa, nužno je prvo sinkronizirati indekse i tek onda pozvati *apt install program* za instalaciju programa. Slično vrijedi i za nadogradnju i pretraživanje dostupnih programa.

Općenito je preporučeno korištenje menadžera paketa na razini distribucije budući da taj način nudi najveću razinu integracije svih programa te dovodi do najmanjeg zauzimanja prostora u odnosu na druge metode jer onda paketi mogu dijeliti potrebne biblioteke. Dodatno, svi paketi ponuđeni u sklopu menadžera paketa distribucije izgrađeni su na poslužiteljima same distribucije te su kriptografski potpisani za dodanu sigurnost zbog čega se ne mora vjerovati autentičnosti programa progamera u smislu pitanja da li je njegov / njezin ponuđen paket nekog programa uistinu deriviran od ponuđenog izvornog koda jer distribucija nudi predloške za

izgradnju svakog paketa čime se postiže reproduktibilnosti krajnjeg paketa.

4.1.1. Apk

Sljedeće naredbene izvršene su unutar docker Alpine kontejnera. Postavljanje istog opisano je u sekciji 4.3.1. Primjer nadogradnje svih instaliranih paketa, traženja, instalacije i deinstalacije *neovim* editora teksta na Alpine distribuciji:

```
# apk upgrade
fetch https://dl-cdn.alpinelinux.org/alpine/v3.14/main/aarch64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.14/community/aarch64/APKINDEX.tar.
    gz
Upgrading critical system libraries and apk-tools:
(1/1) Upgrading apk-tools (2.12.5-r1 -> 2.12.7-r0)
Executing busybox-1.33.1-r2.trigger
Continuing the upgrade transaction with new apk-tools:
(1/4) Upgrading busybox (1.33.1-r2 -> 1.33.1-r3)
Executing busybox-1.33.1-r3.post-upgrade
(2/4) Upgrading alpine-baselayout (3.2.0-r15 -> 3.2.0-r16)
Executing alpine-baselayout-3.2.0-r16.pre-upgrade
Executing alpine-baselayout-3.2.0-r16.post-upgrade
(3/4) Upgrading libretls (3.3.3-r0 -> 3.3.3p1-r2)
(4/4) Upgrading ssl_client (1.33.1-r2 -> 1.33.1-r3)
Executing busybox-1.33.1-r3.trigger
OK: 5 MiB in 14 package

$ apk search neovim
neovim-doc-0.4.4-r1
neovim-lang-0.4.4-r1
fzf-neovim-0.27.2-r1
neovim-0.4.4-r1
py3-pynvim-0.4.3-r2

# apk install neovim
(1/9) Installing libintl (0.21-r0)
(2/9) Installing lua5.1-libs (5.1.5-r7)
(3/9) Installing libuv (1.41.0-r0)
(4/9) Installing libluv (1.36.0.0-r3)
(5/9) Installing msgpack-c (3.3.0-r0)
(6/9) Installing unibilium (2.1.0-r0)
(7/9) Installing libtermkey (0.22-r0)
(8/9) Installing libvterm (0.1.20190920-r1)
(9/9) Installing neovim (0.4.4-r1)
Executing busybox-1.33.1-r3.trigger
OK: 22 MiB in 23 packages

# apk del neovim
(1/9) Purging neovim (0.4.4-r1)
(2/9) Purging libintl (0.21-r0)
(3/9) Purging lua5.1-libs (5.1.5-r7)
(4/9) Purging libluv (1.36.0.0-r3)
(5/9) Purging libuv (1.41.0-r0)
```

```
(6/9) Purging msgpack-c (3.3.0-r0)
(7/9) Purging libtermkey (0.22-r0)
(8/9) Purging unibilium (2.1.0-r0)
(9/9) Purging libvterm (0.1.20190920-r1)
Executing busybox-1.33.1-r3.trigger
OK: 5 MiB in 14 packages
```

Općenito, za sve dodatne opcije menadžera paketa, može se koristiti priručnik pozivom naredbe *man*. Kod Alpine distribucije, priručnici nisu uključeni pa ih je prvo potrebno preuzeti.

```
# apk add mandoc man-pages apk-tools-doc
```

Sada se može pregledati priručnik za apk:

```
$ man apk
```

4.2. Neovisni o distribuciji

Svaki menadžer paketa koristi svoj vlastiti način pakiranja paketa te je zbog toga svaki program potrebno pakirati za svaku ciljanu distribuciju. Dodatna nuspojava ovog pristupa je da isti program na jednoj distribuciju neće nužno imati iste mogućnosti kao onaj na drugoj distribuciji jer se tamo koriste drugi set zastavica prilikom kompajliranja paketa. Na primjer, postoji program *sqlitebrowser* koji omogućava pregled sqlite baza podataka kroz grafičko sučelje. Kod Void Linux distribucije, *sqlitebrowser* paket nije izgrađen s *sqlcipher* bibliotekom zbog čega otvaranje enkriptiranih sqlite baza podataka nije podržano. Na nekoj drugoj distribuciji koju je korisnik prije koristio, ova značajka možda je omogućena pa će tom korisniku, u slučaju prelaska na Void distribuciju, novo ponašanje programa biti neočekivano. U ovom slučaju, moguće je izgraditi paket s dodatnom bibliotekom *sqlcipher-devel* kako bi se omogućila ta značajka, no na nekoj drugoj distribuciji to možda uopće neće biti moguće ako ona ne nudi tu biblioteku u svojem repozitoriju.

Uz sve navedeno, postoji i dodatni problem licenca koji otežava pakiranje i distribuciju određenih programa. Popularan program *discord*, prema uvjetima korištenja [5], ne dozvoljava distribuciju programa. Zbog toga, Void ne distribuira sam programski paket, ali nudi predložak prema kojem se paket za *discord* može izgraditi i instalirati koristeći *xbps-src*. Više o tome u sekciji 5.2.2.

4.2.1. Flatpak

Flatpak je manadžer paketa koji omogućava programerima stvaranje samo jednog paketa kojeg zatim mogu distribuirati na svim distribucijama. Samo okruženje u kojem je takav paket izgrađen jednako je onome kojeg dobiva korisnik. Naime, flatpak paketi imaju svojevrsno virtualno okruženje koje sadrži sam program i sve njemu potrebne biblioteke. To je glavna razlika u odnosu na tradicionalne menadžere paketa u kojima programer nema kontrolu nad bibliotekama koje korisnik koristi, već to ovisi o izgraditelju paketa. Zbog toga, prije naveden

problem biblioteka ne postoji jer će svatko tko koristi isti flatpak paket određenog programa imati isto okruženje kao i svi drugi. [6]

Dodatna prednost flatpak menadžera paketa je u mogućnosti ograničavanja programa u smislu pristupa datotekama i ostalim sučeljima operacijskog sustava (npr. može se ograničiti pristup Dbus-u, kameri, mikrofonu i slično). Osim toga, flatpak nije centraliziran i nakon instalacije nema konfiguriranih repozitorija, već se oni moraju ručno dodati.

Uz sve prednosti, flatpak ima i određene negativne elemente. Naime, zbog toga što se svaki program nalazi u vlastitom izoliranom okruženju, iste biblioteke nisu dijeljene, već moraju biti uključene sa svakim paketom. To dovodi do većeg zauzamnja pohrambenog prostora. Dodatno, zbog toga što sam programer odlučuje o uključenim bibliotekama, one često mogu biti zastarjele i predstavljat sigurnosnu rupu. Iako flatpak nudi mogućnost dodatnih ograničenja, ona često nisu postavljena za novo instalirane programe, a konfiguracija istih nije nužno jednostavna, no postoji flatpak paket Flatseal [7] koji nudi grafičko sučelje za podešavanje dozvola. Postoji dodatan problem tema flatpak aplikacije jer one ne koriste teme sustava. Zbog toga, flatpak aplikacije mogu izgledati sasvim drugačije.

Sve operacije kod flatpak menadžera paketa mogu se izvršiti na razini sustava ili na razini korisnika. U prvom slučaju, instaliran paket bit će dostupan svima, dok će u drugom slučaju biti dostupan samo tom jednom korisniku. U navedenim primjerima koristit će se korisnički pristup. Za korištenje korisničkog načina rada, potrebno je dodati *–user* zastavicu svakoj naredbi.

Tablica 2: Osnovno korištenje Flatpak menadžera paketa

	Flatpak
Instalacija	flatpak install
Nadogradnja	flatpak update
Deinstalacija	flatpak uninstall
Traženje	flatpak search

Kod Void distribucije, flatpak se može instalirati sa:

```
# xbps-install -S flatpak
...
```

Zatim je potrebno dodati repozitorij. Najveći i najkorišteniji repozitorij trenutno je Flathub. [8]

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

Sada možemo instalirati npr. Element aplikaciju koja se koristi za online chat i pozive.
Možemo pretražiti dodane repozitorije s (ispis naredbe skraćen):

```
 $ flatpak search element
Name    Description                Application ID      Remotes
elemen  The Gtk+ Stylesheet for    3theme.elementary   flathub
Element Create, share, communicat  im.riot.Riot        flathub
```

Tražena aplikacija postoji i njezin identifikator je *im.riot.Riot*. Kako bismo ju instalirali, pišemo:

```
$ flatpak install --user im.riot.Riot
Found similar ref(s) for im.riot.Riot in remote flathub (user).
Use this remote? [Y/n]:
$ y
im.riot.Riot permissions: ipc network pulseaudio wayland x11 devices file access
    [1] dbus access [2] bus ownership [3]
[1] xdg-download, xdg-run/keyring, xdg-run/pipewire-0
[2] org.freedesktop.Notifications, org.freedesktop.portal.Fcitx, org.kde.
    StatusNotifierWatcher
[3] org.kde.StatusNotifierItem-2-1

       ID                  Branch         Op        Remote         Download
1.     im.riot.Riot        stable         i         flathub        <  111.0MB

Proceed with these changes to the user installation? [Y/n]:
$ y
...
Installation complete.
```

Aplikacija je sada instalirana. Općenito, svi podaci flatpak aplikacija pohranjuju se u `~/.var/app/appID` direktoriju. Aplikacija se sada može pokrenuti preko pokretača dostupnog u grafičkom sučelju sustava ili preko naredbe:

```
$ flatpak run im.riot.Riot
```

Da bismo deinstalirali aplikaciju, pišemo:

```
$ flatpak uninstall --user io.riot.Riot
       ID                Branch      Op
1.     im.riot.Riot      stable      r

Proceed with these changes to the user installation? [Y/n]:
$ y
...
Uninstall complete.
```

Kako bismo nadogradili postojeće aplikacije:

```
$ flatpak update --user
Looking for ...updates
       ID                              Branch Op Remote  Download
1.     org.gnome.Platform.Locale       3.38   u  flathub <  326.2MB (partial)
2.     org.gnome.Platform              3.38   u  flathub <  345.7MB

Proceed with these changes to the user installation? [Y/n]:
$ y
...
Updates complete.
```

4.2.2. Snap

Prema Canonical Ltd. [9], snap su aplikacijski paketi koji su jednostavni za instalirati i rade na na svim distribucijama bez potrebe ručnog menadžmenta biblioteka i slično jer sve dolazi upakirano.

Snap je menadžer paketa sa sličnim idejama kao flatpak, no provodi ih na drugačiji način. Za razliku of flatpak-a, snap nije uistinu univerzalan jer radi samo na distribucijama koje koriste systemd kao init sustav. Dodatno, snap automatski vrši nadogradnje paketa u pozadini bez korisnikove intervencije. U nekim slučajevima, ovo je pozitivna stvar, no nije tipičan / očekivan način rada menadžera paketa u GNU/Linux sferi. Snap nije dostupan na Void distribuciji (Void koristi runit umjesto systemd) pa su stoga sljedeće naredbe bile izvršene unutar Manjaro virtualne mašine koristeći *virt-manager*.

Tablica 3: Osnovno korištenje Snap menadžera paketa

	Snap
Instalacija	snap install
Nadogradnja	snap refresh
Deinstalacija	snap remove
Traženje	snap find

Prije svega, prvo je potrebno unutar kod Manjaro sustava instalirati snap menadžer paketa:

```
# pacman -S snapd
resolving dependencies...
looking for conflicting packages...

Packages (2) apparmor-3.0.1-3  snapd-2.51.3-2

Total Download Size:   11.43 MiB
Total Installed Size:  62.66 MiB

:: Proceed with installation? [Y/n]
...
# y
```

Zatim je potrebno pokrenuti snapd servis:

```
# systemctl enable --now snapd
```

U primjeru će biti demonstrirana instalacija *Pinta* aplikacije za uređivanje slika. Za pretraživanje repozitorija za taj program:

```
$ snap find pinta
Name   Version  Publisher      Notes  Summary
pinta  1.7      james-carroll  -      Pinta: Painting Made Simple
```

Aplikacija je dostupna. Moguće ju je instalirati sa:

```
# snap install pinta
...
pinta 1.7 from James Carroll installed
```

Ako aplikacije više nije porebna, moguće ju je deinstalirati:

```
# snap remove pinta
...
pinta removed
```

Sve instalirane aplikacije mogu se nadograditi koristeći:

```
# snap refresh
All snaps up to date.
```

4.2.3. Appimage

Appimage nije menadžer paketa, već format pakiranja aplikacija u jednu izvršnu datoteku koja sadržava sve potrebne biblioteke i datoteke koje program treba kako bi funkcionirao. Jednom izgrađen, takav paket može se koristiti na bilo kojoj distribuciji bez potrebe ikakvih dodatnih instalacija. [10]

Glavna prednost appimage paketa je da nije potrebno ništa instalirati. Dovoljno je samo preuzeti paket i pokrenuti ga. Trenutni najpoznatiji repozitorij je *appimage.github.io* [8], no generalno se appimage paketi preuzimaju izravno sa stranice same aplikacije.

Jedino što je potrebno nakon preuzimanja paketa je označiti ga kao izvršnog:

```
$ chmod +x program.AppImage
```

Zatim se može pokrenuti dvostrukim klikom unutar grafičkog sučelja ili preko naredbenog retka:

```
$ ./program.Appimage
```

Appimage paketi generalno se trebaju izbjegavati jer često stvaraju više problema nego ih rješavaju. Naime, zbog toga jer ne postoji centralni repozitorij, jednostavna nadogradnja generalno nije moguće (osim ako je sustav nadogradnje ugrađen u sam paket), već je potrebno ručno preuzet novu verziju paketa. Kad bi korisnik imao deset takvih aplikacija, on bi morao pratiti njihove verzije i samostalno svaku od tih aplikacija nadograđivati. Postoji i problem sigurnosti zbog manjka centraliziranog repozitorija koji provodi kontrolu kvalitete paketa. Sam format ne uključuje nikakvu izolaciju programa od samog sustava kao što je to slučaj kod flatpak menadžera paketa. U suštini, appimage nosi sve loše osobine flatpak paketa, a ne donosi ni jednu od dobrih. Dodatno, kod appimage paketa ne postoji nikakva integracija sa sustavom (korisnik mora pokrenut sam flatpak paket izravno, no moguće ga je ručno postaviti u druge direktorije kako bi bio dostupan preko pokretača programa grafičkog sučelja i slično). Appimage je najkorisniji u situaciji kad je neki program potreban baš u tom trenutku, a isti ne postoji u repozitoriju korisnikove distribucije, a korisnik nema instaliran flatpak.

4.3. Neovisni o operacijskom sustavu

4.3.1. Docker

Docker je menadžer paketa koja omogućava programerima univerzalni format pakiranja programa koje onda mogu distribuirati na svim operacijskim sustavima (ne samo GNU/Linux). Docker ima neke sličnosti s virtualnim mašinama kao što je npr. *VirtualBox*, no razlika je u tome što ne virtualizira sam kernel odnosno jezgru sustava što dovodi do značajno boljih performansa te manjeg zauzimanja prostora i memorije. Za krajnjeg korisnika, docker ima neke sličnosti s flatpak menadžerom paketa definiranog u sekciji 4.2.1, ali on je primarno namijenjen za korištenje na poslužiteljima, dok je flatpak namijenjen za korištenje na osobnim računalima korisnika.

Docker paketi izgrađuju se iz *dockerfile* datoteke koja je zapravo skripta u kojoj se definira niz naredba uz pomoć kojih se izgrađuje sam paket. Svaki docker paket ima definiran temeljni paket koji je zapravo distribucija na kojoj je baziran. Najčešće, to su Alpine ili Debian distribucije, ali dostupno je i puno drugih. Jednom kad programer odabere baznu distribuciju, on u *dockerfile-u* definira niz postupka za preuzimanje i izgradnju paket koji opet ovisi o odabranoj distribuciji.

Bez dodatne konfiguracije, pokrenut paket odnosno docker kontejner je izoliran od sustava i njemu nisu dostupne vanjske datoteke ili ostala sučelja sustava. Centralni repozitorij docker paketa je Docker Hub [11], no moguće je dodavanje vlastitih repozitorija.

Da bismo koristili docker u Void Linux distribuciji, potrebno ga je prvo instalirati:

Tablica 4: Osnovno korištenje Docker menadžera paketa

	Docker
Preuzimanje	docker pull
Prvo pokretanje	docker run
Uklanjanje kontejnera	docker rm
Uklanjanje paketa kontejnera	docker rmi
Traženje	docker search
Popis preuzetih kontejnera	docker images
Popis pokrenutih kontejnera	docker ps
Zaustavljanje / pokretanje / restart	docker stop / docker start / docker restart

```
# xbps-install -S docker
Name       Action    Version           New version            Download size
docker-cli install   -                 20.10.8_1              21MB
containerd install   -                 1.5.2_1                47MB
moby       install   -                 20.10.6_1              -
tini       install   -                 0.19.0_1               -
docker     install   -                 20.10.8_1              540B

Size to download:               68MB
Size required on disk:         269MB

Do you want to continue? [Y/n]
# y
[*] Downloading packages
...
[*] Verifying package integrity
...
[*] Collecting package files
...
[*] Unpacking packages
...
[*] Configuring unpacked packages
...
3 downloaded, 5 installed, 0 updated, 5 configured, 0 removed.
```

Kod Void distribucije (u odnosu na distribucije koje koriste systemd kao init sustav), servisi novo instaliranih paketa nisu automatski pokrenuti. Da bi omogućili docker servis, potrebno je stvoriti simboličnu poveznicu:

```
# ln -s /etc/sv/docker /var/service/
```

Sada je docker instaliran i omogućen te će biti automatski pokrenut kod svakog pokretanja operacijskog sustava. U primjeru će biti demostrirana instalacija Alpine distribucije. Prvo pretražujemo repozitorij za dostupnost Alpine distribucije (ispis naredbe skraćen):

```
$ docker search alpine
NAME                DESCRIPTION                                     STARS   OFFICIAL
alpine              ... image based on Alpine Linux                 7787    [OK]
mhart/alpine-node   Minimal Node.js built on Alpine Linux           483
```

Svatko može prenijeti vlastiti docker paket na Docker Hub. To može predstavljati sigurnosni problem pa stoga Docker Hub službenim i pregledanim paketima dodaje *OFFICIAL* zastavicu. Da bismo preuzeli paket:

```
# docker pull alpine
Using default tag: latest
latest: Pulling from library/alpine
29291e31a76a: Pull complete
Digest: sha256:eb3e4e175ba6d212ba1d6e04fc0782916c08e1c9d7b45892e9796141b1d379ae
Status: Downloaded newer image for alpine:latest
docker.io/library/alpine:latest
```

Pokretanje s najčešće korištenim opcijama:

```
# docker run -d --name Alpine              \
                --restart=unless-stopped   \
                -v /mnt:/mnt               \
                -p 3000:3000               \
                alpine
8e3a13a78b84c18f739e78a9c6cc7ff0dc4120605a74adbaaefdc154d66ccf2c
```

Kratki opis zastavica:

- **-d** - nastavi izvršavanje kontejnera u pozadini
- **–name** - definiraj ime kontejnera za buduće korištenje u naredbama
- **–restart** - definirati kada pokrenuti kontejner (*unless-stopped* pokreće kontejner svaki put kad je docker servis pokrenut osim ako je taj kontejner prije bio zaustavljen)
- **-v** - montiraj odabrani direktorij sustava unutar odabranog direktorija kontejnera
- **-p** - objavi (*eng. publish*) port kontejnera tj. učini ga dostupnim sustavu

Trenutno aktivne kontejnere provjeravamo s *docker ps* naredbom.

```
# docker ps
CONTAINER ID   IMAGE   COMMAND   CREATED  STATUS      PORTS         NAMES
c24a09af0721   alpine "/bin/sh"  4s ago   Up 3s    3000->3000/tcp   Alpine
```

U ovom primjeru, kontejner čini port 3000 dostupan sustavu iako se ne koristi za ništa. Ovo je korisno kad npr. imamo Apache web poslužitelj unutar kontejnera kojeg možemo učiniti dostupnim na određenom portu. Ako ne specificiramo *-p* zastavicu, vanjski servisi neće se moći spajati na taj kontejner, ali će se kontejner i dalje moći spajati na internet. Docker kontejner zaustavljamo/pokrećemo/restartamo s:

```
# docker stop alpine
alpine
# docker start alpine
alpine
# docker restart alpine
alpine
```

Jednom zaustavljen, kontejner se može ukloniti:

```
# docker rm Alpine
```

Ako kontejner programa više nije potreban, može se ukloniti:

```
# docker rmi alpine
```

Sama nadogradnja kontejnera vrši se njegovim zaustavljanjem, uklanjanjem, preuzimanjem novog paketa kontejnera i ponovnim pokretanjem s istim opcijama. Moguće je ukloniti neke od navedenih postupka korištenjem *docker compose* datoteke gdje definiramo sve opcije, no to je van sklopa osnovnog korištenja opisanog u ovoj sekciji.

5. XBPS

Prema Void Linux kontributori [12], XBPS (eng. *X Binary Package System*) je brz menadžer paketa dizajniran za Void Linux distribuciju. U odnosu na druge menadžere pakete, XBPS nije jedan program, već kolekcija od više manjih alata za menadžment paketa.

5.1. Korištenje

U sljedećim sekcijama biti će objašnjeni najčešće korišteni XBPS alati za menadžment paketa u Void Linux distribuciji.

5.1.1. xbps-query

Sve radnje pretraživanja repozitorija ili informacija o lokalno instaliranim paketima provode se koristeći *xbps-query* program. *Xbps-query* može se koristiti nad udaljenim repozitorijem (koristi se *-R* zastavica) ili lokalno.

Tablica 5: Xbps-query

	Udaljeni repozitorij	Lokalno
Pretraživanje paketa	xbps-query -Rs	xbps-query -s
Informacije paketa	xbps-query -RS	xbps-query -S
Popis datoteka paketa	xbps-query -Rf	xbps-query -f
Popis paketa o kojima paket ovisi	xbps-query -Rx	xbps-query -x
Popis paketa koji ovise o paketu	xbps-query -RX	xbps-query -X
Popis instaliranih paketa	/	xbps-query -l
Popis explicitno instaliranih paketa	/	xbps-query -m
Popis paketa o kojima više ni jedan paket ne ovisi	/	xbps-query -O
Popis trenutnih repozitorija	/	xbps-query -L

Za primjer, možemo pretražiti repozitorije za *wget* program. Ispis naredbe zvjezdicom označava već instalirane programe, a ostale s -.

```
$ xbps-query -Rs wget
[-] pwget-2.0_1      Single-binary stateless password manager
[*] wget-1.21.1_2    GNU wget download utility
[-] wgetpaste-2.30_1 Script that automates pasting to a number of pastebin
    services
[-] wpull-2.0.3_3    Wget-compatible web downloader and crawler
```

Pakete o kojima ovisi *wget* možemo pregledati s (nije potrebno koristiti *-R* zastavicu jer je *wget* već instaliran):

```
$ xbps-query -x wget
ca-certificates>=0
libpcre2>=10.22_1
libuuid>=2.18_1
libidn2>=2.1.1_1
libssl1.1>=1.1.1f_1
libcrypto1.1>=1.1.1f_1
zlib>=1.2.3_1
glibc>=2.32_1
```

Lokalni paketi koji ovise o *wget* programu (nema ih trenutno):

```
$ xbps-query -X wget
```

Popis datoteka koje donosi *wget* paket (opet, *-R* nije potreban jer je *wget* već instaliran):

```
$ xbps-query -f wget
/etc/wgetrc
/usr/bin/wget
/usr/share/info/wget.info
/usr/share/locale/af/LC_MESSAGES/wget-gnulib.mo
...
/usr/share/locale/zh_TW/LC_MESSAGES/wget.mo
/usr/share/man/man1/wget.1
```

5.1.2. xbps-install

Sve radnje instalacije i nadogradnje paketa vrše se korištenjem *xbps-install* programa. Naredbe u tablici prikazane su s odvojenim opcijama, no često se kombiniraju (npr. za sinkronizaciju indeksa repozitorija i nadogradnju svih paketa, pišemo *xbps-install -Su*). *Instalacija paketa s dodanih repozitorija* odnosi se na repozitorije trenutno uključene u sustavu.

Tablica 6: Xbps-install

	Lokalno
Sinkroniziranje indeksa repozitorija	xbps-install -S
Nadogradnja paketa	xbps-install -u
Instalacija paketa s dodanih repozitorija	xbps-install
Instalacija paketa s odabranog repozitorija	xbps-install -R url
Instalacija paketa na drugo mjesto	xbps-install -r dir

Svaka instalacija Void distribucije dolazi s glavnim repozitorijem *alpha.de.repo.voidlinux.org* koji je lociran u Finskoj, ali postoje i dodatni repozitoriji u drugim državama. Glavni repozitorij uključuje samo pakete otvorenog koda. [13]

Dodatni repozitorij koji se mogu uključiti:

- nonfree - paketi zatvorenog koda
- multilib - 32-bitne biblioteke za 64-bitne sustave
- multilib/nonfree - 32-bitni paketi zatvorenog koda
- debug - simboli paketa za uklanjanje greška

Uključivanje repozitorija provodi se isto kao i instalacija novih paketa. Za dodavanje *nonfree* repozitorija:

```
# xbps-install -S void-repo-nonfree
1 package will be installed:
void-repo-nonfree-9_5

Size required on disk:           63B
...
Do you want to continue? [Y/n]
# y
[*] Verifying package integrity
void-repo-nonfree-9_5: verifying RSA signature...
[*] Collecting package files
void-repo-nonfree-9_5: collecting files...
[*] Unpacking packages
void-repo-nonfree-9_5: unpacking ...
[*] Configuring unpacked packages
void-repo-nonfree-9_5: configuring ...
void-repo-nonfree-9_5: installed successfully.

1 downloaded, 1 installed, 0 updated, 1 configured, 0 removed.
```

Sad kad je *nonfree* repozitorij dodan, možemo instalirati Skype (program zatvorenog koda dostupan samo preko toga repozitorija):

```
# xbps-install skype
Name  Action    Version           New version            Download size
skype install   -                 8.75.0.140_1           105MB
...
Do you want to continue? [Y/n]
# y
[*] Downloading packages
skype-8.75.0.140_1.x86_64.xbps.sig: 512B [avg rate: 16MB/s]
skype-8.75.0.140_1.x86_64.xbps: 105MB [avg rate: 2309KB/s]
skype-8.75.0.140_1: verifying RSA signature...
[*] Collecting package files
skype-8.75.0.140_1: collecting files...
[*] Unpacking packages
skype-8.75.0.140_1: unpacking ...
[*] Configuring unpacked packages
skype-8.75.0.140_1: configuring ...
Updating GTK+ icon cache for /usr/share/icons/hicolor...
```

```
Updating MIME database...
skype-8.75.0.140_1: installed successfully.

1 downloaded, 1 installed, 0 updated, 1 configured, 0 removed.
```

5.1.3. xbps-remove

Sve radnje koje uključuju uklanjanje instaliranih paketa ili čišćenje predmemorije vrše se koristeći *xbps-remove*.

Tablica 7: Xbps-remove

	Lokalno
Uklanjanje paketa	xbps-remove
Uklanjanje paketa i njemu potrebnih paketa	xbps-remove -R
Brisanje preuzetih paketa starije verzije	xbps-remove -O
Brisanje paketa koji su nekad bili potrebni drugim paketi	xbps-remove -o
Uklanjanje paketa sa drugog mjesta	xbps-remove -r

Da bismo uklonili *Skype* instaliran u sekciji 5.1.2, pišemo:

```
# xbps-remove -R skype
Name  Action    Version           New version            Download size
skype remove    8.75.0.140_1      -                      -

Size freed on disk:            297MB
...
Do you want to continue? [Y/n]
# y
Removing 'skype-8.75.0.140_1' ...
Updating GTK+ icon cache for /usr/share/icons/hicolor...
Updating MIME database...
Removed 'skype-8.75.0.140_1' successfully.

0 downloaded, 0 installed, 0 updated, 0 configured, 1 removed.
```

Dodajemo *-R* zastavicu kako bi uklonili i sve pakete koji su potrebni Skype-u, a nisu potrebni ni jednom drugom paketu. U ovom slučaju, Skype nema dodatnih paketa o kojima ovisi.

5.1.4. xbps-pkgdb

Radnje koje uključuju rad s XBPS bazom paketa provode se koristeći *xbps-pkgdb* alat. Primarno se koristi za onemogućavanje nadogradnje nekog paketa. Dodatno, sve operacije mogu se izvršiti nad drugim korijenskim datotečnim sustavom koristeći *-r* zastavicu.

Ako korisnik želi ostati na trenutnoj verziji Linux kernela:

Tablica 8: Xbps-pkgdb

	Lokalno
Onemogući nadogradnju paketa	xbps-pkgdb -m hold
Omogući nadogradnju paketa	xbps-pkgdb -m unhold
Omogući nadogradnju paketa samo iz inicijalnog repozitorija	xbps-pkgdb -m repolock
Omogući nadogradnju paketa iz bilo kojeg repozitorija	xbps-pkgdb -m repounlock
Označi paket kao automatski instaliran	xbps-pkgdb -m auto
Označi paket kao eksplicitno instaliran	xbps-pkgdb -m manual

```
# xbps-pkgdb -m hold linux
```

Kad korisnik ponovno želi omogućiti nadogradnje:

```
# xbps-pkgdb -m unhold linux
```

5.1.5. xtools

Prema Void Linux kontributori [14], *xtools* je kolekcija pomoćnih alata za korištenje uz XBPS menadžer paketa. Trenutno xtools paket sadrži 41 alat:

```
$ xbps-query -Rf xtools | grep bin/ | wc -l
41
```

Xtools alati se generalno koriste kod izgradnje novih paketa. U sljedećim sekcijama biti će demonstrirano korištenje alata za općenito korištenje. Prije svega, potrebno je instalirati *xtools*:

```
# xbps-install -S xtools
...
```

5.1.5.1. xlocate

Xlocate je namijenjen za identificiranje paketa kojem pripada neka datoteka. Primarno se koristi kod situacije kad se koristi neka naredba, a ne zna se kojem paketu ona pripada. Npr. program *pdfunite* za spajanje pdf dokumenata prisutan je na mnogim distribucijama, no ne postoji sam *pdfunite* paket. Kako bi otkrili kojem paketu pripada taj program, prvo moramo sinkronizirati xlocate bazu tih informacija:

```
$ xlocate -S
```

Zatim možemo pretraživati paket po imenu datoteke ili putanje koju bi on trebao sadržavati. Kao što je objašnjeno u sekciji 3.1, Void Linux sve izvršne datoteke pohranjuje u direktoriju `/usr/bin`. Nije potrebno pretraživati po cijeloj putanji, već je dovoljan samo dio. Ovako se može pretraživati za *pdfunite* izvršnu datoteku:

```
$ xlocate bin/pdfunite
poppler-21.07.0_1     /usr/bin/pdfunite
```

Ispis naredbe u prvom stupcu prikazuje paket kojem ta izvršna datoteka pripada, a u drugom punu putanju. Ako želimo saznati koje ostale programe nudi paket *poppler*, možemo koristiti *xbps-query* (objašnjeno u sekciji 5.1.1):

```
$ xbps-query -f poppler | grep bin/
/usr/bin/pdfattach
/usr/bin/pdfdetach
/usr/bin/pdffonts
/usr/bin/pdfimages
/usr/bin/pdfinfo
/usr/bin/pdfseparate
/usr/bin/pdftocairo
/usr/bin/pdftohtml
/usr/bin/pdftoppm
/usr/bin/pdftops
/usr/bin/pdftotext
/usr/bin/pdfunite
```

5.1.5.2. xdowngrade

Ovaj alat koristi se za vraćanje paketa na raniju verziju. Void ne održava starije verzije paketa u repozitorijima pa stoga ovaj alat radi samo ako korisnik ima raniju verziju programa u predmemoriji. *Xdowngrade* se koristi ukoliko dođe do situacije u kojoj nova verzija programa donosi neke probleme ili ne radi.

Za primjer, pokušajmo nadograditi sustav:

```
# xbps-install -Su
[*] Updating repository `https://alpha.de.repo.voidlinux.org/current/x86_64-
    repodata' ...
Name           Action    Version           New version            Download size
xfsprogs       update    5.12.0_1          5.13.0_1               1169KB
...
Do you want to continue? [Y/n]
# y
...
```

Xfsprogs paket sadrži alate za rad s *XFS* datotečnim sustavom. Recimo da nakom ove nadogradnje pokušavamo stvoriti *XFS* datotečni sustav na novom disku, no ne možemo zbog neke greške. U takvoj situaciji, možemo se vratiti na prošlu verziju paketa gdje taj problem nije postojao. Prvo moramo provjeriti koja verzija je dostupna u XBPS predmemoriji:

```
$ find /var/cache/xbps/ -iname "*xfsprogs*"
/var/cache/xbps/xfsprogs-5.13.0_1.x86_64.xbps
/var/cache/xbps/xfsprogs-5.12.0_1.x86_64.xbps.sig
/var/cache/xbps/xfsprogs-5.12.0_1.x86_64.xbps
/var/cache/xbps/xfsprogs-5.13.0_1.x86_64.xbps.sig
```

Prijašnja dostupna verzija je 5.12.0. Kako bi se vratili na tu verziju, koristimo:

```
# xdowngrade /var/cache/xbps/xfsprogs-5.12.0_1.x86_64.xbps
index: added 'xfsprogs-5.12.0_1' (x86_64).
index: 1 packages registered.

Name     Action    Version           New version            Download size
xfsprogs downgrade 5.13.0_1          5.12.0_1               -

...
Do you want to continue? [Y/n]
# y
[*] Verifying package integrity
xfsprogs-5.12.0_1: verifying SHA256 hash...
[*] Collecting package files
xfsprogs-5.13.0_1: collecting files...
[*] Unpacking packages
xfsprogs-5.12.0_1: unpacking ...
[*] Configuring unpacked packages
xfsprogs-5.12.0_1: configuring ...
xfsprogs-5.12.0_1: installed successfully.

0 downloaded, 1 installed, 0 updated, 1 configured, 0 removed.
```

Sada je paket vraćen na prošlu verziju, međutim, postoji problem. Ako sada pokušavamo opet nadograditi sustav, ponuđena je ponovna nadogradnja tog paketa:

```
# xbps-install -Su
[*] Updating repository 'https://alpha.de.repo.voidlinux.org/current/x86_64-
    repodata' ...

Name     Action    Version           New version            Download size
xfsprogs update    5.12.0_1          5.13.0_1               -
```

Kako bismo ignorirali buduće nadogradnje ovoga paketa, koristimo *xbps-pkgdb* alat opisan u sekciji 5.1.4.

```
# xbps-pkgdb -m hold xfsprogs
```

Ako se sada pokuša nadograditi sustav, paket neće biti ponuđen za nadogradnju.

```
# xbps-install -Su
[*] Updating repository 'https://alpha.de.repo.voidlinux.org/current/x86_64-
    repodata'
```

Kad korisnik želi ponovno preuzimati nadogradnje za *xfsprogs* paket:

```
# xbps-pkgdb -m unhold xfsprogs
# xbps-install -Su
[*] Updating repository 'https://alpha.de.repo.voidlinux.org/current/x86_64-
    repodata'

Name     Action    Version           New version            Download size
xfsprogs update    5.12.0_1          5.13.0_1               -
...
```

5.2. Xbps-src

Pakiranje svih paketa i instalacija paketa čija distribucija inače nije dozvoljena licencom provodi se korištenjem *xbps-src* alata. *Xbps-src*, zajedno sa predlošcima svih paketa u Void distribuciji, nalazi se unutar *void-packages* Github repozitorija. Općenito, *xbps-src* se može koristiti na bilo kojoj distribuciji jer sve radnje provodi unutar virtualnog *chroot* okruženja. [15].

5.2.1. Postavljanje

Da bi koristili *xbps-src*, potrebno je klonirati pripadni git repozitorij i inicijalizirati alat:

```
$ git clone https://github.com/void-linux/void-packages.git
Cloning into "void-packages"...
remote: Enumerating objects: 855447, done.
...
$ cd void-packages
$ ./xbps-src binary-bootstrap
=> xbps-src: installing base-chroot...
=> xbps-src: updating repositories for host (x86_64)...
[*] Updating repository 'https://alpha.de.repo.voidlinux.org/current/x86_64-
    repodata' ...
...
66 packages will be installed:
kernel-libc-headers-5.10.4_1 glibc-2.32_2 glibc-devel-2.32_2
...
66 downloaded, 66 installed, 0 updated, 66 configured, 0 removed.
=> xbps-src: installed base-chroot successfully!
=> xbps-src: reconfiguring base-chroot...
glibc-locales: configuring ...
...
=> xbps-src: removing autodeps, please wait...
=> xbps-src: cleaning up masterdir...
```

Sada se može koristiti xbps-src. Da bi se ažurirao repozitorij, koristimo (unutar void-packages direktorija):

```
$ git pull
remote: Enumerating objects: 17, done.
...
 srcpkgs/linux4.19/template                   | 4 ++--
 srcpkgs/linux5.4/template                    | 4 ++--
 srcpkgs/wine/patches/musl-ns_name_skip.patch | 4 ++--
 srcpkgs/wine/template                        | 6 +++---
 4 files changed, 9 insertions(+), 9 deletions(-)
```

Void-packages ima dobro definiranu hijerarhiju datoteka, a ovom radu relevantne su sljedeće:

- **etc** - konfiguracijske datoteke za xbps-src
- **srcpkgs** - sadržava predloške svih paketa
- **masterdir**/**builddir** - ovdje xbps-src izgrađuje pakete
- **hostdir**/**binpkgs** - mjesto gdje se pohranjuju izgrađeni paketi

```
/void-packages
   |- common
   |- etc
   |- srcpkgs
   |  |- xbps
   |     |- template
   |
   |- hostdir
   |  |- binpkgs ...
   |  |- ccache ...
   |  |- distcc-<arch> ...
   |  |- repocache ...
   |  |- sources ...
   |
   |- masterdir
   |  |- builddir -> ...
   |  |- destdir -> ...
   |  |- host -> bind mounted from <hostdir>
   |  |- void-packages -> bind mounted from <void-packages>
```

Slika 1: Hijerarhija direktorija void-packages repozitorija (Izvor: Void Linux kontributori, 2021)

5.2.2. Izgradnja paketa

Moguće je izgraditi paket ako za njega postoji validan predložak u *srcpkgs*. U primjeru će biti demostrirana izgradnja *Discord* paketa. Prvo je potrebno provjeriti da li postoji predložak za Discord:

```
$ find srcpkgs -iname "*discord*"
srcpkgs/bitlbee-discord
srcpkgs/discord-ptb
srcpkgs/discord
```

Predložak za Discord postoji pa ga korisnik može pokušati izgraditi:

```
$ ./xbps-src pkg discord
=> ERROR: discord-0.0.15_2: does not allow redistribution of sources/binaries (
    restricted license).
=> ERROR: If you really need this software, run 'echo XBPS_ALLOW_RESTRICTED=yes >>
     etc/conf'
```

Dolazi do greške jer, isto kao što Void ne dolazi s uključenim repozitorijima za pakete zatvorenog koda, tako i xbps-src ne dolazi s omogućenom izgradnjom paketa zatvorenog koda. Prema Void Linux kontributori [16], izgradnju takvih paketa moguće je omogućiti s:

```
$ echo "XBPS_ALLOW_RESTRICTED=yes" >> etc/conf
```

Sada se paket može izgraditi.

```
$ ./xbps-src pkg discord
=> xbps-src: updating repositories for host (x86_64)...
[*] Updating repository 'https://alpha.de.repo.voidlinux.org/current/x86_64-
    repodata' ...
...
=> xbps-src: updating software in / masterdir...
=> xbps-src: cleaning up / masterdir...
=> discord-0.0.15_2: removing autodeps, please wait...
=> discord-0.0.15_2: building for x86_64...
=> discord-0.0.15_2: running do-fetch hook: 00-distfiles ...
=> discord-0.0.15_2: fetching distfile 'discord-0.0.15.tar.gz'...
...
=> discord-0.0.15_2: verifying checksum for distfile 'discord-0.0.15.tar.gz'... OK
    .
...
=> discord-0.0.15_2: running do-pkg hook: 00-gen-pkg ...
=> Creating discord-0.0.15_2.x86_64.xbps for repository /host/binpkgs/nonfree ...
=> discord-0.0.15_2: running post-pkg hook: 00-register-pkg ...
=> Registering new packages to /host/binpkgs/nonfree
...
index: added 'discord-0.0.15_2' (x86_64).
index: 1 packages registered.
=> discord-0.0.15_2: removing autodeps, please wait...
=> discord-0.0.15_2: cleaning build directory...
=> discord: removing files from destdir...
```

Jednom izgrađen, paket se pohranjuje u lokalni repozitorij u `srcpkgs/binpkgs` direktoriju. Da bi se novo izgrađen paket instalirao, potrebno je koristiti *xbps-install* iz lokalnog repozitorija (objašnjeno u sekciji 5.1.2):

```
# xbps-install -R hostdir/binpkgs/nonfree discord

Name    Action    Version           New version            Download size
libcxx  install   -                 12.0.0_3               386KB
discord install   -                 0.0.15_2               -
...
Do you want to continue? [Y/n]
# y
[*] Downloading packages
libcxx-12.0.0_3.x86_64.xbps.sig: 512B [avg rate: 17MB/s]
...
1 downloaded, 2 installed, 0 updated, 2 configured, 0 removed.
```

Ovako instaliran paket mora se ručno nadograđivati periodičnim kloniranjem git repozitorija, ponovnom izgradnjom i instalacijom. Dodatno, ako korisnik izgrađuje paket koji već postoji u repozitoriju, a čije izmjene ne želi da budu uklonjene prilikom nadogradnje iz primarnog repozitorija, može ograničiti nadogradnju paketa samo na repozitorij iz kojeg je prvo bio instaliran (objašnjeno u sekciji 5.1.4):

```
# xbps-pkgdb -m repolock discord
```

5.2.3. Stvaranje novih paketa

Za izgradnju novih paketa potrebno je stvoriti predložak. Moguće je ručno stvaranje predloška, no preporuča se korištenje *xnew* alata koji je dio *xtools* paketa. Svaki predložak sastoji se od varijabla i funkcija koje se dodavaju / nadjačavaju po potrebi.

Osnovne varijable [17]:

- pkgname - ime paketa
- version - verzija
- revision - broj izmjene paketa
- build_style - predefinirani načini izgradnje paketa
- hostmakedepends - alati potrebni za izgradnju paketa unutar virtualnog okruženja
- makedepends - biblioteke potrebne za kompajliranje programa unutar virtualnog okruženja
- depends - paketi/biblioteke potrebne za rad programa na korisnikovom sustavu
- short_description - kratki opis paketa do 72 znakova
- maintainer - ime i e-mail stvaratelja paketa
- license - licenca programa
- homepage - službena stranica programa

- distfiles - arhivirani izvorni kod programa
- checksum - hash arhiviranog izvornog koda programa

Funkcije izgradnje paketa [17]:

- pre_fetch() - radnje prije preuzimanja izvornog koda
- do_fetch() - preuzimanje izvornog koda
- post_fetch() - radnje nakon preuzimanja izvornog koda
- pre_extract() - radnje prije ekstrakcije izvornog koda
- do_extract() - ekstrakcija izvornog koda
- post_extract() - radnje nakon ekstrakcije izvornog koda
- pre_patch() - radnje prije dodavanja zakrpa izvornom kodu
- do_patch() - dodavanje zakrpa izvornom kodu
- post_patch() - radnje nakon dodavanja zakrpa izvornom kodu
- pre_configure() - radnje prije konfiguracije kompajlerskih opcija
- do_configure() - konfiguracija kompajlerskih opcija
- post_configure() - radnje nakon definiranja kompajlerskih opcija
- pre_build() - radnje prije kompajliranja programa
- do_build() - kompajliranje programa
- post_build() - radnje nakon kompajliranja programa
- pre_install() - radnje prije instalacije paketa
- do_install() - instalacija paketa
- post_install() - radnje nakon instalacije paketa
- do_clean() - čišćenje virtualnog okruženja

Za izgradnju manjih programa često nije potrebno definirati ni jednu od funkcija, već je dovoljno definirati samo *build_style* nakon čega xbps-src provodi niz predefiniranih radnja za izgradnju paketa određenog tipa. Neki od najčešće korištenih načina izgradnje su gnu-configure, cargo, cmake, configure, fetch, go, python-module itd.

Globalne funkcije [17]

- vinstall() - instaliraj datoteku u direktorij

- vcopy() - rekurzivno kopiraj specificirane datoteke u direktorij
- vmove() - premjesti specificirane datoteke u direktorij
- vmkdir() - stvori direktorij u direktoriju
- vbin() - instaliraj izvršnu datoteku u `/usr/bin`
- vman() - instaliraj priručnik programa u `/usr/share/man`
- vdoc() - instaliraj dokumentaciju programa u `/usr/share/doc`
- vconf() - instaliraj konfiguracijsku datoteku programa u `/etc`
- vsconf() - instaliraj primjere konfiguracije programa u `/usr/share/examples`
- vlicense() - instaliraj licence programa u `/usr/share/licenses`
- vsv() - instaliraj servis programa u `/etc/sv`
- vsed() - izmijeni tekst datoteke
- vcompletion() - instaliraj prijedloge naredba za terminal

Globalne funkcije potrebno je koristiti za određene datoteke koje (ovisno o načinu izgradnje) ne budu nužno automatski instalirane. Često se koriste kod izgradnje paketa programa zatvorenog koda budući da se nema što kompajlirati, već se samo kopiraju izgrađene datoteke u pripadne direktorije.

5.2.3.1. Siege

Za demonstraciju izgradnje paketa programa otvorenog koda, pakirati će se *Siege* alat za testiranje performansa web poslužitelja koji trenutno nije pakiran u Void repozitorijima. Za početak, potrebno je nalaziti se unutar *void-packages* direktorija. Kako bi se korisnik mogao lako vratiti na prvotno stanje repozitorija, predlaže se stvaranje vlastite grane (eng. *branch*) git repozitorija i prebacivanje na istu:

```
$ git branch foi
$ git switch foi
```

Sada je potrebno stvoriti novi predložak. Pozivamo *xnew* i unosimo ime paketa kao argument:

```
$ xnew siege
```

Automatski se otvara novo kreirani predložak u korisnikovom editoru teksta sa sljedećim sadržajem:

```
# Template file for 'siege'
pkgname=siege
version=
revision=1
#archs="i686 x86_64"
#wrksrc=
#create_wrksrc=yes
build_style=gnu-configure
#configure_args=""
#make_build_args=""
#make_install_args=""
#conf_files=""
#make_dirs="/var/log/dir 0755 root root"
hostmakedepends=""
makedepends=""
depends=""
short_desc=""
maintainer=" <>"
license="GPL-3.0-or-later"
homepage=""
distfiles=""
checksum=badbadbadbadbadbadbadbadbadbadbadbadbadbadbadbadbadbadbadbadb
```

Za ovaj paket nam nisu potrebne dodatne ponuđene konfiguracijske opcije koje su komentirane (počinju s # znakom) pa ih možemo ukloniti. Sada možemo unijeti svoje ime i e-mail pod *maintainer* i kratki opis paketa pod *short_desc*. Nakon toga potrebno je definirati gdje se nalazi izvorni kod i koju verziju programa pakiramo te ispuniti *homepage* varijablu službenom stranicom programa kojeg pakiramo (ako program nema službenu stranicu, koristi se adresa izvornog koda). Sav izvorni kod Siege alata nalazi se na `https://github.com/JoeDog/siege`. Pod *releases* vidimo da je najnovija verzija v4.1.1. Xbps-src očekuje samo brojeve s točkama za *version* varijablu pa se stoga unaša samo 4.1.1 za verziju. Pod *distfiles* treba unijeti url same arhive programskog koda. Ovdje je to `https://github.com/JoeDog/siege/archive/refs/tags/v4.1.1.tar.gz`. Zamijenjujemo samu verziju s varijablom za lakše održavanje paketa. Još je preostala licenca koju možemo očitati s Github repozitorija.

```
# Template file for 'siege'
pkgname=siege
version=4.1.1
revision=1
build_style=gnu-configure
hostmakedepends=""
makedepends=""
depends=""
short_desc="Siege is an open source regression test and benchmark utility."
maintainer="Goran Pticek <gpticek@foi.hr>"
license="GPL-3.0-or-later"
homepage="https://www.joedog.org"
distfiles="https://github.com/JoeDog/siege/archive/refs/tags/v${version}.tar.gz"
checksum=badbadbadbadbadbadbadbadbadbadbadbadbadbadbadbadbadbadbadbadb
```

Sada je potrebno izračunati hash arhive izvornog koda. To se opet može ručno, ali preporuča

se korištenje *xgensum* alata iz *xtools* paketa. Dodajemo *-f -i* opcije koje govore alatu neka preuzme paket, izračuna hash i upiše ga u predložak. Pozivamo alat s nazivom paketa kao argument:

```
$ xgensum -f -i siege
=> xbps-src: updating software in / masterdir...
=> xbps-src: cleaning up / masterdir...
=> siege-4.1.1_1: running do-fetch hook: 00-distfiles ...
=> siege-4.1.1_1: fetching distfile 'v4.1.1.tar.gz'...
v4.1.1.tar.gz: 316KB [avg rate: 86MB/s]
=> siege-4.1.1_1: verifying checksum for distfile 'v4.1.1.tar.gz'... OK.
```

Osnovne informacije su popunjene i preostale su sve ostale za samu izgradnju paketa. Prema JoeDog [18], Siege program se može izgraditi koristeći GNU autoconf. Nadalje objašnjava da kod preuzet s Github repozitorija ne sadrži *configure* datoteku zbog čega se prije kompajliranja treba pokrenuti `utils/bootstrap` skripta za postavljanje potrebnih datoteka, a za njezin rad je potrebno imati *autoconf*, *automake* i *libtool* programe. Kako bi omogućili sve značajke programa, navodi se da je potrebno imati *zlib* i *openssl* programe za rad programa te iste biblioteke prilikom izgradnje programa.

Budući da se koristi GNU autoconf, pod *build_style* možemo unijeti pripadni stil izgradnje (gnu-configure). Da bi koristili priloženu skriptu, moramo imati *autoconf*, *automake* i *libtool* programe dostupne unutar virtualnog okruženja pa iste definiramo unutar *hostmakedepends* varijable:

```
# Template file for 'siege'
pkgname=siege
version=4.1.1
revision=1
build_style=gnu-configure
hostmakedepends="autoconf automake libtool"
makedepends=""
depends=""
short_desc="Siege is an open source regression test and benchmark utility."
maintainer="Goran Pticek <gpticek@foi.hr>"
license="GPL-3.0-or-later"
homepage="https://www.joedog.org"
distfiles="https://github.com/JoeDog/siege/archive/refs/tags/v${version}.tar.gz"
checksum=5ddb03fbff11102f73f08db2d0cfcc161f1aac50f3c9cca3998aac15b3c679d6
```

Za omogućavanje svih funkcionalnosti programa potrebno je imati *openssl* i *zlib* biblioteke dostupne unutar virtualnog okruženja (općenito, sve biblioteke koje se koriste prilikom izgradnje paketa u Void distribuciju imaju *-devel* sufiks). Preostaje depends varijable koja definira pakete koji su potrebni radu ovog programa nakon što je izgrađen. U ovom slučaju, to su *openssl* i *zlib* paketi:

```
# Template file for 'siege'
pkgname=siege
version=4.1.1
```

```
revision=1
build_style=gnu-configure
hostmakedepends="autoconf automake libtool"
makedepends="openssl-devel zlib-devel"
depends="openssl zlib"
short_desc="Siege is an open source regression test and benchmark utility."
maintainer="Goran Pticek <gpticek@foi.hr>"
license="GPL-3.0-or-later"
homepage="https://www.joedog.org"
distfiles="https://github.com/JoeDog/siege/archive/refs/tags/v${version}.tar.gz"
checksum=5ddb03fbff11102f73f08db2d0cfcc161f1aac50f3c9cca3998aac15b3c679d6
```

Još samo preostaje pokretanje `utils/bootstrap` skripte. Nakon što xbps-src preuzme i dekompresira arhivu izvornog koda, automatski nas postavlja u direktorij u kojem je pohranjen kod iz arhive. Budući da je opisano da treba pokrenuti samu skriptu prije konfiguracije kompajlerskih opcija, možemo ju pokrenuti u sklopu *pre_configure* funkcije:

```
# Template file for 'siege'
pkgname=siege
version=4.1.1
revision=1
build_style=gnu-configure
hostmakedepends="autoconf automake libtool"
makedepends="openssl-devel zlib-devel"
depends="openssl zlib"
short_desc="Siege is an open source regression test and benchmark utility."
maintainer="Goran Pticek <gpticek@foi.hr>"
license="GPL-3.0-or-later"
homepage="https://www.joedog.org"
distfiles="https://github.com/JoeDog/siege/archive/refs/tags/v${version}.tar.gz"
checksum=5ddb03fbff11102f73f08db2d0cfcc161f1aac50f3c9cca3998aac15b3c679d6

pre_configure() {
  utils/bootstrap
}
```

Predložak je gotov. Spremamo predložak i možemo izgraditi paket:

```
$ ./xbps-src pkg siege
=> xbps-src: updating repositories for host (x86_64)...
[*] Updating repository `https://alpha.de.repo.voidlinux.org/current/x86_64-
    repodata
...
checking whether to build shared libraries... yes
checking whether to build static libraries... yes
...
Configuration is complete
...
=> siege-4.1.1_1: running pre-build hook: 02-script-wrapper ...
=> siege-4.1.1_1: running do_build ...
```

```
...
make[2]: Leaving directory '/builddir/siege-4.1.1/html'
make[1]: Leaving directory '/builddir/siege-4.1.1/html'
=> siege-4.1.1_1: running post-install hook: 00-compress-info-files ...
=> siege-4.1.1_1: running post-install hook: 00-fixup-gir-path ...
...
=> Registering new packages to /host/binpkgs/foi
index: skipping 'siege-4.1.1_1' (x86_64), already registered.
index: 1 packages registered.
=> siege-4.1.1_1: removing autodeps, please wait...
=> siege-4.1.1_1: cleaning build directory...
=> siege: removing files from destdir...
```

Paket je uspješno izgrađen. Budući da su se izmjene provodile nad *foi* granom void-packages git repozitorija, binarni paket nalazi se u `hostdir/binpkgs/foi/`. Paket sada možemo instalirati:

```
# xbps-install -R hostdir/binpkgs/foi siege
Name  Action    Version           New version            Download size
siege install   -                 4.1.1_1                -
Do you want to continue? [Y/n]
# y
...
0 downloaded, 1 installed, 0 updated, 1 configured, 0 removed.
```

Možemo testirati da li se program pokreće:

```
$ siege
SIEGE 4.1.1
Usage: siege [options]
       siege [options] URL
       siege -g URL
Options:
...
```

5.2.3.2. Jaxx

Za demonstraciju izgradnje paketa programa zatvorenog koda, pakirati će se Jaxx program koji se koristi kao digitalni novčanik kriptovaluta. Nalazimo se se u void-packages direktoriju i stvaramo novi predložak:

```
$ xnew jaxx
```

Zatim popunjujemo osnovna polja kao u sekciji 5.2.3.1:

```
# Template file for 'jaxx'
pkgname=jaxx
version=2.6.4
```

```
revision=1
short_desc="Cross-platform Blockchain Wallet"
maintainer="Goran Pticek <gpticek@foi.hr>"
license="proprietary"
homepage="https://jaxx.io"
distfiles="https://download-liberty.jaxx.io/Jaxx.Liberty-${version}.AppImage"
checksum=8947bbd552677fb07865e9b36a8399d25cc32f26c4952c348a088401d03e9f7c
```

Budući da se ovdje radi o programu zatvorenog koda, same izgradnje programa nema pa se stoga koristi *fetch* za build_style. Budući da Void ne distribuira pakete zatvorenog koda u glavnom repozitoriju, potrebno je navesti korištenje *nonfree* repozitorija. Za samu instalaciju možemo nadjačati *do_install()* funkciju i koristiti *vbin()* funkciju kojoj prosljeđujemo naziv izvršne datoteke iz dekompresirane arhive kao prvi argument, a novo ime kao drugi. Dodatno, potrebno je postaviti *nostrip* varijablu kako bi se izbjegla modifikacija izvršne datoteke od strane xbps-src alata (Void odvaja simbole za uklanjanje greška iz izvršnih datoteka).

```
# Template file for 'jaxx'
pkgname=jaxx
version=2.6.4
revision=1
build_style=fetch
short_desc="Cross-platform Blockchain Wallet"
maintainer="Goran Pticek <gpticek@foi.hr>"
license="proprietary"
homepage="https://jaxx.io"
distfiles="https://download-liberty.jaxx.io/Jaxx.Liberty-${version}.AppImage"
checksum=8947bbd552677fb07865e9b36a8399d25cc32f26c4952c348a088401d03e9f7c
repository=nonfree
nostrip=yes

do_install() {
        vbin Jaxx.Liberty-${version}.AppImage  ${pkgname}
}
```

Izgrađujemo paket:

```
$ ./xbps-src pkg jaxx
=> xbps-src: updating repositories for host (x86_64)...
...
=> Registering new packages to /host/binpkgs/foi/nonfree
...
=> jaxx-2.6.4_1: removing autodeps, please wait...
=> jaxx-2.6.4_1: cleaning build directory...
=> jaxx: removing files from destdir...
```

Sada se može instalirati paket. U sekciji 5.2.3.1, instalacija paketa iz lokalnog repozitorija vršila se koristeći *xbps-install* izravno, no ovdje će se koristiti *xi* alat iz *xtools* paketa koji automatski uzima u obzir lokalni repozitorij:

```
# xi jaxx
...
Name Action    Version           New version            Download size
jaxx install   -                 2.6.4_1                -
...
Do you want to continue? [Y/n]
# y
[*] Verifying package integrity
jaxx-2.6.4_1: verifying SHA256 hash...
...
0 downloaded, 1 installed, 0 updated, 1 configured, 0 removed.
```

Paket je instaliran.

5.3. Distribucija paketa

5.3.1. Postavljanje poslužitelja

Za poslužitelj repozitorija koristit će se Raspberry Pi 3 B+ kojeg Void Linux službeno podržava. Preuzimamo najnoviju dostupnu verziju i instaliramo ju na sd karticu (u ovom slučaju, odgovara `/dev/sde` putanji):

```
$ wget https://alpha.de.repo.voidlinux.org/live/current/void-rpi3-20210218.img.xz
# xzcat void-rpi3-20210218.img.xz | dd of=/dev/sde
```

U sklopu ovog rada, svi detalji konfiguracije nisu objašnjeni, ali su i dalje većinski ponuđeni kako bi čitatelj mogao postić sličnu konfiguraciju. Unašamo sd karticu u uređaj, priključujemo potrebnu periferiju i internet kabel, spajamo se koristeći ssh. Dodatno, otvaramo port 443 i 80 u ruteru i usmjerimo ih prema poslužitelju.

Nadograđujemo poslužitelj, instaliramo potrebne programe i omogućimo potrebne servise:

```
# xbps-install -Su
# xbps-install docker ddclient vim
# ln -s /etv/sv/docker /var/service
# docker pull caddy
```

Podesimo DDNS za domenu repozitorija (ovdje se koristi besplatni nsupdate.info servis):

```
# cat <<EOF > /etc/ddclient/ddclient.conf
  protocol=dyndns2
  server=ipv4.nsupdate.info
  login=xbps.nsupdate.info
  password='password'
  xbps.nsupdate.info
  EOF
# ln -s /etv/sv/ddclient /var/service
```

Sada konfiguriramo *Caddy* web poslužitelj koji automatki pribavlja LetsEncrypt ili ZeroSSL certifikate koji su potrebni za https vezu:

```
# mkdir -p /opt/caddy/data
# cat <<EOF > /opt/caddy/Caddyfile
   xbps.nsupdate.info {
     root * /srv
     file_server browse
   }
  EOF
# docker run -d --name Caddy                                \
             --restart=unless-stopped                       \
             -v /opt/caddy/Caddyfile:/etc/caddy/Caddyfile   \
             -v /opt/caddy/data/:/data                      \
             -v /srv/:/srv                                  \
             --net=host                                     \
             caddy
```

5.3.2. Postavljanje repozitorija

Prema Void Linux kontributori [19], svi paketi repozitorija moraju biti potpisani. Kako bismo to učinili, prvo je potrebno generirati ključeve (i dalje se nalazimo u void-packages direktoriju iako bi same ključeve bolje bilo pohraniti na neko drugo mjesto):

```
$ ssh-keygen -t rsa -b 4096 -m PEM -f kljuc.pem
...
```

Potrebno je dodati javni ključ repozitoriju i zatim potpisati sve pakete:

```
$ xbps-rindex --privkey kljuc.pem --sign --signedby "Goran" hostdir/binpkgs/foi
Initialized signed repository (1 package)
$ xbps-rindex --privkey kljuc.pem --sign-pkg hostdir/binpkgs/foi/*.xbps
signed successfully
```

Sada su svi paketi potpisati te možemo prenijeti sav sadržaj iz lokalne datoteke `hostdir/binpkgs/foi` u `/srv` datoteku na poslužitelju koju Caddy poslužuje. Preostaje samo dodavanje novog repozitorija na klijentskoj strani (važno je da datoteka ima *.conf* sufiks):

```
# echo 'repository=https://xbps.nsupdate.info' > /etc/xbps.d/xbps.nsupdate.info.
    conf
```

Možemo pokušati sinkronizirati indeks repozitorija:

```
# xbps-install -S
[*] Updating repository `https://xbps.nsupdate.info/x86_64-repodata' ...
[*] Updating repository `https://alpha.de.repo.voidlinux.org/current/x86_64-
    repodata' ...
```

```
‘https://xbps.nsupdate.info’ repository has been RSA signed by "Goran"
Fingerprint: 0b:35:0a:ca:5a:26:d2:6b:c9:63:41:b3:35:bc:22:35
Do you want to import this public key? [Y/n]
# y
```

Kako bi testirali da li sve radi kako treba, možemo deinstalirati *siege* instaliran iz lokalnog repozitorija i pokušati ga instalirati preko novog online repozitorija:

```
# xbps-remove -R siege
Name  Action    Version           New version            Download size
siege remove    4.1.1_1           -                      -
...
0 downloaded, 0 installed, 0 updated, 0 configured, 1 removed.

# xbps-install siege
Name  Action    Version           New version            Download size
siege install   -                 4.1.1_1                103KB

Size to download:              103KB
...
Do you want to continue? [Y/n]
# y
[*] Downloading packages
siege-4.1.1_1.x86_64.xbps.sig: 512B [avg rate: 22MB/s]
siege-4.1.1_1.x86_64.xbps: 103KB [avg rate: 450KB/s]
siege-4.1.1_1: verifying RSA signature...

[*] Collecting package files
siege-4.1.1_1: collecting files...

[*] Unpacking packages
siege-4.1.1_1: unpacking ...

[*] Configuring unpacked packages
siege-4.1.1_1: configuring ...
siege-4.1.1_1: installed successfully.

1 downloaded, 1 installed, 0 updated, 1 configured, 0 removed.
```

Repozitorij funkcionira. Sve buduće pakete isto treba potpisati i prenijeti ih u `/srv` direktorij na poslužitelju.

5.4. Zašto xbps-src?

Ako paket nije dostupan unutar repozitorija, jasno je da je potrebno koristiti xbps-src ako se paket želi instalirati u sklopu xbps menadžera paketa Void distribucije, no postoji li razlog za korištenje istog čak ako je željeni program već dostupan unutar postojećih repozitorija?

U prošlim sekcijama bio je objašnjen problem inkonzistentnosti dostupnih značajka istog programa u različitim distribucijama te kako je moguće taj problem rješiti korištenjem xbps-src alata, no postoji još mnogo drugih aplikacija u kojima je xbps-src koristan.

Kod novog IT startup poduzeća koje ne očekuje veliki promet, a želi koristiti vlastito sklopovlje za poslužitelje, korištenje ARM arhitekture procesora moglo bi se pokazati jeftinije sa stajališta potrošnje električne energije, a možda i cijene samog sklopovlja. Iako je podrška za ARM sve veća, još uvijek većina distribucija podržava samo x86_64 arhitekturu zbog čega i većina izgrađenih izvršnih datoteka programa isto tako postoji samo za x86_64 arhitekturu. U takvom slučaju, moguće je koristiti xbps-src sa vlastitim repozitorijem u kojem bi se vršila izgradnja backend programa poduzeća zbog čega bi mogli koristiti samo značajke koje su im potrebne i tako povećati efikasnost vlastitih procesa, posebice i radne memorije ako se radi o musl C biblioteci.

Dodatno, ako poduzeće zahtjeva veću razinu sigurnosti, korištenjem alternativnih arhitektura (za koje je potrebno koristiti xbps-src) mogla bi se povećati razina sigurnosti čime bi se izbjegli sigurnosni propusti x86_64 arhitekture kao što su Spectre i Meltdown, a i općenito potencijalno smanjiti mogućnost novih eksploatacija zbog manje koda prisutnog u musl C biblioteci zbog čega je jednostavnije / brže vršiti sigurnosne zakrpe.

Sve navedeno posebice je korisno kod IOT (eng. *Internet Of Things*) uređaja za mikroservise gdje su resursi veoma ograničeni.

6. Zaključak

Menadžment paketa centraliziranim alatom često se čini kompliciranim novom korisniku, no takav alat značajno olakšava održavanje samog sustava na dulje staze. U GNU/Linux sferi postoji puno različitih načina pakiranja paketa, neki od kojih su prikazani u ovom radu.

Općenito, preporuča se korištenje menadžera paketa koji dolazi s distribucijom budući da se taj način osigurava najbolja integracija i efikasnost u smislu potrošnje računalnih resursa, no uvijek postoje iznimke o kojima sam korisnik treba donijeti odluku. Ako npr. trebamo paket zatvorenog koda, a isti je dostupan i unutar repozitorija distribucije i unutar Flathub flatpak repozitorija, korisnik bi mogao izabrati flatpak rutu instalacije koja mu omogućava podešavanje dodatnih ograničenja u smislu direktorija / sučelja računala koje program može pristupiti. U slučaju instalacije programa na poslužitelju, korisniku bi možda bolje bilo koristiti docker ako ima npr. python program koji treba velik broj biblioteka koje korisnik ne želi instalirane van virtualnog okruženja.

Rad je primarno usredotočen na XBPS menadžer paketa koji još nema veliku popularnost, ali nudi različite napredne funkcije i veoma transparentam proces izgradnje paketa zbog čega mali tim kontributora trenutno može održavati pakete za x86_64, i686, armv6l, armv7l i aarch64 platforme, s još više dodanih platforma za koje se ne distribuiraju paketi, ali ih korisnik može sam izgraditi. Dodatno, sve navedene platorme održavaju se za dvije libc C biblioteke (musl i Glibc). Većina ostalih distribucija službeno podržava samo x86_64 platformu s GNU libc C bibliotekom. Navedeno demonstrira dobar dizajn i visoku kvalitetu Void Linux alata za menadžment i distribuciju paketa.

Void Linux alati za izgradnju paketa objašnjeni su i demonstrirani su u zadnjem dijelu rada kako bi čitatelj mogao reproducirati svaki korak izgradbenog procesa.

Popis literature

[1] R. M. Stallman, *Linux and the GNU System*, 2021. adresa: https://www.gnu.org/gnu/linux-and-gnu.en.html (pogledano 30. 8. 2021.).

[2] E. Siever, S. Figgin, R. Love i A. Robbins, *Linux in a nutshell*, 6. izd. 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly Media Inc., 2009.

[3] Void Linux kontributori, *The Void (Linux) distribution*, 2021. adresa: https://voidlinux.org (pogledano 30. 8. 2021.).

[4] Alpine Linux razvojni tim, *Alpine Linux*, 2021. adresa: https://alpinelinux.org/about (pogledano 30. 8. 2021.).

[5] Discord tim, *Discord Terms of Service*, 2020. adresa: https://discord.com/terms (pogledano 30. 8. 2021.).

[6] Flatpak tim, *The Future Of Apps On Linux*, 2021. adresa: https://www.flatpak.org (pogledano 30. 8. 2021.).

[7] Martin Abente Lahaye, *Manage Flatpak permissions*, 2021. adresa: https://github.com/tchx84/flatseal (pogledano 30. 8. 2021.).

[8] Flathub tim, *An app store and build service for Linux*, 2021. adresa: https://flathub.org (pogledano 30. 8. 2021.).

[9] Canonical Ltd., *Snap are universal Linux packages*, 2021. adresa: https://snapcraft.io (pogledano 30. 8. 2021.).

[10] AppImage projekt, *AppImage documentation*, 2021. adresa: https://docs.appimage.org/introduction/index.html (pogledano 30. 8. 2021.).

[11] Docker Inc., *Docker Hub Container Image Library*, 2021. adresa: https://hub.docker.com (pogledano 30. 8. 2021.).

[12] Void Linux kontributori, *XBPS Package Manager*, 2021. adresa: https://docs.voidlinux.org/xbps/index.html (pogledano 30. 8. 2021.).

[13] ——, *Repositories*, 2021. adresa: https://docs.voidlinux.org/xbps/repositories/index.html (pogledano 30. 8. 2021.).

[14] ——, *Xtools*, 2019. adresa: https://man.voidlinux.org/xtools (pogledano 30. 8. 2021.).

[15] ——, *The XBPS source packages collection*, 2021. adresa: https://github.com/void-linux/void-packages (pogledano 30. 8. 2021.).

[16] ——, *Restricted Packages*, 2021. adresa: https://docs.voidlinux.org/xbps/repositories/restricted.html (pogledano 30. 8. 2021.).

[17] ——, *The XBPS source packages manual*, 2021. adresa: https://github.com/void-linux/void-packages/blob/master/Manual.md (pogledano 30. 8. 2021.).

[18] JoeDog, *INSTALLATION PROCEDURE PLATFORM INFORMATION*, 2018. adresa: https://github.com/JoeDog/siege/blob/master/INSTALL (pogledano 30. 8. 2021.).

[19] Void Linux kontributori, *Signing Repositories*, 2021. adresa: https://docs.voidlinux.org/xbps/repositories/signing.html (pogledano 30. 8. 2021.).

Popis slika

1. Hijerarhija direktorija void-packages repozitorija (Izvor: Void Linux kontributori, 2021) 26

Popis tablica

1. Osnovno korištenje menadžera paketa distribucije 7
2. Osnovno korištenje Flatpak menadžera paketa 10
3. Osnovno korištenje Snap menadžera paketa 12
4. Osnovno korištenje Docker menadžera paketa 15
5. Xbps-query 18
6. Xbps-install 19
7. Xbps-remove 21
8. Xbps-pkgdb 22