GYMNASIUM JANA KEPLERA

Parléřova 2/118, 169 00 Praha 6

Správce konfiguračních souborů pro unixové systémy

Maturitní práce

Autor: Jakub Havlíček

Třída: R8.A

Školní rok: 2020/2021

Předmět: Informatika

Vedoucí práce: Emil Miler

Praha, 2023

GYMNASIUM JANA KEPLERA
Kabinet informatiky

ZADÁNÍ MATURITNÍ PRÁCE

Student:	**Jakub Havlíček**
Třída:	**R8. A**
Školní rok:	**2022/2023**
Vedoucí práce:	**Emil Miler**
Název práce:	**Dotfile manager**

Pokyny pro vypracování:

Cílem maturitní práce je vytvořit správce konfihuračních souborů (doftile manager) pro unixové systémy. Dovolí uživateli zálohovat systémové konfigurační soubory, známé jako "dotfiles", a následně konfiguraci aplikovat na jiných zařízeních.

Doporučená literatura:

[1] ROCHKIND, Marc. *Advanced UNIX Programming*. **Addison-Wesley Professional, 2006. ISBN: 978-0-131-41154-8**

[2] STEVENS, Richard W. and Stephen A. RAGO. *Advanced Programming in the UNIX Environment*. **Addison-Wesley Professional, 2013. ISBN: 978-0-321-63773-4**

URL repozitáře:

`https://github.com/Krafi2/ladybug`

student *vedoucí práce*

V Praze dne 29. 9. 2022

Prohlášení

Prohlašuji, že jsem svou práci vypracoval samostatně a použil jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů. Nemám žádné námitky proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Praze dne 24. března 2023 Jakub Havlíček

Poděkování

Děkuji Emilu Milerovi za veškerou pomoc při zpracování této maturitní práce.

Abstrakt

Tato práce představuje návrh a implementaci dotfile manageru pro unixové systémy, který poskytuje pohodlný a flexibilní způsob správy konfiguračnách souborů a systémových aplikací. Program obsahuje vlastní konfigurační jazyk, který uživatelům umožňuje snadno definovat složité konfigurace. Související konfigurace lze strukturovat do modulárního upspořádání, aby i komplexní nastavení zůstala přehledná. Program navíc poskytuje integraci se systémovým správcem balíčků, což uživatelům dovoluje instalovat a spravovat aplikace spolu s jejich konfiguracemi.

Klíčová slova

správce konfiguračních souborů, unix, příkazový řádek

Abstract

This thesis presents the design and implementation of a dotfile manager for Unix systems, which provides a convenient and flexible way to manage dotfiles and their configurations. The dotfile manager includes a custom configuration language that allows users to define complex configurations with ease. Related dotfiles can be structured in a modular arrangement so that even complex configurations remain manageable. Moreover, the dotfile manager provides an integration with the system package manager, allowing users to install and manage packages alongside their dotfile configurations.

Keywords

dotfile manager, unix, command-line

Obsah

1. Teoretická část

1.1 Úvod

Dotfiles jsou konfigurační soubory, které slouží k přizpůsobení operačních systémů založených na Unixu. Tyto soubory jsou obvykle umístěny v domovském adresáři uživatele a řídí různé aspekty systému, včetně nastavení terminálu, konfigurace prostředí, preferencí editoru, atd. Softwaroví inženýři a další technicky zdatní uživatelé často mají své dotfiles organizovány a spravovány pomocí verzovacích systémů, jako je například Git, aby mohly být snadno sdíleny a nasazovány na různých zařízeních.

Cílem této práce je návrh a implementace nástroje zvaného Ladybug, který bude sloužit pro správu konfiguračních souborů na unixových systémech. Program bude umožňovat uživatelům snadno definovat složité konfigurace pomocí vlastního konfiguračního jazyka a organizovat je do modulární struktury. Dále bude umožňovat integraci se systémovým správcem balíčků, aby bylo možné instalovat a spravovat aplikace spolu s jejich konfiguracemi.

1.2 Srovnání s existujícími řešeními

Existuje několik ustanovených programů, které stejně jako Ladybug řeší problém správy systemových souborů. Mezi jeden z nejstarších patří GNU Stow, který podobně jako Ladybug ukládá konfiguraci do centrální složky a vytváří symbolické odkazy na systém uživatele. Mimo to však nepodporuje žádnou další funkcionalitu jako je spouštění skriptů nebo instalace aplikací.

Na druhé straně spektra je automatizační software Ansible od firmy RedHat. Dovoluje uživateli instalovat aplikace i konfigurační soubory, ale kvůli vysoké složitosti je vhodný spíše pro použití administrátory za účelem konfigurace serverů a obchodních aplikací.

Pro uživatele je vhodnější dotfile manager Chezmoi, kerý poskytuje integraci s mnoha programy pro ukládání hesel a dalších tajných informací. Bohužel však nepodporuje přímou instalaci systémových balíčků. Mimo to také pro konfiguraci používá jazyk Toml, který je určen pro obecné konfigurace a není specializován pro užití v dotfile managerech.

2. Implementace

Ladybug je implementován v moderním jazyce Rust. Jedná se o kompilovaný jazyk, takže program lze distribuovat ve formě jednoho binárního souboru.

Jednou z největších výhod Rustu je jeho přístup k paměti. Rust používá koncept „borrow checkeru“, který kontroluje, jak jsou proměnné používány v programu a zabraňuje tak chybám, jako jsou například segmentační chyby, double-free chyby nebo null pointer chyby. Tento mechanismus umožňuje psát bezpečnější kód s menší pravděpodobností chyb spojených s prací s pamětí.

Rust má také velmi rozvinutý ekosystém knihoven a nástrojů, které usnadňují implementaci nízkoúrovňových operací. Na webovém repozitáři „crates.io“ je k dispozici více než 100 000 knihoven, které mohou být použity při implementaci. V neposlední řadě také Rust nabízí vestavěný nástroj pro instalaci knihoven a kompilaci zvaný Cargo, který usnadňuje vývoj a iteraci.

2.1 Interakce se systémem

Ladybug interaguje se systémem uživatele skrze takzvané „poskytovatele“. Každý poskytovatel je zodpovědný za instalaci balíčků z jednoho zdroje. V tuto chvíli program implementuje dva poskytovatele pro instalaci aplikací pomocí package managerů zypper a flatpak a jeden pro instalaci konfiguračních souborů. Modulární polymorfická architektura systému poskytovatelů umožňuje jednoduše přidávat další způsoby instalace.

2.1.1 Zypper

Zypper instaluje balíčky typu *rpm* z uživatelem zvolených repozitářů. Poskytovatel je implementovám jako podproces, který s hlavním programem komunikuje pomocí zpráv v jazyce XML. Implementace byla obtížná, jelikož Zypper nikde nedokumentuje přesný formát XML zpráv, které vypisuje, a v některých případech dokonce ani neposílá validní XML.

2.1.2 Flatpak

Poskytovatel pro každou operaci spouští nový proces, což je poněkud neefektivní. V budoucnosti by bylo lepší prozkoumat možnost interakce skrze nativní API poskytované knihovnou *libflatpak*. Nejenom že by byl díky tomu poskytovatel účinnější, ale zároveň bychom se mohli vyhnout některým chybám ve výstupu programu *flatpak*.

2.1.3 Filesystem

Poskytovatel pro interakci se souborovým systémem se stará o to, aby vytvořil odkazy na konfigurační soubory umístěné v centrální složce. Používá k tomu funkce ze standartní knihovny určené pro manipalici se souborovým systémem.

2.2 Konfigurační jazyk Ldbg

Jazyk Ldbg slouží pro konfiguraci jednotlivých modulů. Dovoluje uživateli specifikovat, jaké aplikace má modul nainstalovat, které konfigurační soubory používá a napsat skripty, které se mají spolu s modulem spustit. Program implementuje lexer a parser, které zpracovávají vstupní soubory.

Dokument se skládá z bloků, které ovládají jednotlivé prvky modulu. Ukázkový dokument by mohl vypadat například takto:

```
# Hlavička specifikuje informace o modulu
unit {
    name: Foo
    desc: Bottom text
    topic: $variable
    members: [member_a member_b]
}

# Blok žádá poskytovatele zypper, aby nainstaloval balíček git
packages(provider: zypper) [
    git
]

# Skript který se spustí při instalaci
deploy {{
    cd ~
    git clone https://github.com/Krafi2/ladybug
}}
```

2.2.1 Lexer

Lexer je napsaný tak, aby neztrácel žádné informace a z výsledného token-streamu šel rekonstruovat původní text. Tato vlastnost bude užitečná v případě, že bude někdy třeba automaticky modifikovat soubory.

Lexer zpracovává vstup do jednácti různých druhů tokenů:

- **Ctrl** - kontrolní znaky jako závorky a čárky
- **Ident** - identifier
- **IdentOrStr** - buď identifier, nebo pouze string
- **Str** - string
- **DelimStr** - string, který je vymezený pomocí uvozovek
- **Bool** - booleanovská hodnota
- **Var** - reference na proměnnou
- **Code** - blok kódu
- **Comment** - komentář
- **Space** - whitespace
- **Line** - nový řádek

2.2.2 Parser

K implementaci parseru je použita knihovna Chumsky, která umožňuje konstrukci recursive descent parserů skrze metodu jejich kombinace. Knihovna poskytuje různá kombinační a parsovací primitiva, která ulehčují implementaci jednoduchých sub-parserů.

Pokud parser narazí na naplatnou syntaxi, pokusí se vrátit do správného stavu, aby mohlo pokračovat ověřování syntaxe. Díky tomu lze diagnostikovat další chyby i v neplatných souborech.

Uvnitř programu existuje několik desítek jednotkových testů pro ověření správnosti jednotlivých sub-parserů. Díky tomu je jednoduché modifikovat kód parseru a ověřit, že stále funguje tak, jak má.

2.3 Použité knihovny

Projekt využívá následující knihovny:

- **ariadne** *(MIT)* - generace uživatelsky přívětivých chybových hlášení pro parsery
- **chumsky** *(MIT)* - tvorba a kombinace parserů
- **clap** *(MIT)* - usnadňuje tvorbu aplikací používaných přes příkazový řádek
- **color-eyre** *(MIT)* - polymorfický typ pro chyby
- **directories-next** *(MIT)* - interakce se systémovými adresáři
- **indent** *(MPL 2)* - manipulace s textem
- **indicatif** *(MIT)* - ukazatele průběhu pro příkazový řádek
- **nix** *(MIT)* - unixová primitiva
- **quick-xml** *(MIT)* - parsování XML
- **thiserror** *(MIT)* - automatická generace chybových typů
- **timeout-readwrite** *(MIT)* - I/O funkce s časovým limitem
- **tracing** *(MIT)* - framework pro shromažď ování logů
- **tracing-subscriber** *(MIT)* - dodatečná funkcionalita pro vypisování logů

3. Technická dokumentace

3.1 Testovací prostředí

Pro spuštění testovacího prostředí je potřeba aplikace *podman*. Repozitář obsahuje skript, který spustí kontejner s testovacím prostředím, ve kterém je nainstalovaný program spolu s ukázkovou konfigurací. Pokud máte naistalovaný podman, stačí uvnitř repozitáře zavolat `./run.sh` a otevře se testovací prostředí.

Adresář s konfiguračními soubory se nachází v */dotfiles*. Můžete je modifikovat nebo pokračovat s ukázkovými. Dále postupujte podle dokumentace.

3.2 Moduly

Ladybug dovoluje uživateli strukturovat svoji konfiguraci do modulů, které potom mohou být instalovány ve skupinách. Platí, že modul závisí na svém rodiči a jeho rodič bude vždy naistalován první.

Každý modul se nachází ve vlastní složce, jejíž jméno je cesta k němu *oddělená tečkami*. Jedinou vyjímkou je hlavní modul *main*, který je na stejné úrovni jako jeho děti. Adresář ukázkové konfigurace by potom mohl vypadat takto:

- **foo/**
- **foo.bar/**
- **foo.bar.dar/**
- **main/** (hlavní modul)

3.3 Specifikace konfigurace

Pro konfiguraci se používá jazyk Ldbg (soubory také končí .ldbg). Každá konfigurace se skládá z bloků, které mají název, potom může následovat seznam argumentů v kulatých závorkách a na konci máme tělo bloku.

Existují tři typy bloků. Mapový blok používá složené závorky a obsahuje libovolný počet klíčů a jejich hodnot. Seznamový blok je ohraničen hranatými závorkami a obsahuje seznam výrazů. Jako poslední máme skriptový blok, který užívá dvojité složené závorky a obsahuje libovolný skript.

Pro lepší znázorní můžeme uvést takovýto příklad:

```
mapa(parametr1: hodnota, parametr2: "hodnota") {
    klíč: hodnota
}
seznam [
    výraz1
    výraz2
]
skript {{
    echo "Hello world"
}}
```

Ldbg obsahuje čtyři typy výrazů:

- **Řetězce** - Řetězce nemusí výt ohraničeny uvozovkami, pokud neobsahují žádné syntakticky významné znaky. Pokud potřebujete, můžete však uvozovky použít.
- **Proměnné** - Na proměnné můžete odkazovat pomocí syntaxe `$název`.
- **Booleany** - Booleany mají dvě hodnoty: `yes` a `no`.
- **Seznam** - Seznam výrazů oddělených mezerami.
- **Kód** - Libovolné kusy kódu ohraničeny pomocí `{{` a `}}`.

3.3.1 Globální konfigurace

Globální konfigurace se nachází v souboru */.config/ladybug/config.ldbg*. Skládá se z jednoho bloku *config*, který obsahuje jeden klíč *dotfiles*. Ukázková konfigurace může vypadat takto:

```
config {
    dotfiles: ~/dotfiles
}
```

3.3.2 Konfigurace modulů

Každý modul obsahuje vlastní konfigurační soubor, který se jmenuje <jméno modulu>.ldbg. Například:

- **foo/foo.ldbg**
- **foo.bar/bar.ldbg**
- **main.ldbg**

Každý modul musí začínat hlavičkou s těmito klíči:

- **name** (string) - povinné pole, specifikuje jméno modulu
- **desc** (string) - povinné pole, specifikuje popisek modulu
- **topic** (string)- nepovinné pole, specifikuje kategorii modulu
- **members** (list) - nepovinné pole, specifikuje podmoduly

```
config {
    name: Name
    desc: Desc
    topic: test
    members: [member1 member2]
}
```

Potom může následovat blok *env* určující proměnné. Kromě těchto bloků také moduly dědí prostředí od svých rodičů.

```
env {
    foo: bar
    list: [string_a string_b]
}
```

Blok *files* dovoluje uživateli instalovat konfigurační soubory z jeho konfigurace. Přijímá čtyři různé parametry:

- **method** (string) - metoda instalace. Možné hodnoty: *softlink* (výchozí) | *hardlink* | *copy*
- **conflicts** (string) - chování v případě, že instalovaný soubor již existuje. Možné hodnoty: *rename* (výchozí) | *remove* | *abort*
- **source** (string) - adresář uvnitř modulu, ze kterého bude probíhat instalace. Výchozí hodnota je modul samotný.
- **target** (string) - cíl umístění souborů. Tato hodnota je *povinná*.

```
# Vytvoří symbolický odkaz '~/.config/ladybug/config.ldbg'
# směřující na '$module/config/config.ldbg'
files(method: link, conflicts: rename, source: config, target: ~/.config/ladybug) [
    config.ldbg
]
```

Blok *packages* slouží k instalaci aplikací pomocí různých poskytovatelů. Parametr *provider* specifikuje, kterého poskytovatele použít. Přijímané hodnoty jsou *zypper* a *flatpak*.

```
packages(provider: zypper) [
git
    rustup
]
```

Bloky *deploy*, *remove* a *capture* obsahují skripty, které se spouštějí za různých podmínek. *deploy* se spouští při instalaci, *remove* při odinstalaci a *capture* ve chvíli, kdy uživatel zadá příkaz `ladybug capture`. Přijímají tři nepovinné parametry:

- **shell** (list) - jaký program použít pro spuštění skriptu. Řetězec *%c* je nahrazen kódem skriptu. Výchozí hodnota je `[bash -c %c]`, ale může být i například `[python -c %c]`.
- **stdout** (bool) - jestli má skript zobrazovat svůj výstup
- **workdir** (string) - adresář, ve kterém bude skript spuštěn. Výchozí hodnota je adresář modulu.

3.4 Dostupné příkazy

Funkcionalita programu je rozdělena mezi několik příkazů, které se spouštějí přes `ladybug <příkaz>`. Pokud není specifikována možnost *–no-root*, program se pokusí získat kořenová privilegia pomocí programu *sudo*, aby mohl spravovat systém.

Možnosti:

- **–no-root** - Nepoužívej kořenová privilegia
- **–root** - Potvrď kořenová privilegia
- **–config <CONFIG>** - Manuálně nastav konfigurační soubor
- **–dotfiles <DOTFILES>** - Manuálně nastav adresář konfigurace

3.4.1 Deploy

Příkaz *deploy* provede instalaci specifikovaných kategorií. Pokud nejsou žádné kategorie uvedeny, nainstaluje všechny moduly.

```
deploy [argumenty] [kategorie]
```

Argumenty:

- **[kategorie]** - Dobrovolný seznam kategorií, které budou nainstalovány

Možnosti:

- **–revert-all** - Pokud dojde k chybě, vrať všechny změny
- **–dry-run** - Neprováděj žádné změny na systému uživatele
- **–abort-early** - Pokud dojde k chybě, okamžitě ukonči instalaci

3.4.2 Remove

Příkaz *remove* odinstaluje specifikované kategorie. Pokud nejsou žádné kategorie uvedeny, odinstaluje všechny moduly.

```
remove [argumenty] [kategorie]
```

Argumenty:

- **[kategorie]** - Dobrovolný seznam kategorií, které budou odinstalovány

Možnosti:

- **–dry-run** - Neprováděj žádné změny na systému uživatele

3.4.3 Capture

Příkaz *capture* spustí skripty capture všech uvedených kategorií. Pokud nejsou žádné kategorie uvedeny, aktivují se všechny moduly. Účelem tohoto příkazu je aktualizovat konfigurací, které nejsou uloženy v souborech.

```
capture [argumenty] [kategorie]
```

Argumenty:

- **[kategorie]** - Dobrovolný seznam kategorií, které budou aktualizovány

Možnosti:

- **–dry-run** - Neprováděj žádné změny na systému uživatele

Závěr

Vrámci této práce byl navržen a implementován dotfile manager pro unixové systémy včetně vlastního konfiguračního jazyka a systémovou integrací. Finální program budu odteď rozhodně používat pro osobní potřebu.

Řekl bych, že nejvíce mě obohatila zkušenost s návrhem gramatiky a implementací vlastního parseru. Je složité dobře vybudovat gramatiku tak, aby byla ergonomická, ale parser zůstal jednoduchý. Na druhou stranu systémová integrace byla spíše úmorná kvůli špatně řešitelným chybám v cizích programech.

Seznam obrázků

Seznam tabulek