Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra teoretické informatiky

Diplomová práce

Comma-shell, interaktivní debugger shellu

Bc. Tomáš Nesrovnal

Vedoucí práce: Ing. Jan Baier

17. dubna 2017

Poděkování

Doplňte, máte-li komu a za co děkovat. V opačném případě úplně odstraňte tento příkaz.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 17. dubna 2017 .....

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Nesrovnal, Tomáš. *Comma-shell, interaktivní debugger shellu.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017. Dostupný také z WWW: ⟨`https://github.com/nesro/nesrotom-dip-2016`⟩.

Abstrakt

V několika větách shrňte obsah a přínos této práce v češtině. Po přečtení abstraktu by se čtenář měl mít čtenář dost informací pro rozhodnutí, zda chce Vaši práci číst.

Klíčová slova Nahraďte seznamem klíčových slov v češtině oddělených čárkou.

Abstract

Sem doplňte ekvivalent abstraktu Vaší práce v angličtině.

Keywords Nahraďte seznamem klíčových slov v angličtině oddělených čárkou.

Obsah

Seznam obrázků

Úvod

Grafické uživatelské rozhraní (GUI) se jednoduše ovládá, ale ne vždy je k dispozici. To platí zejména při ovládání serverů.

Rozhraní příkazové řádky (CLI) je základní textové prostředí pro komunikaci s operačním systémem. Umožňuje spouštění programů, vkládat vstupní data a sledovat výstupní data v terminálu.

Jedním ze základních bodů UNIXové filosofie je mít jednoduché programy, které dělají pouze jednu věc, ale dělají ji dobře. To platí zejména pro základní příkazy ze sady GNU coreutils, tedy příkazy pro základní manipulaci se soubory, shellem a textem.

Tyto základní příkazy je možné řetězit a tím vytvářet užitečné jednořádkové skripty.

TODO: napsat o tom, ze pro začátečníka to může byt neintuitivní, musí si pamatovat spoustu příkazu. o návratových kódech, o historii prikazu a o logovani

Motivace

Příkazová řádka a základní práce s ní by měly být jedna z prvních věcí, které by se uživatel měl naučit, pokud chce umět používat svůj systém na vyšší úrovni. Stejně jako v programování platí, že počítač provádí dokonale a přesně to, co mu řekneme. Ale i při sebemenší chybě v našem příkazu selže celá sekvence akcí, které se měli provést. Není těžké udělat i nějakou fatální chybu, po které není snadné uvést vše do původního stavu.

Náš projekt, Comma-shell, si dává za úkol vytvořit z příkazové řádky místo, ve kterém je těžší nějaké chyby udělat, případně před chybou varovat, nebo vysvětlit, jak danou věc udělat lépe. Důležitý cíl je také umožnit nastavit toto prostředí tak, aby bylo snadno použitelné, nebylo potřeba externích terminálů, nebo grafických aplikací a aby bylo možné snadno omezit repetitivní výstupy programu. Comma-shell se dělí na tři části, každá pomáhá řešit jiný problém.

Comma-shell hooks, nebo-li části kódu, které lze vykonat před, nebo po vykonání spuštěného příkazu a dokonce zabránit jeho spuštění umožňují ochránit uživatele před vykonáním příkazu, který je v například nějakém smyslu nesprávný, špatný, nebo je v nepořádku. Pokud tedy napíše příkaz, který obsahuje nějaký překlep, nebo chybu, která by způsobila neočekávané chování, může být uživatel varován, než bude takový příkaz proveden. Nejde jen o ochranu uživatele před špatnými příkazy. Díky informacím o spouštěném příkazu a následně o jeho výsledku lze psát různé skripty, které by jinak bylo složité vytvořit.

Comma-shell debugger, tedy ladící program pro příkazy spouštěné z interaktivní příkazové řádky. Pokud uživatel píše složitější program, obsahující například smyčku, nebo roury a příkaz nefunguje tak, jak uživatel očekává, tak nejjednodušší je rozdělit si příkaz na podpříkazy a ujistit se, že každý příkaz dělá to, co od něj uživatel očekává. Comma-shell debugger by měl tento postup automatizovat.

Poslední částí jsou takzvané bezpečné příkazy. Pokud uživatel vykoná příkaz, jehož efekt byl jiný než očekával, chce spustit opačný příkaz, aby napravil svůj omyl, nebo má často mnohem větší problém, pokud šlo například o nesprávné použití programu pro mazání souborů. Cílem bezpečných příkazů je tedy zaprvé umožnit vrátit provedené změny, ale i informovat uživatele o tom, co se stane. Pokud uživatel používá ke specifikování souborů, které mají být cílem nějakého příkazu, je častý omyl vybrat nějaké soubory nechtěně. Nebo například provést příkaz ze špatného adresáře.

Kapitola 1

Definice a pojmy

Příkazová řádka, anglicky Command Line Interface, je uživatelské rozhraní ovládané příkazy. Příkazy spuštěné z příkazové řádky vypisují výstup také na příkazovou řádku a další interakce se spuštěným programem probíhá zase zadáváním příkazů. Na rozdíl od textového nebo grafického rozhraní uživatel nemůže nijak zasahovat do již vypsané či vykreslené části.

TODO: obrazek prikazove radky, tui a gui

Shellem se obecně myslí uživatelské prostředí pro využívání funkcí jádra operačního systému. V této práci budeme slovem shell označovat právě interpret příkazů v příkazové řádce. Tedy program, ovládaný příkazy, který umí spouštět ostatní programy a zajišťovat jejich vstup a zobrazovat jejich výstup.

Prompt je krátký řetězec znaků, který se objeví na začátku řádky, do které se bude vepisovat příkaz v příkazové řádce. Je používán k zobrazování důležitých informací, jako například, ve kterém adresáři se uživatel nachází či jaké je jméno počítače, na kterém se nachází.

GNU je projekt zabývající se tvorbou a propagací svobodného software. Cílem je vytvořit kompletní svobodný operační systém unixového typu. Protože jádro operačního systému od GNU s názvem Hurd nebylo tak úspěšné, začalo se používat jádro Linux. Kombinace operačního systému GNU a jádra Linux se označuje jako GNU/Linux. GNU vytvořilo i svůj shell, Bash. Sada základních příkazů, jejichž existence je předpokládána v nějaké formě na každém operačním systému, se jmenuje GNU coreutils. Patří do ní příkazy jako ls, cp, rm, atd. Je důležité si uvědomit, že příkazy jako cd, echo jsou vestavěné příkazy shellu Bash.

Terminál byl z historického hlediska elektronické zařízení k základní komunikaci s počítačem. Dnes se v operačních systémech spouští terminálové emulátory, ve kterých lze spustit různé programy. Nejčastěji je to právě shell.

Provádění příkazů můžeme rozdělit na dvě kategorie. Jednou z nich je interaktivní režim, kdy interpret spouští příkazy tak, jak je uživatel píše do příkazové řádky. V tomto režimu může uživatel také používat klávesové zkratky, aliasy, automatické doplňování a další funkce v interaktivním režimu. Druhou

kategorií je dávkový režim, kdy uživatel nejprve příkazy napíše do souboru. Takovému souboru obsahujícímu příkazy se říká skript. Takový soubor může být zavolán interpretem shellu, který postupně vykoná všechny příkazy. Tento postup se hodí pro složitější a opakující se příkazy.

Kapitola 2

Historie shellu a dnešní využití

V této kapitole stručně shrneme historii rodin nepoužívanějších operačních systému a shellů, které se v nich používají.

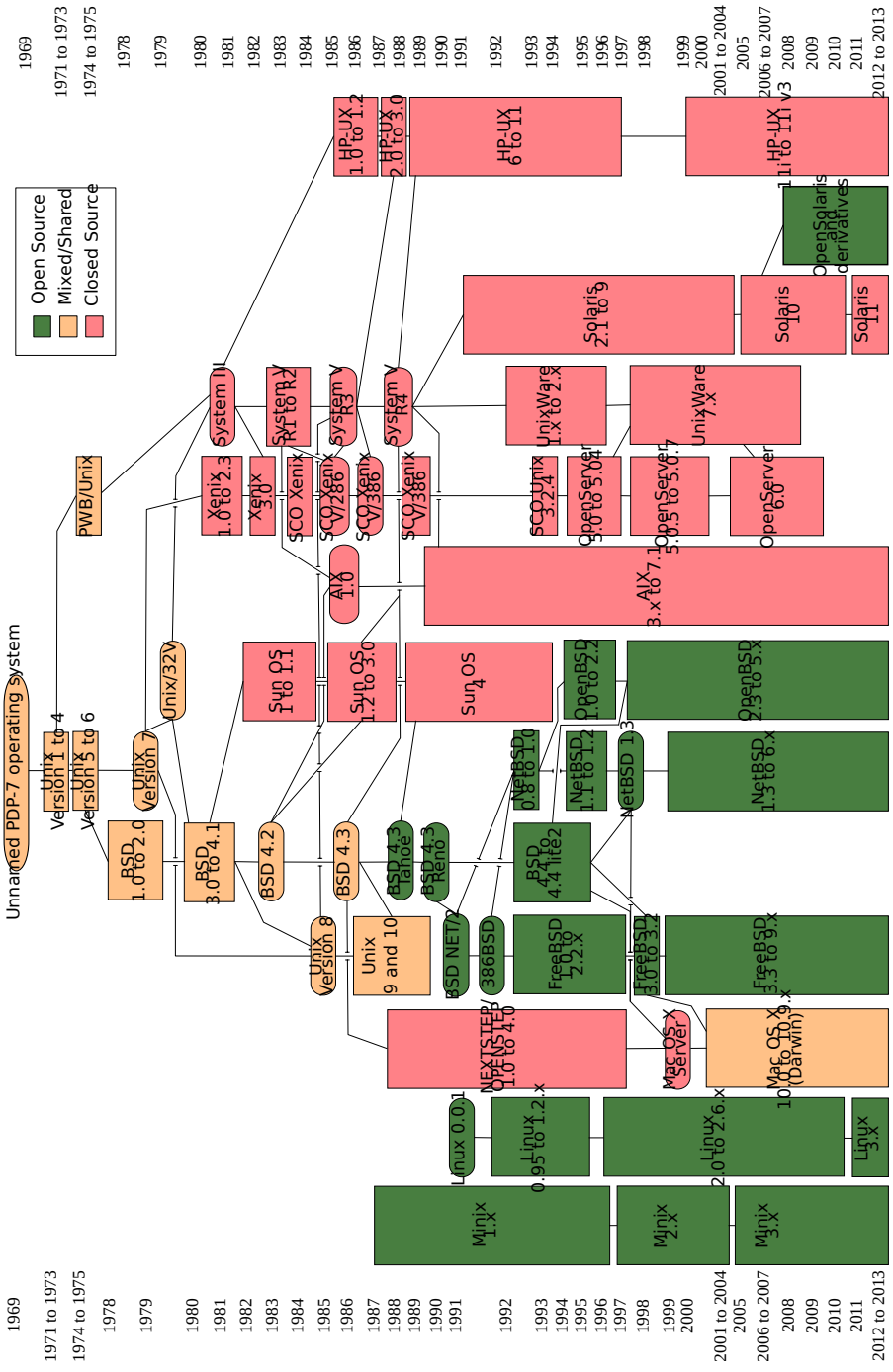
2.1 Windows

Operační systém Windows je obecně uživatelsky přívětivější, protože skoro vše je možné nastavit přes grafické rozhraní. Ovšem ne vždy je grafické prostředí dostupné a příkazová řádka je jediná možnost, jak ovládat počítač.

2.2 UNIX

Unix je rodina operačních systémů, které zvládnou spustit více úkonů najednou a do kterých se v jeden okamžik může připojit více uživatelů. Historie těchto sytému sahá až na konec sedmdesátých let, kdy v Bell laboratořích firmy AT&T byl v jazyce C vytvořen systém UNIX, jehož název byl zaregistrován jako ochranná známka. Časovou osu ukazuje obrázek 2.1. Specifikace "Single UNIX Specification"je souhrn standardu, které operační systém musí splňovat a dodržovat, aby se mohl označovat za UNIX.

Filosofie Unixu je sada doporučení a pravidel, která vznikla postupem času dle zkušeností tvůrců Unixu. Filosofie Unixu zdůrazňuje tvoření jednoduchého, přehledného a hlavně snadno rozšiřitelného a znovupoužitelného softwaru. Váží si programátorův čas více, než čas práce počítače. Nejznámější pravidlo je tvořit programy, které dělají jen jednu věc a dělají jí správně, rychle a bez chyb. Takové programy lze použít jako filtry a skládat je za sebe. To se často používá v příkazové řádce, kdy se pomocí rour spojují výstupy programů se vstupy jiných programů. Zdánlivě složitý úkol tak lze udělat rychle a efektivně bez nutnosti programování v nějakém jazyce.

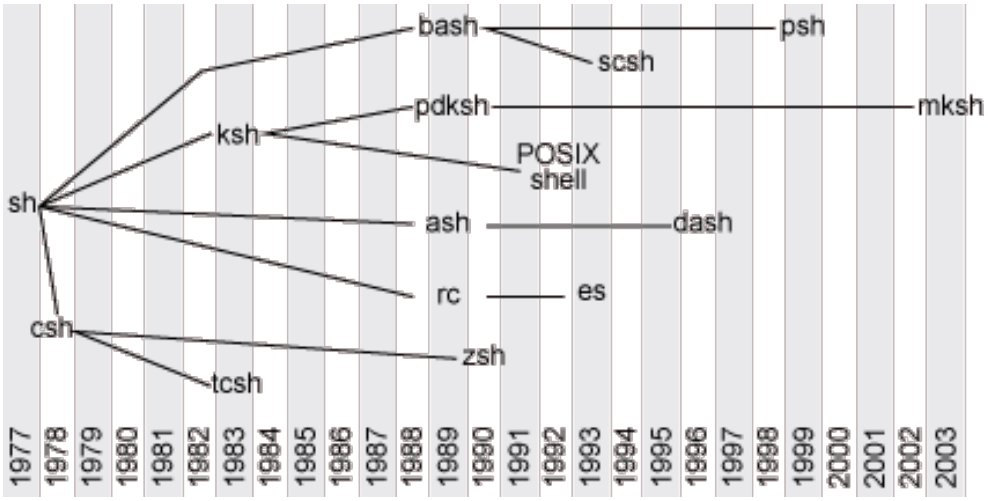


Obrázek 2.1: Historie unixu

2.3 Linux

Linux je otevřený a svobodný Unix-like operační sytém. Linuxem se obecně myslí GNU operační systém s jádrem Linux. Unix-like znamená, že přestože je takový operační systém velice podobný systému Unix, nemá certifikaci "Single UNIX Specification". Linuxová distribuce je GNU/Linuxový systém, který většinou obsahuje balíčkovací systém, kterým je snadné doinstalovat ostatní programy. Linuxové distribuce jsou i předpřipravené, s grafickým prostředím. Například v této práci ukazujeme návod na vytvoření takového připraveného systému, GNU/Linuxové distribuci Debian s předpřipraveným grafickým prostředím LXDE a aplikacemi z rodiny LXDE.

2.4 Shell



Obrázek 2.2: Historie Unixového Shellu

2.5 cmd.exe

Shellem v operačním systému MS-DOS a PC-DOS byl COMMAND.COM, 16-bitový intepretr příkazů. V novějších systémech Windows byl tento shell nahrazen CMD.EXE, 32-bitovým interpretem příkazů. V CMD.EXE byla dostupná emulace COMMAND.COM, pro zpětnou kompatibilitu.

Tato příkazová řádka, shell, již umí přesměrovávat vstupy a výstupy a to jak do souboru tak mezi procesy. Také je schopná vykonávat příkazy ze souborů, které se nazývají dávkové a zpravidla mívají příponu bat.

2.6 Power Shell

Protože je základní příkazová řádka Windows omezená co se funkcionality týče, existují další shelly pro operační systémy Windows, často i s otevřeným zdrojovým kódem. Firma Microsoft chtěla vytvořit i svůj shell, který bude umět více věcí. S jeho vývojem začala v roce 2002 a po 3 letech jej zveřejnila jako Microsoft Command Shell (MSH) s krycím jménem Monad, který byl později přejmenován na Power Shell.

Power Shell je integrován s platformou .NET od Microsoftu. Je tedy možné využívat již vytvořené funkce a nástroje, které usnadní a urychlí vývoj.

Oproti cmd.exe umí Power Shell mnoho věcí. Power Shell umožňuje spouštět příkazy v intervalech, nebo v nějaký přesný čas. Přibyly konstrukce pro větvení. Příkazy je možné spouštět na pozadí, nebo vzdáleně.

2.7 v9 - Thomson Shell

Pro první shell pro systém UNIX napsal Ken Thomson v roce 1971 a byl nazván V9 shell. Jednalo se již o uživatelský program, tedy spuštěný mimo jádro operačního sytému. Shell byl velice jednoduchý a některá základní funkcionalita byla umožněna dalšími programy. Například expanze parametrů byla zajišťována příkazem glob. Základní větvení, if, bylo také jako samostatný program.

V9 shell přinesl syntaxi pro přesměrování a pipelining. Uměl spouštět více oddělených příkazů a dokázal příkazy spouštět na pozadí. Naopak chyběla třeba podpora pro spouštění skriptů.

Díky jednoduchosti a oddělenosti ostatní funkcionality měly zdrojové kódy pouze 900 řádek v jazyce C.

Originalní zdrojové kódy jsou stále dostupné, `https://github.com/yvesnrb/Thompson-Shell`. Projekt Osh, `https://v6shell.org/`, obsahuje jak port tohoto původního shellu, tak i jeho vylepšenou verzi.

2.8 Sh - Bourne Shell

Bourne shell byl vytvořen pro UNIX V7 v roce 1977 Stephenem Bournem v laboratořích Bell a AT&T. Na rozdíl od Thomsonova v9 shellu umí Bourne shell spouštět příkazy ze souborů, skriptů. Šlo tedy psát znovupoužitelné sady příkazů, které zjednodušili a urychlili vývoj různých aplikací. Interaktivní režim je samozřejmě podporován také.

Bourne shell přinesl také podporu smyček, proměnných, signály, subshelly a HERE dokumenty. Z tohoto shellu poté začaly vznikat další, jak je vidět na obrázku 2.2.

Dnes se Bourne shell stále používá. Je totiž menší, rychlejší a stabilnější než ostatní shelly. Má také minimum závislostí na externí knihovny.

2.9 Dash - Debian Almquist Shell

todo

`https://cs.wikipedia.org/wiki/Debian_Almquist_shell` `https://wiki.archlinux.org/index.php/Dash`

2.10 Bash - Bourne Again Shell

2.11 Zsh - Z Shell

todo

`http://zsh.sourceforge.net/FAQ/`

2.12 Práce v příkazové řádce

todo: tady bych chtel shrnout obecne to, ze se pisou prikazy jako text a uzivatel vse vidi jako text. co z toho plyne za nasledky a jak je snadne udelat chybu

2.13 Nastavení shellu

GNU/Linuxové distribuce, které nabízejí předpřipravené prostředí, mají pro výchozí shell bash připravené takzvané rc soubory, které nějakým způsobem upravují běžící interaktivní shell. Jednou z nejviditelnějších změn je nastavení proměnné PS1, o které bude řec v pozdější kapitole, která určuje styl promptu. Ve výchozím nastavení bashe, tedy po spuštění bashe bez načtení rc souborů (bash –norc), se v PS1 zobrazuje pouze název shellu, jeho verze a zdali je uživatel root. Po načtení výchozích rc souborů se v PS1 ukazují informace jako je například jméno uživatele, jméno počítače a co je velmi důležité, jméno aktualního adresáře.

Tyto informace jdou snadno získat použitím některých základních příkazů, například whoami, hostname, pwd. Protože tyto informace jsou důležité a potřebujeme je vedět pořád, chceme je mít pořád na očích.

2.14 Terminál

todo: asi by se hodilo napsat i neco o tech programech, ve kterych shell bezi

Napsat o tom, ze cilem prace je usnadnit praci v prikazove radce a sepsat zakladni funkcionalitu debuggeru.

2.15 Existující řešení

Mezi seznamem různých zajímavých řešeních awesome (github.com/sindresorhus/awesome) existuje i podsekce awesome-shell (github.com/alebcay/awesome-shell), ve které lze nalézt spoustu užitečných nástrojů pro práci s příkazovou řádkou, nebo psaním skripů v bashi.

Spousta těchto nástrojů je nad rámec této práce.

2.16 Bash frameworky pro psaní skriptů

Existuje mnoho projektů, jejichž cílem je vytvořit framework v bashi, který má nějakým způsobem zjednodušit vytváření, především větších, skriptů v bashi. Spousta dnešních vývojářů je zvyklá na objektově orientovaný přístup k programovaní a je tedy pro těžké vytvořit větší program, který by zůstal přehledný.

2.16.1 Bash OO framework

Bash OO framework (`github.com/niieani/bash-oo-framework`) je framework napasny v bashi, ktery umoznuje vytvaret tridy, vyjimky a testy. Jeho cílem je vytvořit prostredi pro psani skriptu, kde se bude snadneji psat citelny kod, bez casti, ktere se opakuji.

2.17 Bash frameworky pro správu doplňků

Existuje spousta věcí, které uživatel příkazové řádky potřebuje občas řešit. Řešení pro daný problém je víc. Buď si uživatel vyřeší problém sám, nebo bude hledat řešení na internetu. Shellovské frameworky, které umějí i spravovat doplňky, mohou mít řešení pro daný problém. Instalace a použití pak bude velmi jednoduché, intuitivní a bude zde nějaká záruka o fuknčnosti a kvalitě.

Přestože je tato práce převážně o shellu bash, zsh je v tomto mnohem rozšířenější.

2.17.1 Oh My Zsh

Oh My Zsh (`ohmyz.sh`) je framework pro Zsh.

2.17.2 Bash-it

Bash-it (`http://github.com/Bash-it/bash-it`)
(itsfoss.com/bash-it-terminal-tool/)

2.18 Bash frameworky pro úpravu promptu

2.18.1 Liquid Prompt

Projekt liquidpromp `github.com/nojhan/liquidprompt` je adaptivní prompt v interaktivním bashi.

2.18.2 Sexy Bash Prompt

Projekt sexy-bash-prompt `https://github.com/twolfson/sexy-bash-prompt` je další používaný prompt v interaktivním bashi.

Kapitola 3

Analýza a návrh

3.1 Cíl práce

todo

3.2 Fungování shellu

3.2.1 Životní cyklus příkazu

todo: popsat, co vsechno se stane, od napsani prikazu az po jeho vykonani

3.2.2 Gramatika shellu

Soubor s gramatikiou bashe, parse.y, ma pres 6000 radek. Chtel bych zde napsat zjednodusenou gramatiku, ktera by se dala snadno pochopit (ono to zas tak slozite neni).

Popsat jakym zpusobem parsuje gramatiku BASH (yacc) a jakym to delaji parsery bashlex a bashast.

3.2.2.1 Projekty parsující shell

https://github.com/bemeurer/beautysh/blob/master/beautysh/beautysh.py

3.2.2.2 Bashlex

https://github.com/idank/bashlex/blob/master/bashlex/parser.py

3.2.2.3 Bashast

https://github.com/neloe/libbash/blob/master/bashast/bashast.g

3.2.3 Spouštění příkazů

Popsat základní principy jak funguje shell. Popsat procesy v unixu, fork, exec, co vsechno se musi stat, aby shell mohl spustit prikaz.

3.2.4 Struktura BASHe

todo: Zdrojovy kod je rozdelen do souboru, mozna by bylo dobre popsat popsat co ktery soubor dela, aby si ctenar udelal alespon trosku obrazek.

3.3 Debugovaní shellu

tohle je jeden bod zadani: "Proveďte rešerši existujících nástrojů pro statickou analýzu, krokování a hledání chyb v BASH skriptech."

3.3.1 Debugovaní Bashe

todo: navod na debugovani bashe primo pomoci gdb. (tohle nakonec pro moji praci nebylo potreba)

3.3.2 Interní nástroje

todo: popsat to, jake nastroje ma v sobe bash zabudovane v zakladu

3.3.2.1 Debugovací mód BASHe (jak funguje shopt s extdebug)

shopt s extdebug

3.3.2.2 set x, u, v, e

priklady do skriptu

3.3.2.3 PS0, PS4

PS0 bude v novém bashi, my ji proto nebudeme používat, PS4 se vypisuje při debugovaní. todo: ps0 se nakonec nepovedlo

3.3.3 Externí nástroje

3.3.4 BASH Debugger

todo: popsat jak funguje, co vsechno umi, nejake priklady `http://bashdb.sourceforge.net/`

3.3.5 BashEclipse

BashEclipse je plugin do Eclipse, ktery umi krokovat v gui eclipse.
`http://unix.stackexchange.com/questions/131491/is-there-a-gui-debugger-for-shell-scripts` `https://sourceforge.net/projects/shelled/` `https://sourceforge.net/projects/basheclipse/`

3.4 Možnosti debugování v interaktivním shellu

Nebyl nalezen žádný nástroj pro debugování interaktivního shellu. todo

3.4.1 Definice debugování interaktivního shellu

todo: nema cenu debugovat jednoduche prikazy, ale slozite ano. chceme umet rozkrokovat pipy a subshelly (tedy to co nam to dela ted)

3.4.2 GNU Readline

GNU Readline umožnuje přemapovat enter tak, abysme mohli spustit prikaz v nami definovane funkci. Problémem je, že takto upravený příkaz se uloží do historie. Dalším problémem jsou víceřádkové příkazy, tedy takové, pro jejichž napsání musíme několikrát zmáčknout enter. TODO: ukázka.

3.4.3 Napsání nového REPLu

Zprovoznění základní funkcionality by bylo snadné, vzhledem ke komplexnosti BASHe však téměř nemožné mít stejné chování jako v BASHi.

3.4.4 DEBUG trap

Současné řešení. Při zapnutém extdebug je možné příkazy nepustit a jen evalovat poslední příkaz z historie. TODO: je potřeba popsat základní chování historie (např mezera na začátku příkazu, atd.)

3.5 Statická analýza skriptů

bash ma validaci syntaxe: bash -n, ale to nezjisti skoro zadne chyby.

3.5.1 Check Bashisms

`http://checkbashisms.sourceforge.net/` Některé skripty cheme kvůli rychlosti spouštět v Bourne-Shellu. Protože ten je pouze podmnožina bashe, tento nástroj najde výskyty syntaxe, která je podporovaná pouze v bashi a v Bourne-Shellu nemusí fungovat správně.

3.5.2 Explain Shell

Tato analýza nehledá chyby, ale vyparsuje vložený kód a vyhledá příslušnou dokumentaci v manuálových stránkách. Využívá knihovny bashlex, kterou využijeme v našem debuggeru. `http://www.explainshell.com/`

3.5.3 ShellCheck

todo: tady bych se mohl rozepsat vic. o tom, ze to je napsane v haskellu, o tom ze existuje databaze veci, ktere se ve skriptech hledaji a ze ke kazdym je wiki..

Kapitola 4

Realizace

v zadani je: Navrhněte a implementujte nástroj, který umožní psát uživatelské skripty pro analýzu příkazů a ovlivňování jejich spouštění a vykonávání. = to jsou hooky

Nástroj musí umožňovat krokovat složitější skripty po jednotlivých příkazech. = to je debugger

Pro analýzu spouštěných skriptů využijte vhodný nástroj z rešeršní části. = to je shellcheck

4.1 Nespouštění příkazů

Pro zabránění spouštění používáme DEBUG trap. todo: popsat problemy a reseni

4.2 Implementace debuggeru

todo: napsat o tom, ze cast je napsana v pythonu a ze se cela vec spousti v pre-hooku, takze to je presne ten cas, kdy se muze prikaz debuggovat a originalni prikaz se nespusti.

popsat parsovani v bashlex a jak se cela vec predava do bashe

4.3 Hooks

Aby byl kód přehledný, kromě bezpečných příkazů je funkcionalita je rozdělena do hooků, nebo-li modulů, které obsahují kód, který je spuštěn před, nebo i po vykonání příkazu. Kód vykonaný před příkazem může rozhodnout, zda-li má dojít k zabránění vykonání příkazu.

4.3.1 Implementace hooků

todo: napsat o tom, že se hooky pousti automaticky pred i po a ze je snadne vytvorit vlastni hook

4.3.2 Hooky před vykonáním příkazu

todo: napsat co všechno můžu dělat

4.3.3 Hooky po vykonáním příkazu

todo: napsat co všechno můžu dělat

4.3.4 Předvytvořené hooky

todo: napsat o tom, že

4.3.5 Shellcheck hook

todo: tohle je nejlepsi vec, kterou cele commash umi. tady by to mozna chtelo ukazat pripady, kdy se to hodi. taky popsat to, ze nektere veci jdou snadno zakazat

4.3.6 Bashlex hook

todo: tohle je bashovska cast debuggeru

4.3.7 Explain RC hook

4.3.8 Notfound hook

4.4 Bezpečný mód

Bezpečný mód umožňuje dvě základní věci. Tou jednodušší je pouze vypsání efektu příkazu, který má nějaké destruktivní následky. Složitější varianta dovoluje vracení do stavu před vykonáním příkazu.

todo: napsat o tom, ze po zkouseni jsem dosel k zaveru, ze je lepsi uzivateli nejprve ukazat co se deje, co se stane a jake ma moznosti. castokrat uz

4.4.1 Bezpečné rm

todo: napsat o tom, jak to je vyřešené a proč. napsat, ze rm -i i rm -I je k ničemu a proč je naše verze lepší popsat, jak funguje freedestkop.org trash a ze ukladame seznam souboru, ktere byly smazany jednim rm, aby sel cely spusteny prikaz rm vratit

4.5 Historie

Popsat jak jsem vyresil ukladani historie prikazu, ukladani vystupu, navratove kody, jak vracet nasledky prikazu do puvodniho stavu. todo: historie taky neni v zadani a moc jsem to nedoresil. teoreticky by na to sel napsat pre a post hook, ktery zaznamena vsechno potrebne a zapise to nekam do souboru. todo: slo by snadno udelat: podobna historie co je ted, jen se ulozi odkud se prikaz spustil

4.6 Instalace

todo: odkaz na github, vypsat sem to je i tam, tedy: tvorba virtualniho stroje a instalace popsat, ze se predopklada instalace do /.commash

4.7 Testování

Kapitola 5

Testování

5.0.1 Testování shell skriptů

Popsat jak funguji nektere testovaci frameworky.shunit2, roundup

5.1 Automatizované spouštění příkazů

Popsat jak se dají automaticky spouštět příkazy. At uz lokalne, nebo vzdalene. Popsat jak rekonstrukci z typescriptu, tak treba Tcl, Expect.

5.1.1 Expect

todo: jak se pracuje s expectem

Pro testování byl zvolen skriptovací jazyk Expect, který je rozšířením jazyka Tcl. Lze tedy velice snadno otestovat chování Comma-shellu a jeho částí pro různé vstupy.

Testy se skládají ze vstupů a z řetězců, které se očekávájí na výstupu. Přes Expect se potom vytvoří nová instance bashe, ve které se automaticky načte Comma-shell a postupně se posílají vstupy a sledují se výstupy pro očekáváné řetězce.

Testy jsou rozdělené na menší části, kde každá testuje jednu konkrétní věc. Přes skript lze pustit sérii všech testů.

5.1.2 Testování Debuggeru

5.1.3 Testování Hooks

5.1.4 Testování Bezpečných příkazů

5.1.5 Měření paměťových nároků

Závěr

todo: urcite bych napsal. ze nikdo nic podobneho neudelal a ohlasy byly celkem pozitivni. pokud se toho chytne alespon par lidi a budou hlasit chyby, co je potreba vylepsit, nebo dokonce i prispivat, mohl by z toho byt jednou opravdu vyborny pomocnik

sem napište závěr Vaší práce

Literatura

Příloha **A**

Seznam použitých zkratek

GUI Graphical user interface

XML Extensible markup language

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS