Jihočeská univerzita v Českých Budějovicích

Přírodovědecká fakulta

Sbírka řešených a neřešených úloh ze skriptování v Bashi pro výuku předmětu Operační systémy 2

Bakalářská práce

Erik Pach

Vedoucí práce: Mgr. Jiří Pech, Ph.D.

České Budějovice 2015

Jihočeská univerzita v Českých Budějovicích
Přírodovědecká fakulta

ZADÁVACÍ PROTOKOL BAKALÁŘSKÉ PRÁCE

Student: Erik Pach

(jméno, příjmení, tituly)

Obor – zaměření studia: 1801R001 / Aplikovaná informatika
.....

Školitel: .Mgr. Jiří Pech, Ph.D.

Garant z PřF: .....
(jméno, příjmení, tituly, katedra – jen v případě externího školitele)

Školitel – specialista, konzultant:
(jméno, příjmení, tituly, u externího š. název a adresa pracoviště, telefon, fax, e-mail)

Téma bakalářské práce:

Sbírka řešených a neřešených úloh ze skriptování v BASHi pro výuku předmětu Operační systémy 2

.....

Student by měl splnit následující cíle:

- Popsat možnost skriptování v UNIXu a porovnat jednotlivé možnosti.
- Udělat rešerši volně dostupných materiálů pro výuku skriptování v BASHi.
- Sestavit sbírku řešených i neřešených úloh na vytváření skriptů v BASHi o různém stupni obtížnosti.
- Tuto sbírku otestovat v praxi ve výuce předmětu Operační Systémy 2 ve spolupráci s vyučujícím předmětu.
- Sbírka bude publikována jako příloha bakalářské práce.

Základní doporučená literatura :

Financování práce :.....

Vedoucí práce Mgr. Jiří Pech, Ph.D. podpis : Pech

U externích vedoucích fakultní garant práce.....podpis : .....

Garant oboru bak.. studia (nepožaduje se u zaměření „příprava na mag. studium biologie) .....

podpis : .....

Vedoucí katedry: RNDr. Libor Dostálek podpis .....

Případný souhlas vedoucího ústavu AV .....podpis : .....

V Českých Budějovicích dne 18.10.2012

Převzal/a dne 18.10.2012 podpis : .....

Bibliografické údaje

Pach E., 2015: Sbírka řešených a neřešených úloh ze skriptování v Bashi pro výuku předmětu Operační systémy 2

[Collection of solved and unsolved Bash scripting examples for teaching the subject Operating Systems 2. Bc. Thesis, in Czech] – 78 p., Faculty of Science, The University of South Bohemia, České Budějovice, Czech Republic.

Abstrakt

Tato bakalářská práce se zabývá problematikou skriptování v Bashi s následným vytvořením sbírky úloh pro potřeby výuky v předmětu Operační systémy 2. Teoretická část se zabývá popisem obecných informací o skriptování, rozborem různých možností skriptování, samotným Bashem, použitým softwarem a analýzou informačních zdrojů použitých při vytváření sbírky úloh. Praktická část se pak zaměřuje na proces tvorby sbírky úloh, návodu pro její používání a především také na vyhodnocení použití této sbírky v praxi. Sbírka samotná je přiložena jako příloha této práce.

Abstract

This bachelor thesis deals with the problem of scripting in Bash and offers the tasks collection for teaching purposes in Operating Systems 2. The theoretical part consists of general information about scripting, analysis of different scripting options, Bash itself and the used software. The analysis of the information sources used in the process of making the tasks collection is included, too. The practical part focuses on the process of creation of the tasks collection, manual for its application and especially on evaluation of this collection in practical use. The collection itself is attached as an appendix of my work.

Prohlášení

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích, dne 24. 4. 2015

Podpis:.....

Poděkování

Chtěl bych poděkovat svému vedoucímu práce Mgr. Jiřímu Pechovi, Ph.D. a zároveň také panu Mgr. Miloši Prokýškovi, Ph.D. za čas a rady, které mi věnovali při tvorbě bakalářské práce. Dále bych chtěl poděkovat všem studentům absolvujícím předmět Operační systémy 2 v zimním semestru akademického roku 2013/2014 a 2014/2015 za jejich ochotu spolupracovat při vyplňování dotazníku. Na závěr také děkuji své rodině a nejbližším za podporu.

Obsah

1 Úvod

Tématem bakalářské práce je problematika skriptování v Bashi, která je součástí osnov předmětu Operační systémy 2, vyučovaném na Ústavu aplikované informatiky Přírodovědecké fakulty Jihočeské univerzity v Českých Budějovicích. Skriptování je stejně jako programování proces, který vede od počáteční formulace problému k vytvoření počítačového programu (v tomto případě skriptu). Účelem tohoto procesu je nalezení posloupnosti příkazů (algoritmů), které tento daný problém řeší. Bash je unixový shell. Jedná se o textové uživatelské rozhraní (také známé jako příkazový řádek), které předcházelo grafickému uživatelskému rozhraní a pomocí kterého může uživatel zadáváním textových příkazů ovládat počítač. Skriptování v Bashi pak znamená spojení těchto algoritmů a textových příkazů do souboru zvaného skript.

Cílem bakalářské práce je za pomoci analytických a srovnávacích metod provést rozbor dostupných informačních zdrojů a na jejich základě vytvořit jednotný výukový materiál (zvaný sbírka úloh) primárně určený pro potřeby výuky předmětu Operační systémy 2. Záměrem je poté nabídnout tento výukový materiál studentům předmětu a vyplnit tak jejich poptávku po dostupnějším výukovém materiálu, což by mělo mít za následek odbourání potřeby tyto informace vyhledávat samostatně, jejich nejednotnost a v závěru také k celkovému zefektivnění výuky.

Bakalářská práce je rozdělena do dvou hlavních částí, a to části teoretické a části praktické. V teoretické části se nachází základní informace o skriptování jako takovém s následným hlubším zaměřením na Bash. Poté je představen software potřebný pro práci se sbírkou úloh a rozbor dostupných informačních zdrojů použitých při vytváření sbírky. Praktická část práce se zabývá popisem tvorby sbírky úloh a návodem pro její použití. Na závěr následuje vyhodnocení dotazníkového šetření a analýza použití sbírky při studiu. Sbírka úloh je pak samostatnou přílohou bakalářské práce.

2 Cíle práce

Důvodem pro vznik této bakalářské práce a zároveň i jejím hlavním cílem je snaha o vytvoření uceleného výukového materiálu (sbírky úloh) pro potřeby výuky skriptování v Bashi, jež je součástí osnov předmětu Operační systémy 2.

Úkoly vedoucí k dosažení tohoto cíle jsou následující:

1) Vysvětlit základní pojmy jako jsou např. skript, skriptovací jazyk atd.
2) Stručně popsat a porovnat vybrané skriptovací jazyky. Poté se blíže zaměřit na samotný Bash.
3) Za pomoci analytických a srovnávacích metod porovnat vybrané zdroje informací na dané téma.
4) Na základě tohoto rozboru vytvořit strukturu a obsah sbírky úloh.
5) Vytvořit sbírku úloh s množinou řešených i neřešených příkladů.
6) Ve spolupráci s vedoucím předmětu otestovat využití sbírky v praxi.
7) Pomocí dotazníkového řešení provést vyhodnocení použití sbírky úloh v praxi.

3 Teoretická část

3.1 Obecné informace o skriptování

3.1.1 Skript

V oblasti informačních technologií je skript program, neboli také sekvence instrukcí, určený pro speciální prostředí (operační systém, webový prohlížeč, softwarové aplikace atd.), který má za úkol automatizovat provádění jednotlivých úkolů, které by jinak musel ručně provádět člověk. V unixových operačních systémech se nejčastěji jedná o textový soubor obsahující sérii příkazů, které jsou postupně spouštěny tzv. interpretem příkazů.

3.1.2 Skriptovací jazyk

Skriptovací, neboli také interpretovaný, jazyk je v podstatě programovací jazyk určený k vytváření skriptů. Existují jazyky, které byly koncipovány výslovně jako skriptovací. Mezi ty nejznámější patří např. Perl, Python, PHP, JavaScript a také Bash, na který je zaměřena tato bakalářská práce a posléze také sbírka úloh.

Hlavním rozdílem oproti kompilovanému jazyku (druhá hlavní skupina jazyků) je fakt, že skriptovací jazyk nevyžaduje tzv. kompilaci a pro spuštění skriptu je zapotřebí jeho zdrojový kód a speciální program zvaný interpret, který tento zdrojový kód provádí (interpretuje).

Protikladem této skupiny jsou jazyky kompilované. U těchto jazyků je nutný proces kompilace. Jedná se o proces, při kterém se zdrojový kód napsaný v daném programovacím jazyce převede do jazyka jiného (ve většině případů do jazyka binárního) a vznikne tak spustitelný program. Výhoda kompilace spočívá v rychlosti. Zkompilovaný program může po zavedení do paměti přímo zpracovávat procesor počítače, kdežto nezkompilovaný skript musí před procesorem zpracovat jiný program. Nevýhodou tohoto přístupu však je, že u jazyků, které vyžadují kompilaci,

musí být zdrojový kód před každým spuštěním zkompilován, což nějaký čas trvá. Mezi tuto skupinu jazyků patří např. C, C++ a Java.

3.1.3 Srovnání skriptovacích jazyků

Tato kapitola je zaměřena na srovnání vybraných skriptovacích jazyků. Mezi tyto jazyky patří Perl, Python, Ruby, PHP, JavaScript a Bash. Prvních pět jazyků bylo vybráno na základě jejich umístění v žebříčku TOIBE[1] srovnávajícího popularitu jednotlivých programovacích jazyků. Bash byl do srovnání zařazen, protože je na něj zaměřena tato bakalářská práce.

U každého jazyka je uvedena oblast jeho použití. Mezi tyto oblasti patří např. softwarové aplikace, webové aplikace a jednoduché skripty určené pro použití v serverovém prostředí. Dále je uveden výčet výhod a nevýhod daného jazyka.

Perl

Perl je skriptovací jazyk vytvořený Larry Wallem roku 1987. Je založený na jazycích jako jsou např. C, AWK a SED. Používá se pro vytváření softwarových aplikací používaných např. v oblasti bioinformatiky, finančního sektoru a počítačových sítí. Svou popularitu získal především díky podpoře pro vytváření tzv. CGI skriptů, což jsou spustitelné soubory umístěné na webovém serveru sloužící k předávání obsahu webové stránky. Mezi jeho silné stránky patří zejména složité zpracování textu. Nevýhodami tohoto jazyka jsou pak např. složitost syntaxe, spotřeba paměti a v porovnání s ostatními jazyky také použitelnost.

Python

Python je dynamický, objektově-orientovaný jazyk, který v roce 1991 vytvořil Guido van Rossum. Používá se pro vývoj jak softwarových, tak i webových aplikací. Mezi nejznámější firmy používající tento jazyk patří např. Yahoo, Dropbox a NASA. K jeho přednostem patří zejména čitelnost syntaxe, která je sice velice striktní, ale právě díky tomu napomáhá k přehlednosti. Je tedy velmi jednoduchý z hlediska

1 Dostupné z http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html.

učení, což vede k větší produktivitě programátora. Nevýhodami jsou pak menší rychlost, náročnost na zdroje paměti a procesoru a v porovnání s ostatními jazyky i nepříliš rozsáhlá dokumentace.

Ruby

Ruby je stejně jako Python dynamický, objektově-orientovaný a víceúčelový jazyk. Byl navržen a vyvinut v roce 1995 a jeho tvůrcem je Yukihiro Matsumoto. Je založen na základech jazyků Perl, Smalltalk a Lisp. Rozsah jeho použití je velmi široký a to od vytváření jednoduchých unixových skriptů až po robustní webové aplikace. Obliba tohoto jazyka vzrostla díky frameworku Ruby on Rails, který slouží právě k vývoji webových aplikací. Výhodami tohoto jazyka jsou pokročilá práce s regulárními výrazy, velký počet rozšíření a knihoven a také jednoduchost syntaxe, která vede k rychlejšímu učení. Mezi nevýhody se pak řadí např. nižší rychlost, a opět jako i v př,ípadě, Pythonu i nedostatek kvalitně zpracované dokumentace.

PHP

PHP je populární skriptovací jazyk, který v roce 1994 vytvořil Rasmus Lerdorf. Je určený především pro vytváření dynamických webových stránek a webových aplikací. Lze jej však použít i k tvorbě aplikací konzolových. Syntaxe tohoto jazyka je založena na jazycích Perl, C a také Java. PHP využívají i ty největší internetové projekty mezi něž patří např. Facebook, Wikipedia a WordPress. Mezi hlavní výhody PHP patří dobrá práce s databázovými systémy, souborovými systémy, široká komunita uživatelů a jednoduchost. Nevýhodami jsou pak např. pomalejší běh aplikací, zpracování chyb, nejednotnost pojmenování některých funkcí a bezpečnost.

JavaScript

JavaScript je skriptovací jazyk, jehož autorem je Brendan Eich a který vznikl v roce 1995. Využívá se především v prostředí webových stránek a aplikací, kde má zpravidla za úkol manipulaci s interaktivními prvky a strukturou stránky. Skript napsaný v JavaScriptu se spouští až na klientském počítači po načtení stránky do

prohlížeče. Od příchodu aplikací jako je např. Rhino a Node.js je však možné JavaScript využívat i na serveru. Mezi další použití tohoto jazyka pak patří také vytváření her a mobilních aplikací. Hlavní výhodou JavaScriptu je v případě využití u klienta menší zátěž na server. V případě serverového využití je to pak zejména rychlost zpracování požadavku. Mezi nevýhody se řadí přístup a manipulace se soubory, rozdílné zpracování a celková nekompatibilita mezi prohlížeči a také bezpečnost.

Bash

Bash je interpret příkazů, který byl vytvořen v roce 1989 a jehož autorem je Brian Fox. Kombinuje nejlepší vlastnosti ostatních unixových shellů. Využívá se především pro vytváření interaktivních skriptů pro potřeby příkazové řádky. Mezi jeho výhody patří jednoduchost použití a záruka funkčnosti na všech unixových systémech. Nevýhodu je poté zbytečné vytváření procesů pro každý použitý shell příkaz.

Jazyk	Použití	Výhody	Nevýhody
Perl	softwarové aplikace, skripty	zpracování složitých textů	složitá syntaxe, větší spotřeba paměti
Python	softwarové aplikace, webové aplikace	jednoduchá syntaxe, rychlost učení	menší rychlost, náročnost na zdroje, dokumentace
Ruby	softwarové aplikace, webové aplikace, skripty	jednoduchá syntaxe, velké množství rozšíření a knihoven	nižší rychlost, dokumentace
PHP	webové aplikace, skripty	jednoduchost, široká komunita uživatelů a použití	pomalejší běh, zpracování chyb, bezpečnost

Jazyk	Použití	Výhody	Nevýhody
JavaScript	webové aplikace, mobilní aplikace	menší zátěž na server, rychlost	nekompatibilita mezi prohlížeči, bezpečnost
Bash	skripty	jednoduchost, kombinace výhod jiných shellů	zbytečné vytváření procesů

Tabulka 1: Porovnání vybraných skriptovacích jazyků

3.2 Bash (Bourne Again Shell)

Bash je příkazový interpret, který byl vytvořen v roce 1989 primárně pro operační systémy GNU. Jednalo se o nástupce příkazového interpretu Bourne Shell (sh), ale přejímal i některé vlastnosti (včetně syntaxe) z několika předchozích příkazových interpretů (např. Korn Shell nebo C Shell). Jeho tvůrcem je Brian Fox, který v té době pracoval po boku Richarda Stallmana pro organizaci Free Software Foundation. V dnešní době je to jeden z nejčastěji používaných příkazových interpretů pro operační systém Linux. Pro mnohé linuxové distribuce je to také zároveň příkazový interpret výchozí. Oproti předchozím nabízí mnohá vylepšení, a to jak v oblasti programování, tak i v interaktivním režimu.

Jak již bylo zmíněno, Bash je příkazový interpret určený primárně pro linuxové operační systémy, avšak jeho použití je možné i na jiných operačních systémech. První skupinou takovýchto systémů jsou unixové systémy. Mezi ně patří například OS X od firmy Apple nebo BSD. Druhou skupinou jsou operační systémy založené na jiném nežli unixovém základě. Asi nejznámějším zástupcem této skupiny je operační systém Windows od firmy Microsoft. Bash je možné na tomto druhu systémů používat pomocí různých emulátorů speciálně určených pro tento účel a dalších jiných subsystémů. Asi nejznámějším emulátorem pro systém Windows je program Cygwin.

3.3 Použitý software

Tato kapitola obsahuje stručný popis softwaru použitého při vytváření sbírky úloh. Ačkoliv je vytváření a spouštění Bash skriptů možné na vícero platformách (Windows, OS X atd.), byl pro účely této bakalářské práce a sbírky úloh vybrán linuxový operační systém Ubuntu 14.04. Hlavním důvodem tohoto výběru je fakt, že Bash byl primárně vytvořen pro unixové systémy. Jako textový editor pro vytváření a editaci zdrojového kódu byl zvolen program Nano, který je standardní součástí distribuce Ubuntu. Všechny skripty obsažené ve sbírce úloh byly vypracovány pomocí tohoto textového editoru a to ve verzi 2.2.6.

3.3.1 Linuxová distribuce Ubuntu

Linuxová distribuce je označení pro operační systém založený na linuxovém jádře, které je základní součástí každé distribuce. Distribuce jsou vytvářeny proto, aby si uživatel nemusel jádro a další doplňující software instalovat složitým způsobem sám.

Ubuntu [9] je jednou z nejznámějších a nejoblíbenějších linuxových distribucí dneška. Je vhodná pro použití na laptopech, stolních počítačích i serverech. Tento operační systém založený na distribuci Debian je pod záštitou britské společnosti Canonical vyvíjen vývojáři z celého světa jako volně šiřitelný software.

Mezi jeho přednosti patří obrovská základna uživatelů, kteří se aktivně podílejí na vývoji a jsou mimo jiné velmi ochotni poradit ostatním uživatelům tohoto systému. Cílem této distribuce je zpřístupnit tento systém tolika uživatelům, jak je to jen možné.

3.3.2 Textový editor Nano

Nano [10] je jednoduchý textový editor používaný primárně v unixových operačních systémech používajících prostředí příkazové řádky. Tento editor vychází z editoru Pico, který byl součástí emailového klienta Pine, kde sloužil jako nástroj pro vytváření a modifikaci emailových zpráv. V porovnání s nástrojem Pico je ale

editor Nano uvolněn pod licencí GNU General Public Licence a jedná se tedy o svobodný a volně dostupný software.

Mezi výhody tohoto editoru patří zejména široká podpora klávesových zkratek, automatické doplnění názvu souborů a zvýraznění syntaxe na základě daného obsahu. Asi největší nevýhodou je pak jeho přílišná jednoduchost a omezená funkcionalita, která ale plně postačuje pro účely skriptování v Bashi.

3.4 Rozbor informačních zdrojů

Při analýze a následném vytváření této sbírky úloh bylo vycházeno ze čtyř různých informačních zdrojů a doporučení.

Tím prvním a zároveň nejdůležitějším zdrojem byl obsah předmětu Operační systémy 2[2], který prakticky určoval rozsah teoretických informací, které musí být součástí sbírky tak, aby byl student schopen pomocí tohoto učebního materiálu úspěšně absolvovat závěrečný test z této oblasti.

Druhým zdrojem informací byly již hotové a dlouhodobě používané materiály dostupné z přednášek a cvičení.

Třetím zdrojem se stala knižní literatura, a to jak ve fyzické, tak i elektronické podobě. Tato literatura byla vyhledána v internetovém knihkupectví Amazon[3], které je jedním z nejstarších a největších online knihkupectví na světě. Celkově byly na základě uživatelského hodnocení[4] vybrány čtyři knihy, které se v žebříčku nacházely mezi deseti nejlépe hodnocenými.

Čtvrtým a posledním zdrojem informací byly informace dostupné na internetu. Jednalo se především o několikadílné seriály a návody věnující se skriptování v Bashi. Tyto internetové zdroje byly vybrány na základě výsledku vyhledávání

2 Jedná se pouze o obsah té části kurzu, která se věnuje skriptování v Bashi.

3 Dostupné z http://www.amazon.com/.

4 Výsledky uživatelského hodnocení platné v Prosinci roku 2013.

portálu Google a také díky své přítomnosti mezi doporučenými zdroji informací uvedenými na portále Wikipedia[5].

Na základě obsahové analýzy těchto informačních zdrojů byla sestavena ideální osnova sbírky úloh, která zahrnuje veškeré teoretické informace potřebné pro výuku předmětu Operační systémy 2. Hlubší analýzou byla následně vytvořena struktura každé kapitoly sbírky úloh.

Nyní bude následovat výčet a srovnání několika knih a internetových zdrojů, z kterých bylo čerpáno. Tyto zdroje byly srovnávány z hlediska obsahu, struktury a také jejich náročnosti.

3.4.1 Knižní vydání

V prvním čtvrtletí roku 2015, kdy tato práce vznikala, neexistovala kniha psaná v českém jazyce věnující se přímo skriptování v Bashi. K dispozici byly pouze knihy, jejichž hlavním zaměřením byl operační systém Linux a které se skriptování v Bashi věnovaly jenom okrajově. Při zkoumání dostupné literatury bylo tedy čerpáno hlavně z anglické literatury, která je v tomto ohledu bohatší.

Linux shell scripting with Bash / Ken O. Burtch

Kniha je poměrně rozsáhlá a obsahuje velké množství převážně pokročilých technik a informací, mezi něž patří například rozšířená práce se soubory, databází, sítí a tak dále. Prvních pár kapitol se však zabývá základy skriptování v Bashi jako jsou proměnné, řídící struktury a další a je tím pádem vhodná i pro začínající jedince.

Obsahem každé kapitoly této knihy je krátké seznámení čtenáře s teorií, kterou se bude daná kapitola zabývat. Následuje samotný obsah, který se skládá z vysvětlení problematiky a spousty ukázkových příkladů přičemž každý z nich je doplněn podrobným slovním vysvětlením, což je jedna z neužitečnějších částí knihy.

Velmi užitečná je rovněž závěrečná sekce každé kapitoly, která uvádí seznam příkazů, které jsou v dané kapitole použity, a to i s výčtem těch nejdůležitějších

5 Dostupné z http://cs.wikipedia.org/wiki/Bash.

přepínačů. Mezi nevýhody této knihy se pak řadí náročnost textu, který je místy příliš rozsáhlý a stává se tak nesrozumitelným.

Bash Guide for Beginners / Machtelt Garrels

Druhá kniha nese název „Příručka Bashem pro začátečníky“. Z názvu je patrné, že by se mělo jednat o materiál určený pro začínající uživatele bez větších předchozích znalostí. A název skutečně nelže. Tato publikace se vskutku zaměřuje na úplné základy skriptování v Bashi. Je to dáno především tím, že na tuto knihu navazuje další díl s názvem „Příručka Bashem pro pokročilé“, který je podrobněji popsán dále.

Obsah knihy se skládá z úvodu do skriptování, samotného Bashe a unixového prostředí. Následují kapitoly věnující se proměnným, podmínkám, cyklům a funkcím, což jsou základní stavební kameny. Mezi další kapitoly patří například úvod do regulárních výrazů a popis práce s textovými nástroji sed a awk.

Každá kapitola se skládá ze stručného slovního úvodu jež je následován opět velmi stručným popisem teorie a několika málo příklady. Mezi největší přednosti tohoto studijního materiálu se řadí např. stručnost, jednoduchost a zároveň fakt, že tato kniha jako jediná obsahuje soubor několika neřešených příkladů, které jsou přiloženy na konci vybraných kapitol.

Advanced Bash Scripting Guide / Mendel Cooper

„Příručka Bashem pro pokročilé“ je pokračováním předešlé publikace. Jedná se již o věci opravdu pokročilé a značně mimo rámec kurzu Operační systémy 2. Mimo tyto rozšiřující informace, mezi které patří kupříkladu testování, správa procesů a paměti, ale kniha obsahuje i zopakování některých základů. Tyto základní informace jsou popsány lehce odlišným způsobem, nežli v předchozí knize pro začátečníky a lze z nich tedy pochopit další souvislosti. Kniha je mimo této skutečnosti vhodná také ke zkoumání složitějších příkladů, které ale obsahují základní syntaxi jazyka. Kromě těchto výhod je vše prakticky totožné s příručkou pro začátečníky. Tato kniha je tedy vhodná pouze jako doplněk ke studiu.

Learning the Bash Shell / Cameron Newham

Poslední z používaných knih je publikace s názvem „Učíme se Bash Shell“. Jedná se o zdroj informací vhodný jak pro začátečníky, tak i středně pokročilé. První kapitola se kromě samotného Bashe věnuje i shellu, což je užitečné. Základními kapitolami pro začátečníky jsou opět kapitoly věnující se proměnným, řídícím strukturám a funkcím. Mezi ty složitější témata poté patří například ošetření chyb, testování a řízení procesů.

Tato kniha obsahuje asi nejvíce pozitivních vlastností. Nechybí zde srozumitelně a jednoduše vysvětlená teorie doplněna ukázkami příkladů společně i s jejich popisem, souhrn použitých příkazů a také závěrečná sekce, která shrnuje obsah dané kapitoly. Jediným nedostatkem je nepřítomnost neřešených příkladů, jež ale postrádá většina publikací.

3.4.2 Internetové zdroje

Seriál o Bashi / Bohdan Milar

Jedná se o podobný internetový seriál jako v případě prvního zmíněného. Tento je však z roku 2007 a byl napsán Bohdanem Milarem. Oproti předešlému seriálu je tento poněkud rozšířenější a obsahuje 25 dílů. V úvodu jsou standardně popsány základy práce s příkazovou řádkou, proměnné a procesy. Následují konstrukce for, if else, while, case, select a na závěr některé složitější konstrukce a problémy. Seriál je napsán přehledně, stručně a obsahuje spoustu ukázkových příkladů. Tento seriál lze doporučit jak začínajícím, tak i mírně pokročilým uživatelům. Jeho další výhodou je dostupnost celého seriálu v podobě PDF souboru, který si lze stáhnout a přistupovat tak k informacím i offline, popřípadě si celý dokument snadno vytisknout.

Seriál BASH / Jan Fuchs

Tento internetový seriál je takřka jedním z prvních návodů týkající se programování v Bashi u nás v České republice. Napsal ho Jan Fuchs na konci roku

2003 a byl publikován na serveru ABC Linuxu, jež je dodnes jedním z předních a největších webových portálů zabývající se Linuxem v naší zemi. Tento seriál se skládá ze šesti po sobě jdoucích dílů. V prvních dvou dílech je čtenář seznámen s příkazovou řádkou a se základními linuxovými příkazy. Následují kapitoly pojednávající o proměnných, podmínkách, cyklech a funkcích. Na závěr se seriál věnuje regulárním výrazům a pokročilejším technikám pro ladění skriptů. Mezi hlavní užitečné prvky struktury patří jednoduché a srozumitelné příklady a také seznamy důležitých příkazů. Dalo by se říci, že tento seriál je po tolika letech již poněkud zastaralý, ale opak je pravdou. Základní syntaxe jazyka se za ta léta nezměnila a nejspíše ani měnit nebude.

Název	**Obsah**	**Užitečné prvky struktury**	**Náročnost**
Linux shell scripting with Bash	proměnné, podmínky, cykly, funkce, práce se soubory, databáze	úvod do kapitoly, ukázkové příklady i s vysvětlením, seznamy příkazů	začátečník, středně pokročilý
Bash Guide for Beginners	úvod do skriptování, proměnné, cykly, podmínky, funkce, regulární výrazy	srozumitelná teorie, neřešené příklady	začátečník
Advanced Bash Scripting Guide	proměnné, cykly, podmínky, procesy, testování, regulární výrazy, práce se sítí a pamětí	rozsáhle množství řešených příkladů	pokročilý
Learning the Bash Shell	základní informace, proměnné, podmínky, cykly, procesy, ostatní typy shellu, odstraňování chyb	srozumitelná teorie, řešené příklady i s vysvětlením, shrnutí kapitoly	začátečník, středně pokročilý

Název	Obsah	Užitečné prvky struktury	Náročnost
Seriál o Bashi	řízení procesů, proměnné, podmínky, cykly, funkce, práce s textovými řetězci, konfigurační soubory	úvod do kapitoly, rozsáhlý slovní popis teoretických informací, shrnutí kapitoly	začátečník, mírně pokročilý
Seriál BASH	základní příkazy, proměnné, podmínky, cykly, funkce, komentáře, regulární výrazy, zpracování signálů	jednoduché příklady, ukázky kódu, seznamy příkazů	začátečník, mírně pokročilý

Tabulka 2: Porovnání vybraných informačních zdrojů

4 Praktická část

4.1 Tvorba sbírky úloh

4.1.1 Sestavení obsahu

Při tvorbě sbírky úloh bylo prvním důležitým krokem vymezení jejího teoretického obsahu, neboli kapitol. Primárním zdrojem informací pro tento úkol byl sylabus předmětu Operační systémy 2, který určuje rozsah informací vyučovaných v tomto kurzu. Dalším krokem bylo vyhledání potřebné literatury a dalších zdrojů, které byly přečteny a prostudovány[6]. Sloučením všech těchto poznatků byl vypracován ideální obsah sbírky úloh. Tento obsah se skládá z:

1) Úvodních informací – shell, Bash, vytváření a spouštění skriptu, parametry...
2) Proměnné a prostředí – typy proměnných, konstanty...
3) Řízení toku programu – podmínky, řídící struktury, logické operátory...
4) Cykly – základní informace, konstrukce...
5) Funkce – deklarace, volání, parametry, návratová hodnota...

4.1.2 Sestavení struktury

Po vytvoření obsahu přišlo na řadu sestavení struktury, neboli toho, jaké sekce a informace má každá kapitola sbírky obsahovat. Zde bylo opět zapotřebí sloučit informace načerpané při analýze informačních zdrojů a na jejich základě vytvořit ideální řešení. Struktura jedné kapitoly sbírky úloh je následující:

1) Úvod – seznámení s problematikou dané kapitoly
2) Teorie – teoretické informace doplněné o jednoduché ukázky

6 Výsledky tohoto průzkumu jsou obsaženy v kapitole 3.4.

3) Souhrn – závěrečné shrnutí dané kapitoly
4) Řešené úlohy – slovně zadané úlohy a jejich následné řešení doplněné o slovní vysvětlení

Na závěr celé sbírky byly umístěny dvě další kapitoly, které nejsou součástí teoretického obsahu. Tou první z nich je soubor slovně zadaných úkolů k procvičení avšak bez poskytnutého řešení, neboli kolekce neřešených úloh. Tou druhou a zároveň i poslední kapitolou sbírky jsou přílohy, jejichž obsahem jsou přehledy základních příkazů, proměnných atd.

4.1.3 Vytvoření úloh

Řešené i neřešené úlohy byly postupně čerpány a přidávány ze všech informačních zdrojů použitých při tvorbě sbírky. Jelikož však téměř žádný z materiálů neobsahoval srozumitelné slovní vysvětlení dané úlohy, bylo v případě řešených úloh nutné provést jejich vyhodnocení a následně toto srozumitelné vysvětlení vytvořit.

4.2 Návod pro práci se sbírkou úloh

Sbírka úloh není vázána pouze k jednomu danému vyučovanému bloku. Přestože je primárně určena jako doplněk ke cvičením, je možné ji využívat i v přednáškách a také při přípravě na závěrečnou zkoušku.

Celkový obsah sbírky a její kapitoly jsou tvořeny tak, aby bylo možné každou kapitolu studovat samostatně a nezávisle na těch ostatních. Pokud tedy někdo již disponuje základními znalostmi a chce se dozvědět pouze to, co ho zajímá, může bez obav úvodní kapitoly sbírky přeskočit. V případě začátečníků je však doporučeno sbírku procházet postupně tak, jak je navržena. Co se týče samotného obsahu jednotlivých kapitol, zde je již doporučené jej procházet v daném pořadí, neboli seznámení se s problematikou, teorie a příklady. Neřešené úlohy lze procvičovat jak v průběhu studia při dokončení dané kapitoly a nebo až nakonec po dokončení všech kapitol sbírky.

4.3 Analýza sbírky úloh

4.3.1 Předchozí testování

V zimním semestru akademického roku 2013/2014 proběhlo testování první verze sbírky úloh. Tato verze obsahovala určité nedostatky, které již byly v současné verzi odstraněny. Mezi tyto nedostatky patřilo např. nedostatečně přesné slovní zadání některých úloh, nadbytečné informace týkající se základních informací o Linuxu a jeho příkazech a také nejasná struktura celé sbírky. Přes všechny tyto nedostatky však vyhodnocení dotazníkového šetření proběhlo bez problémů a výsledky byly pozitivní a uspokojivé.

4.3.2 Dotazníkové šetření

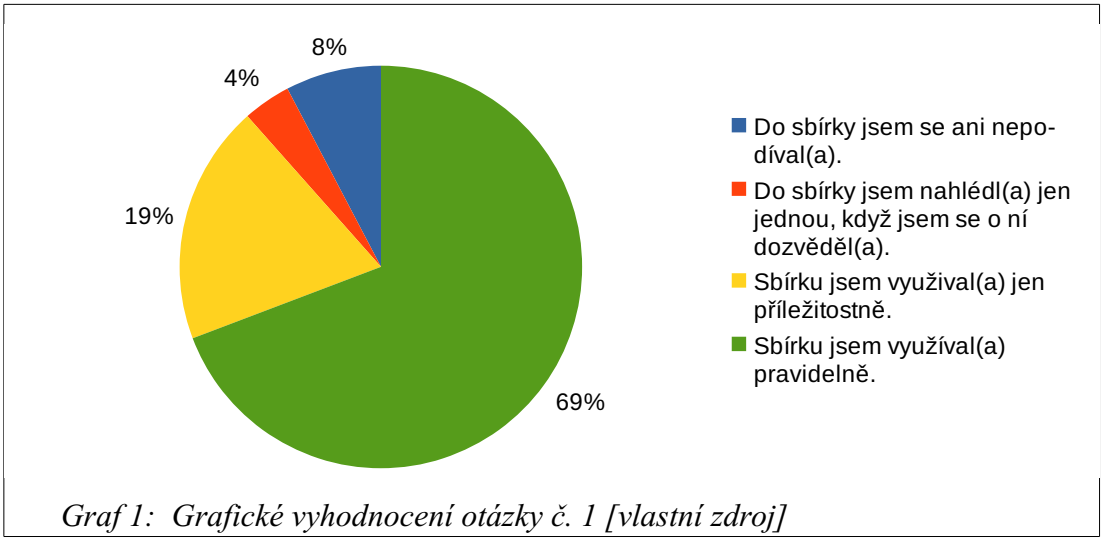
Dotazníkové šetření této nové a opravené verze sbírky úloh probíhalo v zimním semestru akademického roku 2014/2015 na Ústavu aplikované informatiky Jihočeské univerzity v Českých Budějovicích. Nově zpracovaný dotazník, který se stejně jako ten předešlý skládal celkem z deseti otázek, byl ke konci semestru předložen studentům předmětu Operační systémy 2. Vyplnění dotazníku nebylo stejně jako při prvním testování povinné, avšak počet odpovědí byl velmi uspokojivý. Z celkového počtu 33 studentů, kteří tento předmět v daném semestru studovali, jej vyplnilo celkem 29, což činí celkem 87,9 %. To je o celých 41 % více než v předešlém akademickém roce, kdy dotazník vyplnilo pouze 46,9 % dotázaných studentů. Dotazník je přílohou této bakalářské práce.

4.3.3 Vyhodnocení dotazníku

1) Jak často jste sbírku úloh používal(a)?

První otázka měla za úkol odhalit, zdali studenti sbírku úloh vůbec používali a pokud ano, tak do jaké míry. Jedná se o uzavřenou otázku s výběrem čtyř před připravených možností vyjadřujících četnost použití.

Pouze dva studenti uvedli, že se do sbírky ani nepodívali, z čehož vyplývá, že zbylých 27 studentů sbírku nějakým způsobem využívalo. Celkem čtyři studenti uvedli, že se do sbírky minimálně podívali. Ostatní naprostá většina uvedla, že se sbírkou pracovala pravidelně. Z této otázky tedy plyne, že poskytnutá sbírka úloh se stala přinejmenším jedním z hlavních učebních materiálů.

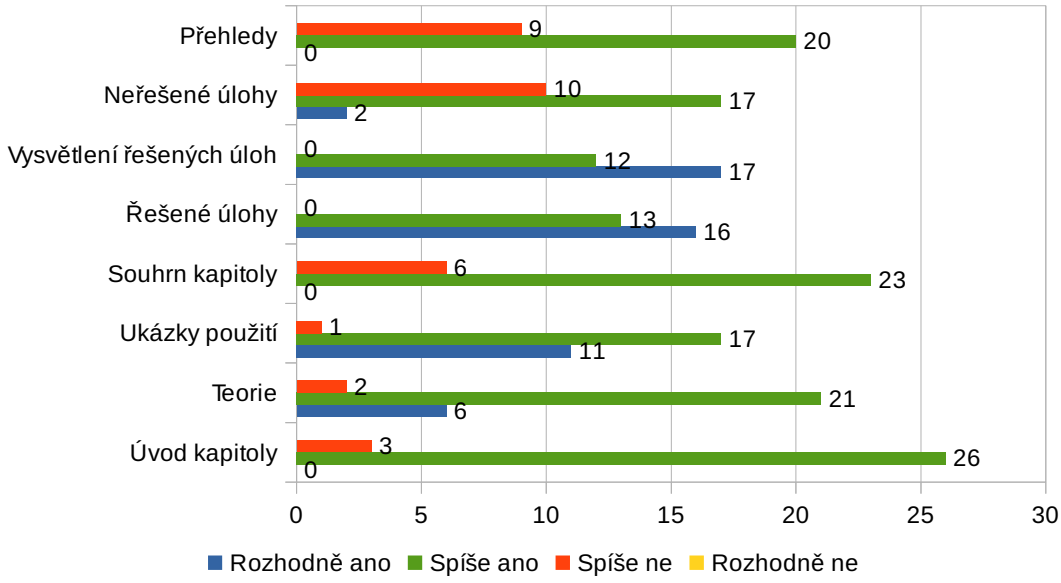


Graf 1: Grafické vyhodnocení otázky č. 1 [vlastní zdroj]

2) Jaké části sbírky považujete za užitečné?

Druhá otázka byla zaměřena na jednotlivé součásti sbírky. Tato otázka byla sestrojena pomocí matice otázek, které vyjadřovali užitečnost jednotlivých částí sbírky úloh, které byly sestaveny na základě rozboru informačních zdrojů. Na měření užitečnosti jednotlivých částí byla použita likertova škála.

Podle očekávání se jako nejužitečnější součástí celé sbírky projevily již vyřešené úlohy a jejich slovní vysvětlení. Mezi ty méně důležité pak patří přehledy příkazů a jiných tabulek, souhrn kapitoly a překvapivě také neřešené úlohy, z čehož vyplývá, že studentům k pochopení dané látky buďto stačily již vyřešené úlohy, nebo se jim neřešené úlohy nechtělo řešit.

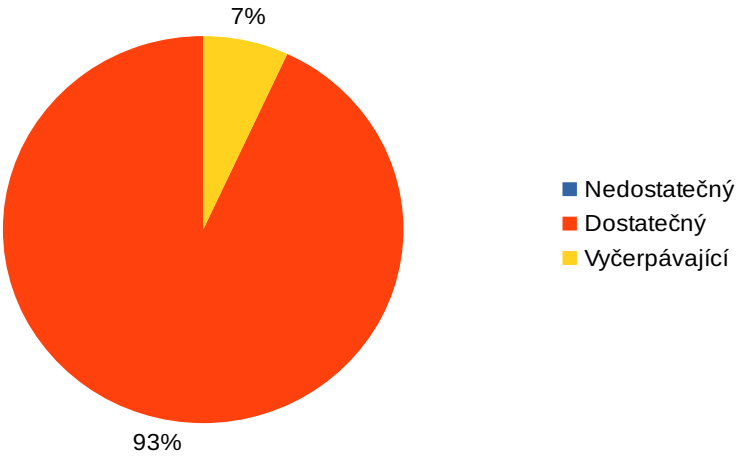


Graf 2: Grafické vyhodnocení otázky č. 2 [vlastní zdroj]

3) Jak hodnotíte množství teoretických informací obsažených ve sbírce úloh?

Úkolem této uzavřené otázky nabízející tři možnosti odpovědi bylo ověření zpracování teoretické částí sbírky. Každá kapitola sbírky obsahuje svoji teoretickou část, která má za úkol seznámit čtenáře s daným problémem (teorií) a to i za pomoci názorných ukázek. Každý student měl vyjádřit svůj názor na to, zdali se mu množství teorie zdálo nedostačující, dostatečné a nebo až vyčerpávající, což v podstatě znamená příliš rozsáhlé.

Výsledek je jednoznačný. Ani jeden student neuvedl, že by se mu množství teorie zdálo nedostačující a naopak drtivá většina odpověděla kladně a to tak, že množství teorie se jim zdálo dostatečné. Výsledkem zkoumání těchto odpovědí je tedy fakt, že teoretická část sbírky úloh byla optimalizována natolik, že vyhovuje nárokům většiny.



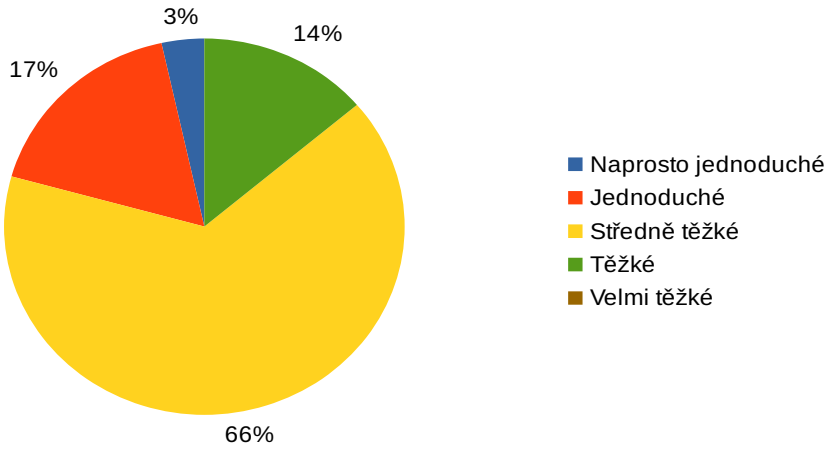
Graf 3: Grafické vyhodnocení otázky č. 3 [vlastní zdroj]

4) Jak moc obtížené pro Vás bylo vypracování řešených úloh?

Následující tři otázky zkoumají úlohy obsažené ve sbírce. První z těchto otázek se zabývá řešenými úlohami a to konkrétně jejich obtížností. Jedná se opět o uzavřenou otázku s výběrem pěti možných odpovědí vyjadřujících stupeň obtížnosti vypracování řešených úloh.

Jelikož z otázky číslo dvě vyplynulo, že tyto řešené úlohy a jejich vysvětlení jsou prakticky nejdůležitější součástí sbírky, tak byl výsledek této otázky pro posouzení celkové prospěšnosti sbírky velice důležitý, ne-li až klíčový.

Naštěstí ve prospěch sbírky je výsledkem této otázky fakt, že dvě třetiny všech studentů označily řešené úlohy jako středně těžké. To znamená, že tyto úlohy jsou sestaveny tak, aby nebyly příliš jednoduché ani složité a v podstatě tak vyhovovaly širokému spektru studentů. Protože však ne každý student je stejně nadaný, je zde i zbývající jedna třetina. Z této skupiny pouze jeden student označil tyto úlohy za příliš jednoduché, pět studentů je považuje za jednoduché a čtyři za těžké. Výsledkem je tedy zjištění, že řešené úlohy jsou sestaveny dobře a vyhovují valné většině.



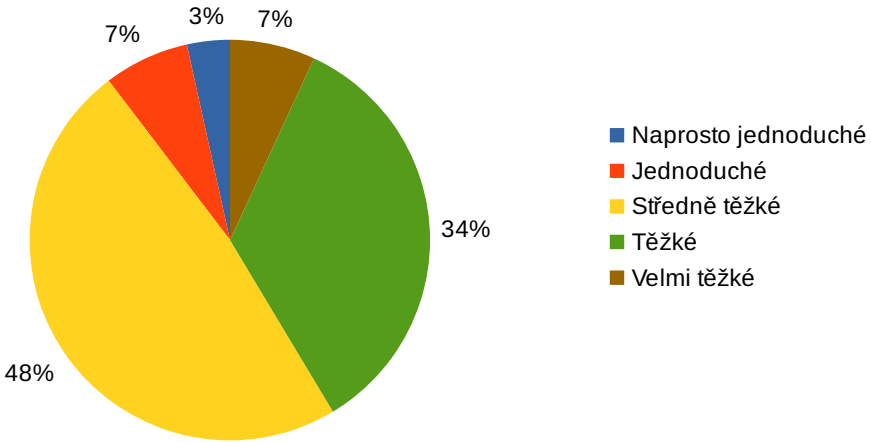
Graf 4: Grafické vyhodnocení otázky č. 4 [vlastní zdroj]

5) Jak moc obtížené pro Vás bylo vypracování neřešených úloh?

Druhá otázka týkající se úloh byla zaměřena na obtížnost neřešených úloh jež jsou umístěny na samotném konci sbírky a které slouží pro ověření nabytých znalostí z předešlých kapitol sbírky. Typ této otázky i odpovědí je stejný jako v otázce předešlé s tím rozdílem, že tentokrát se jedná o úlohy neřešené.

Zde je opět lehce splněn předpoklad vyplývající z otázky číslo dvě a to ten, že studenti neřešeným úlohám nepřikládali takovou váhu. Z grafu vyplývá, že dohromady 41 % všech dotazovaných připadaly tyto úlohy těžké nebo dokonce až velmi těžké. Složitost těchto neřešených úloh byla tudíž nejspíše rozhodujícím faktorem pro vynechání této části a spokojení se jenom s předešlými kapitolami sbírky.

Na druhou stranu počet studentů, který považuje neřešené úlohy za středně těžké nebo jednoduché, je více jak jedna polovina, což vede k závěru, že by bylo zapotřebí některé úlohy modifikovat tak, aby nebyly příliš těžké a zároveň aplikovat způsob, jak přilákat studenty k řešení těchto úloh.

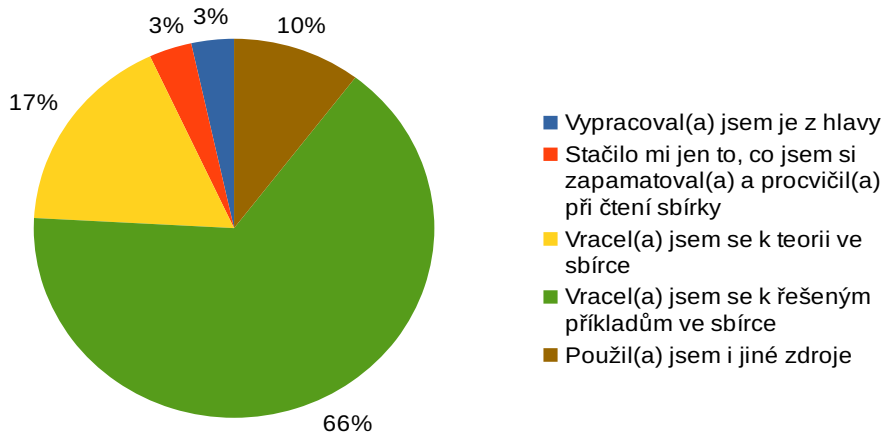


Graf 5: Grafické vyhodnocení otázky č. 5 [vlastní zdroj]

6) Jak jste postupoval(a) při vypracování neřešených úloh?

Poslední otázka zabývající se úlohami souvisela opět s neřešenými úlohami. Jednalo se o polouzavřenou otázku, která obsahovala čtyři konkrétní a jednu textovou odpověď. Jejím úkolem bylo zjistit, jak studenti postupovali při vypracování neřešených úloh.

Zde se opět potvrdil závěr z otázky číslo dvě, která zkoumala důležitost jednotlivých částí sbírky a to konkrétně ten, že již vyřešené úlohy a návod na jejich řešení jsou nejdůležitější součástí sbírky. Celé dvě třetiny studentů totiž uvedly, že se při zpracovávání neřešených úloh často vracely právě k těm již vypracovaným. Zhruba jedna čtvrtina poté uvedla, že se mimo jiné musela vracet i k teoretické části sbírky. Celých 10 % také uvedlo, že při vypracování neřešených úloh čerpalo i z jiných zdrojů. Mezi tyto zdroje patřily především internetové zdroje, a to konkrétně přesně ty, které jsou uvedeny v rozboru informačních zdrojů. Jedná se především o seriál „Skriptování v Bashi“ od Bohdana Milara. V tomto směru by tedy bylo užitečné optimalizovat neřešené úlohy natolik, aby student při jejich řešení vůbec nemusel používat jiné zdroje a vystačil si tak pouze s touto sbírkou úloh.

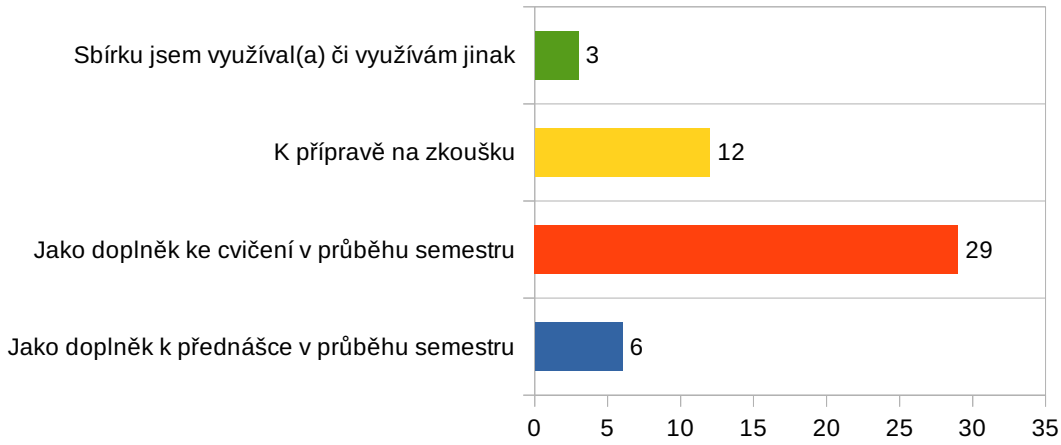


Graf 6: Grafické vyhodnocení otázky č. 6 [vlastní zdroj]

7) Jakým způsobem jste sbírku úloh využíval(a)?

Sedmá otázka se zabývá využitím sbírky v různých částech celého kurzu Operační systémy 2. Jedná se o polouzavřenou otázku s výběrem tří konkrétních a jedné textové odpovědi přičemž je možné zvolit více odpovědí.

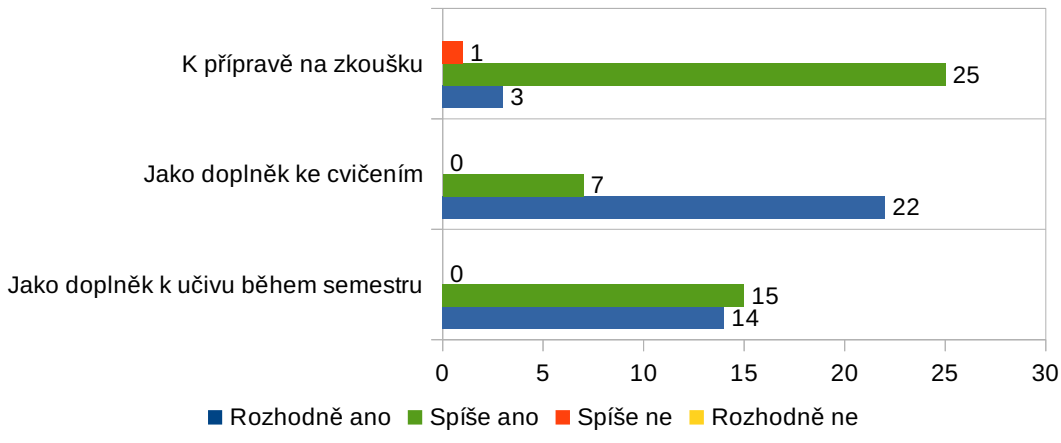
Mezi tři hlavní části kurzu patří přednáška, cvičení a zkouška. Není žádným překvapením, že všech 29 studentů účastnících se kurzu odpovědělo, že sbírku používali jako doplněk ke cvičení. Překvapením to není z toho důvodu, že primárním úkolem této sbírky úloh je právě pomoc při praktickém studiu, neboli na cvičeních. Příjemným zjištěním je poté fakt, že celkem dvanáct studentů tuto sbírku využívalo i jako materiál k přípravě na závěrečnou zkoušku a šest i na přednáškách. Méně příjemná je odpověď tří studentů, kteří uvedli, že sbírku využívali i jiným způsobem. Bohužel ani jeden z nich u této otázky neuvedl slovní odpověď jakým jiným způsobem.



Graf 7: Grafické vyhodnocení otázky č. 7 [vlastní zdroj]

8) Jak hodnotíte celkovou prospěšnost sbírky?

Tato otázka přímo navazuje na tu předchozí. Za pomoci likertovi škály zkoumá, jakou měrou je sbírka prospěšná v jednotlivých částech kurzu. Tedy prospěšnost při cvičení, přednášce a nakonec také u závěrečné zkoušky. Zjištění jsou takřka jenom pozitivní. Za nejvíce prospěšnou je sbírka považována při cvičení. Za spíše prospěšnou pak také u přednášek a zkoušky.

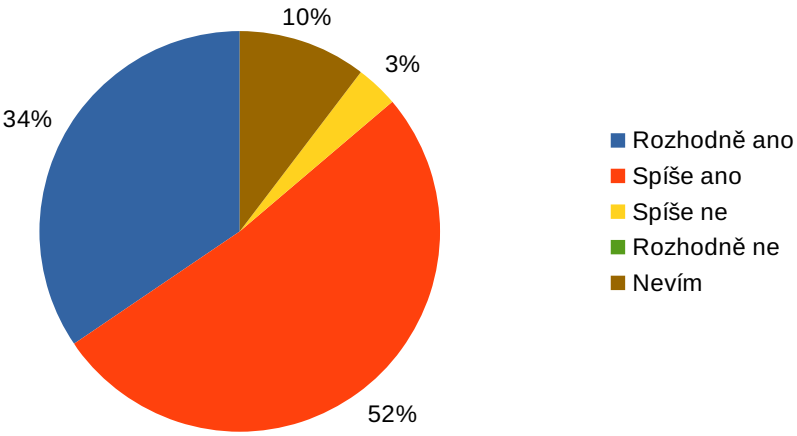


Graf 8: Grafické vyhodnocení otázky č. 8 [vlastní zdroj]

9) Pomohla Vám tato sbírka při studiu problematiky skriptování v Bashi?

Poslední shrnující otázka byla vcelku jednoduchá a přímá. Měla odhalit skutečnost, zdali student považuje sbírku za prospěšnou a zdali mu při jeho studiu pomohla či nikoliv. Zde je opět využita likertova škála vyjadřující míru spokojenosti.

S potěšením lze oznámit, že celých 86 % všech studentů, kteří byli ochotni vyplnit tento dotazník, odpovědělo, že ano. Toto je naprosto pozitivní a uspokojující výsledek a dokazuje, že má sbírka se stala užitečným studijním materiálem.



Graf 9: Grafické vyhodnocení otázky č. 9 [vlastní zdroj]

10) Je něco, co Vám ve sbírce úloh chybělo?

Závěrečnou otázkou bylo uvedení doplňujících poznámek týkajících se nedostatků sbírky úloh. Jednalo se o otevřenou otázku nabízející pouze slovní odpověď. Bohužel ani jeden z dotazovaných studentů svou odpověď neuvedl, což je nepříjemné z důvodu ještě větší zpětné vazby na tuto sbírku úloh.

4.3.4 Srovnání testovaných verzí

Jak již bylo zmíněno v kapitole 4.3.1, v zimním semestru akademického roku 2013/2014 proběhlo testování první verze sbírky úloh. Nabízí se zde tak prostor pro vzájemné srovnání výsledků tohoto předchozího a současného testování sbírky úloh.

V obou případech bylo dosaženo pozitivního hodnocení. Studenti v průběhu studia tuto sbírku využívali jako svůj primární zdroj informací a uvedli, že jim byla celkově velmi nápomocná při závěrečných zkouškách. Jediným rozdílem bylo hodnocení náročnosti řešených a neřešených úloh. V předchozí verzi sbírky byly některé úlohy označeny jako příliš složité. V té současné verzi již bylo vše v pořádku a úlohy byly hodnoceny jako středně těžké. Dotazník a grafické znázornění výsledků prvního testování jsou mimo jiné také přílohou této bakalářské práce.

Hodnocení	**Předchozí verze**	**Současná verze**
Použití sbírky při studiu	ano	ano
Množství teoretických informací	dostatečné	dostatečné
Náročnost úloh	těžké	středně těžké
Prospěšnost sbírky	spíše ano	ano

Tabulka 3: Srovnání testovaných verzí sbírky úloh

5 Závěr

Bakalářská práce se zabývá výukou skriptování v Bashi, která je součástí osnov předmětu Operační systémy 2, vyučovaném na Ústavu aplikované informatiky Přírodovědecké fakulty Jihočeské univerzity v Českých Budějovicích.

Hlavním cílem této bakalářské práce bylo vytvoření uceleného výukového materiálu pro potřeby výuky skriptování v Bashi. Tento cíl byl splněn vytvořením sbírky úloh obsahující potřebný teoretický základ a množinu řešených i neřešených příkladů na dané téma.

Tato sbírka úloh byla následně ve spolupráci s vyučujícím předmětu uvedena do praxe. Na závěr semestru bylo mezi studenty předmětu provedeno dotazníkové šetření, jehož pomocí došlo k vyhodnocení spokojenosti, použitelnosti a také celkové prospěšnosti sbírky při studiu. Výsledek byl pozitivní a vyplynulo z něj, že studenti byli se sbírkou spokojeni a využívali ji jako jeden z hlavních zdrojů informací. Lze tedy předpokládat, že v důsledku použití sbírky úloh a těchto zjištění mohlo dojít ke značnému zefektivnění výuky.

Sbírka úloh se tak ukázala býti užitečným zdrojem informací a nic nebraní tomu, aby se stala stálým výukovým materiálem předmětu Operační systémy 2. Jejím dalším rozšířením by mohlo být navýšení počtu řešených i neřešených úloh, jejich rozdělení do kategorií podle stupně náročnosti či doplnění teorie o pokročilejší techniky skriptování v Bashi.

6 Použitá literatura

[1] FUCHS, Jan. Seriál BASH. ABC Linuxu [online]. 2008. Dostupné z: http://www.abclinuxu.cz/serialy/bash

[2] MILAR, Bohdan. Seriál o Bashi. LinuxEXPRES: opravdový linuxový magazín [online]. Brno: QCM. Dostupné z: http://www.linuxexpres.cz/praxe/serial-o-bashi

[3] BURTCH, Ken O. Linux shell scripting with Bash. Vyd. 1. Indianapolis: Sams Publishing, 2004, 408 s.ISBN 9780672326424

[4] GARRELS, Machtelt. Bash Guide for Beginners. Vyd. 2. Fultus Corporation, 2010, 214 s. ISBN 9781596822016

[5] COOPER, Mendel. Advanced Bash Scripting Guide. Vyd. 2. lulu.com, 2010, 266 s. ISBN 9781435752191

[6] NEWHAM, Cameron. Learning the bash Shell. Vyd. 3. O'Reilly Media, 2009, 354 s. ISBN 9780596009656

[7] BLUM, Richard. Linux command line and shell scripting bible. Vyd. 2. Indianopolis: Wiley, 2011, 812 s. ISBN 9781118004425

[8] MARTINEK, David. Bourne Again SHell - BASH. BLATNÝ, Jan. FIT VUT v Brně [online]. Brno: Vysoké Učení technické v Brně, Fakulta informačních technologií, 2012. Dostupné z: http://www.fit.vutbr.cz/~martinek/gymnazium/bash.html

[9] Ubuntu Desktop Download. [online]. Dostupné z: http://www.ubuntu.com/download/desktop

[10] Nano editor [online]. Dostupné z: www.nano-editor.org

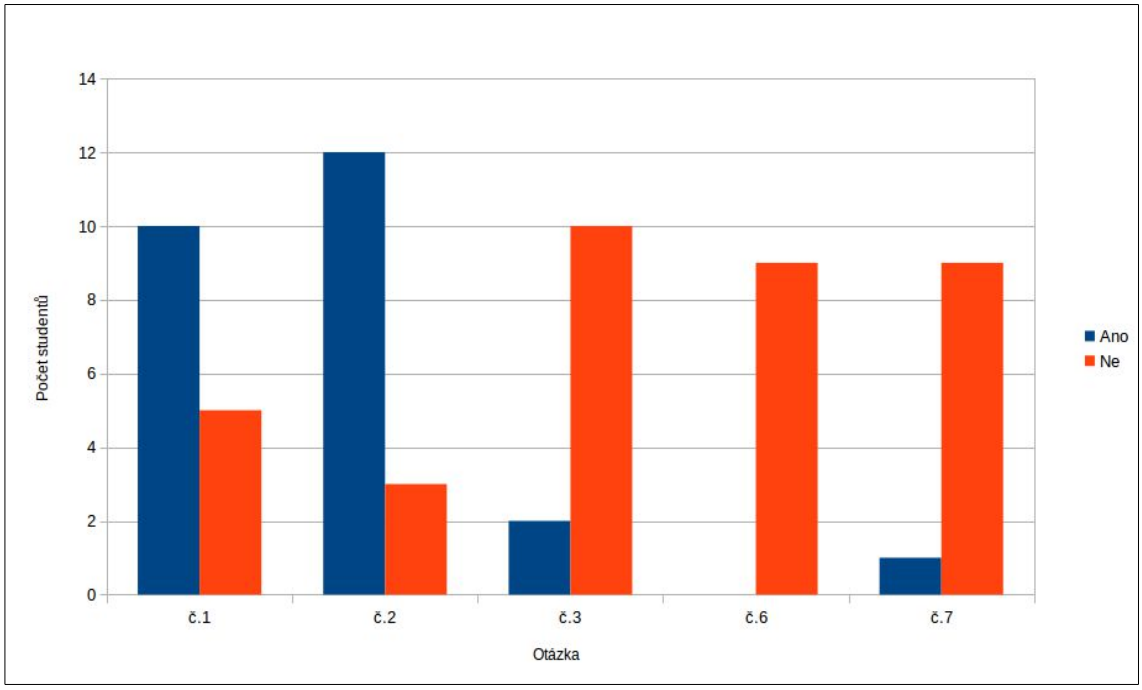
7 Přílohy

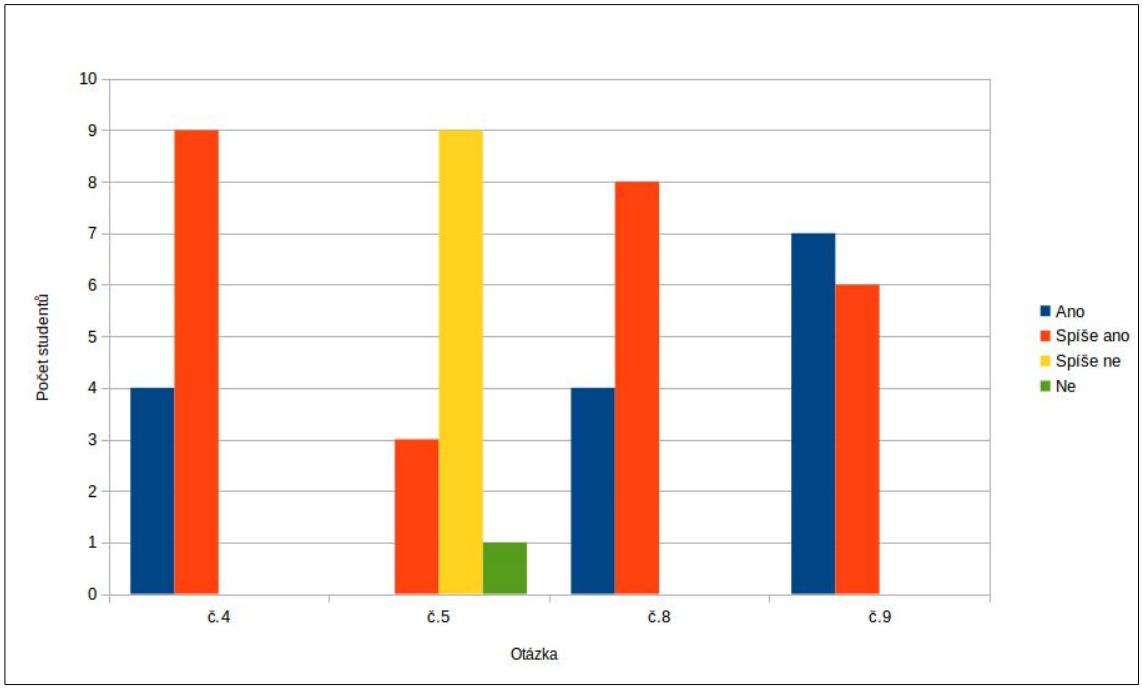
7.1 Dotazník - předchozí verze sbírky úloh

1. Jste ochoten/na zodpovědět otázky v tomto dotazníku?
   a) Ano
   b) Ne
2. Používal/a jste mou sbírku úloh při výuce skriptování v Bashi?
   a) Ano
   b) Ne
3. Měl/a jste již nějaké předešlé zkušenosti ze skriptováním v Bashi?
   a) Ano
   b) Ne
4. Bylo podle Vás množství teorie a informací dostatečné?
   a) Ano
   b) Spíše ano
   c) Spíše ne
   d) Ne
5. Zdáli se Vám neřešené úlohy příliš složité?
   a) Ano
   b) Spíše ano
   c) Spíše ne
   d) Ne

6. Čerpal/a jste při Vašem studiu i z jiného zdroje informací? Pokud ano, uveďte, v čem byl tento zdroj pro Vás přínosnější či naopak.
   a) Ano + odpověď
   b) Ne
7. Je něco, co Vám ve sbírce scházelo? Pokud ano, uveďte, o co se jedná.
   a) Ano + odpověď
   b) Ne
8. Byla pro Vás sbírka úloh přínosem a pomohla Vám při Vašem studiu?
   a) Ano
   b) Spíše ano
   c) Spíše ne
   d) Ne
9. Doporučil/a byste moji sbírku i ostatním?
   a) Ano
   b) Spíše ano
   c) Spíše ne
   d) Ne

7.2 Výsledek vyhodnocení - předchozí verze sbírky úloh





7.3 Dotazník - současná verze sbírky úloh

1. Jak často jste sbírku úloh používal(a)?
   a) Do sbírky jsem se ani nepodíval(a).
   b) Do sbírky jsem nahlédl(a) jen jednou, když jsem se o ní dozvěděl(a).
   c) Sbírku jsem několikrát použil(a) a pročetl(a) si několik úloh.
   d) Sbírku jsem využíval(a) či využívám pravidelněji ke studiu úloh.
   e) Sbírku jsem využíval(a) či využívám pravidelněji, úlohy se snažím řešit samostatně.
   f) Sbírku jsem využíval(a) či využívám jinak.
2. Jaké části sbírky považujete za užitečné?
   a) Úvod kapitoly – (Rozhodně ano, Spíše ano, Spíše ne, Rozhodně ne)
   b) Teorie – (Rozhodně ano, Spíše ano, Spíše ne, Rozhodně ne)
   c) Ukázky použití – (Rozhodně ano, Spíše ano, Spíše ne, Rozhodně ne)
   d) Souhrn kapitoly – (Rozhodně ano, Spíše ano, Spíše ne, Rozhodně ne)
   e) Řešené úlohy – (Rozhodně ano, Spíše ano, Spíše ne, Rozhodně ne)
   f) Vysvětlení řešených úloh – (Rozhodně ano, Spíše ano, Spíše ne, Rozhodně ne)
   g) Neřešené úlohy – (Rozhodně ano, Spíše ano, Spíše ne, Rozhodně ne)
   h) Přehledy – (Rozhodně ano, Spíše ano, Spíše ne, Rozhodně ne)
3. Jak hodnotíte množství teoretických informací obsažených ve sbírce úloh?
   a) Nedostatečný
   b) Dostatečný
   c) Vyčerpávající

4. Jak moc obtížené pro Vás bylo vypracování řešených úloh?

   a) Naprosto jednoduché

   b) Jednoduché

   c) Středně těžké

   d) Těžké

   e) Velmi těžké

5. Jak moc obtížené pro Vás bylo vypracování neřešených úloh?

   a) Naprosto jednoduché

   b) Jednoduché

   c) Středně těžké

   d) Těžké

   e) Velmi těžké

6. Jak jste postupoval(a) při vypracování neřešených úloh?

   a) Vypracoval(a) jsem je z hlavy

   b) Stačilo mi jen to, co jsem si zapamatoval(a) a procvičil(a) při čtení sbírky

   c) Vracel(a) jsem se k teorii ve sbírce

   d) Vracel(a) jsem se k řešeným příkladům ve sbírce

   e) Použil(a) jsem i jiné zdroje. Uveďte jaké.

7. Jakým způsobem jste sbírku úloh využíval(a)?

   a) Jako doplněk k přednášce v průběhu semestru

   b) Jako doplněk ke cvičení v průběhu semestru

   c) K přípravě na zkoušku

   d) Sbírku jsem využíval(a) či využívám jinak:

8. Jak hodnotíte celkovou prospěšnost sbírky?
    a) Jako doplněk k učivu během semestru – (Rozhodně ano, Spíše ano, Spíše ne, Rozhodně ne)
    b) Jako doplněk ke cvičením – (Rozhodně ano, Spíše ano, Spíše ne, Rozhodně ne)
    c) K přípravě na zkoušku – (Rozhodně ano, Spíše ano, Spíše ne, Rozhodně ne)
9. Pomohla Vám tato sbírka při studiu problematiky skriptování v Bashi?
    a) Rozhodně ano
    b) Spíše ano
    c) Spíše ne
    d) Rozhodně ne
    e) Nevím
10. Pokud máte ke sbírce jakoukoli další poznámku, prosím napište ji zde.

7.4 Sbírka úloh

Sbírka řešených a neřešených úloh je zde připojena jako samostatný celek.

Jihočeská univerzita v Českých Budějovicích

Přírodovědecká fakulta

Sbírka řešených a neřešených úloh ze skriptování v Bashi pro výuku předmětu operační systémy 2

Erik Pach

České Budějovice 2015

Obsah

1 Úvod

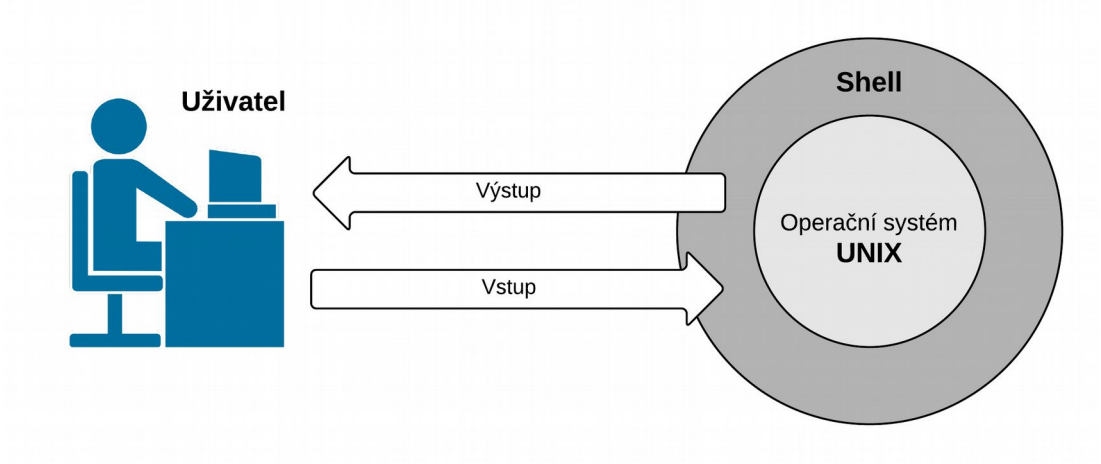
1.1 Shell

Shell, nebo též příkazový procesor či interpret, je textové uživatelské rozhraní ovládané pomocí tzv. příkazové řádky (angl. CLI – Command Line Interface), které je předchůdcem grafického uživatelského rozhraní (angl. GUI - Graphical User Interface).

Primárním úkolem shellu je přijímat příkazy zadané přímo uživatelem nebo je číst ze souboru zvaného shell skript a překládat je do binárního kódu, což jsou instrukce srozumitelné pro operační systém. Poté, co operační systém tyto příkazy provede, je výstup zobrazen zpět uživateli. Shell tedy slouží jako prostředník (překladatel) mezi uživatelem a jádrem operačního systému.

Spousta uživatelů unixových systémů dodnes považuje textové uživatelské rozhraní za mnohem pohodlnější způsob ovládání počítače než grafické uživatelské rozhraní.

Jak se unixové systémy vyvíjely, vyvíjely se i různé typy shellu. Mezi tyto typy patří například SH (Bourne Shell), BASH (Bourne Again Shell), CSH (C Shell), KSH (Korn Shell), TCSH (TENEX C Shell) a další.



Ilustrace 1: Znázornění úlohy shellu v operačním systému

1.2 Bash

Bourne Again Shell vytvořil Brian Fox v roce 1988 pro použití v projektu GNU, v němž se měl stát výchozím interpretem. Tento interpret příkazů vychází, a je zpětně kompatibilní, ze SH (Bourne Shell) a kombinuje výhody CSH (C Shell) a KSH (Korn Shell). Dnes je Bash výchozím interpretem téměř ve všech systémech postavených na linuxovém jádře. Za použití emulace jej lze spustit i v systému Microsoft Windows a dalších.

Syntaxe je kompletně převzata z SH a navíc přidává další příkazy. To znamená, že většinu skriptů napsaných v SH lze bez problémů spustit v Bashi. Mezi hlavní výhody Bashe patří zejména: historie příkazů, automatické doplňování syntaxe (názvy programů, souborů, proměnných atd.) a kontrola procesů.

Dalšími výhodami jsou například zjednodušení vstupně-výstupního přesměrování, provádění celočíselných operací bez pomoci jiných procesů, podpora regulárních výrazů a podpora pro asociativní pole.

1.3 Vytváření a spouštění skriptů

Jak již bylo zmíněno v předchozích kapitolách, shell dokáže přijímat příkazy přímo od uživatele, ale i číst je přímo ze souboru. Tyto soubory se nazývají skripty a skládají se z posloupnosti příkazů, které jsou postupně zpracovávány. Tento přístup je užitečný zejména v případě, kdy potřebujeme opakovaně vykonávat sérii příkazů, které bychom jinak museli vždy ručně zadávat do příkazové řádky. Jsou vhodné také v případě potřeby zadané příkazy spouštět na jiném zařízení či opakovaně v daném časovém intervalu.

Každý skript je jednoduchý textový soubor, který může nebo nemusí obsahovat příponu (obvykle je zvykem používat příponu `.sh`). K tomu, aby byl skript funkční a spustitelný, jsou zapotřebí dvě věci:

1) Uvést jméno interpreta

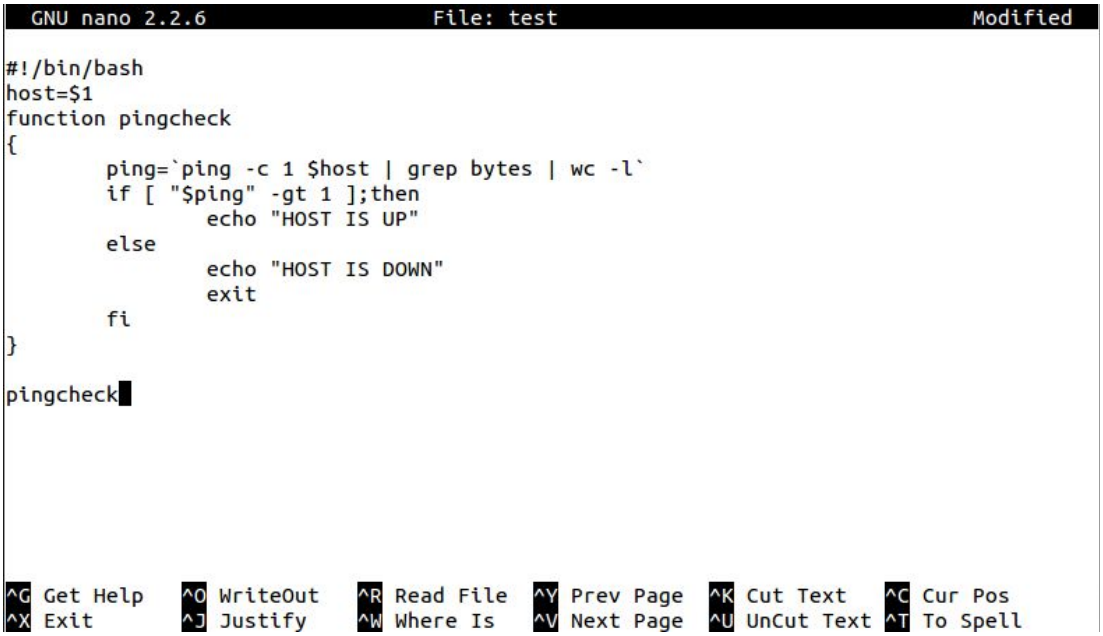
Každý skript musí na své první řádce uvádět jméno interpreta, který bude daný skript obsluhovat. V případě Bashe se jedná o posloupnost znaků `#!/bin/bash`.

2) Nastavit pro skript práva na spuštění

Pokud je skript hotov a jeho obsah je uložen do souboru, nelze tento skript zatím spustit. Je zapotřebí z něho udělat spustitelný soubor. Toho v unixových systémech docílíme velmi snadno - zadáním příkazu `chmod +x skript.sh`.

Nyní lze skript spustit zadáním příkazu `./skript.sh` nebo `sh skript.sh`, ale pouze v případě, že se nacházíme ve stejném adresáři jako skript samotný. Pokud bychom chtěli skript spouštět pouhým zadáním jeho názvu kdekoliv v systému, je potřeba ho umístit do adresáře uvedeného v systémové proměnné `PATH`[1].

1 Více o systémových proměnných v kapitole 2.2



Ilustrace 2: Ukázka jednoduchého Bash skriptu

1.4 Předávání parametrů skriptu

V Bashi je možné vytvářet skripty, které mohou přijímat parametry zadané při spouštění skriptu z příkazové řádky. Tyto parametry, neboli také argumenty, jsou užitečné v případě potřeby, kdy skript musí vykonat rozdílné úlohy právě v závislosti na jeho vstupních parametrech.

Ukázka použití

```
$ sh vytiskni.sh dokument1 dokument2 dokument3
```

Tyto tři parametry (dokument1, dokument2, dokument3) jsou poté ve skriptu zpřístupněny pomocí speciálních proměnných `$1`, `$2`, $3 a tak dále, kde proměnná `$1` odkazuje na první předaný argument, proměnná `$2` na druhý předaný argument a tak dále. Speciální význam má proměnná `$0`, která uchovává kompletní název skriptu a také proměnná `$#`, která uchovává počet přijatých parametrů.

1.5 Komentáře

Komentáře jsou nedílnou součástí každého programovacího jazyka a každý programátor by je měl využívat. Hlavním důvodem pro komentování zdrojového kódu je jeho následné snazší pochopení jiným čtenářem. Tito čtenáři pak mohou lépe pochopit danou úlohu a proč byla právě takto napsána.

V Bashi existují dva druhy komentářů. Prvním z nich je tzn. řádkový komentář, který začíná znakem #. Vše za tímto znakem až do konce řádky je považováno za komentář a při běhu skriptu je to ignorováno. Druhým typem je tzv. HERE, neboli víceřádkový komentář. Tento komentář začíná sérií znaků `<<HERE` a končí `HERE`, kde slovo „HERE“ může být nahrazeno jakýmkoliv jiným slovem.

Ukázka použití

```
# Toto je jednoradkovy komentar
<<KOMENTAR
Toto je
viceradkovy komentar
KOMENTAR
```

1.6 Souhrn

- UNIX je otevřený svobodný operační systém vytvořený na počátku 70. let.
- Linux je založen na operačním systému Unix. Jedná se o jeho vylepšenou verzi s mnoha vylepšeními včetně grafického uživatelského prostředí.
- Shell je textové rozhraní, které slouží k přijímání příkazů a jejich následnému překládání do jazyka, kterému rozumí operační systém.
- BASH je interpret příkazů odvozený od SH a slučující výhody CSH a KSH.
- Aby mohl být skript spustitelný, musí být na první řádce uveden název interpreta a soubor musí mít práva na spuštění.
- Při spuštění je možné skriptu předávat parametry, které jsou poté zpřístupněny pomocí speciálních proměnných.

- Je dobrým zvykem zdrojový kód patřičně komentovat použitím jednoho či více řádkových komentářů

1.7 Příklady

Příklad 1.7.1

Zadání

Vytvořte skript, který na standardní výstup (obrazovku) vypíše text „Ahoj světe!“.

Řešení

```
#!/bin/bash
echo “Ahoj světe!“
```

Vysvětlení

1) Tento skript obsahuje pouze dvě řádky. První řádka říká systému, jaký interpret příkazů má být použit pro zpracování souboru.
2) Druhá řádka pouze pomocí příkazu `echo` vypíše řetězec „Ahoj světe!“.

Příklad 1.7.2

Zadání

Vytvořte skript, který na standardní výstup (obrazovku) vypíše text „Ahoj jmeno!“, kde „jmeno“ je nahrazeno řetězcem, který je předán jako parametr skriptu při jeho spuštění.

Řešení

```
#!/bin/bash
echo “Ahoj $1 !“
```

Vysvětlení

1) Na první řádce je opět uvedena cesta k interpretu příkazů.

2) Na druhé řádce je pomocí příkazu `echo` vypsán složený řetězec, který se skládá z řetězce „Ahoj“, poté následuje proměnná `$1` (první parametr skriptu) a nakonec řetězec „!“.

Příklad 1.7.3

Zadání

Vytvořte skript, který na standardní výstup (obrazovku) vypíše počet argumentů, které byly zadány při jeho spuštění.

Řešení

```
#!/bin/bash
echo $#
```

Vysvětlení

1) Tento skript vypíše pomocí příkazu `echo` hodnotu speciální proměnné `$#`, která v sobě uchovává počet argumentů zadaných při spuštění skriptu.

Příklad 1.7.4

Zadání

Vytvořte skript, který pomocí příkazu `pwd` vypíše aktuální cestu. Poté se pomocí příkazu `cd` přepne do adresáře `/tmp` a zde za použití příkazu `touch` vytvoří soubor „temp.txt“. Nakonec se přepne do adresáře aktuálně přihlášeného uživatele.

Řešení

```
#!/bin/bash
pwd
cd /tmp
touch temp.txt
cd ~
```

Vysvětlení

1) Uvedení cesty k interpretu příkazů.
2) Vypsání absolutní cesty k adresáři, ve kterém se aktuálně nacházíme.

3) Přepnutí se do adresáře `/tmp`.

4) Vytvoření prázdného souboru „temp.txt“.

5) Přepnutí se do domovského adresáře aktuálně přihlášeného uživatele.

2 Proměnné a prostředí

Co se v této kapitole naučíte

- co jsou to proměnné a k čemu slouží
- co jsou to globální proměnné a jaký mají význam
- jak se proměnné vytváří a používají
- co jsou to pole a konstanty

2.1 Proměnné

Jako v ostatních programovacích jazycích, tak i v Bashi jsou proměnné určeny pro ukládání dat. Na rozdíl od ostatních jazyků ale Bash nerozlišuje datové typy. Proměnné v Bashi mohou obsahovat číslo, znak nebo sadu znaků (řetězec).

Proměnné jsou ukládány do sdílené paměti a lze k nim tedy přistupovat a pracovat s nimi o mnoho rychleji, než například s daty uloženými v souboru na pevném disku. Poskytují dočasné úložiště pro informace, které jsou potřebné při běhu programu.

2.2 Systémové proměnné

Systémové proměnné (nebo také proměnné prostředí či shellu) jsou proměnné, které byly vytvořeny a jsou spravovány samotným shellem a jsou dostupné ve všech jeho typech, tedy i v Bashi. Jejich názvy jsou definovány velkými písmeny. Slouží především jako úložiště pro nastavení Bashe a systému, které lze podle potřeby měnit. Je v nich uloženo například jazykové nastavení, cesta k domovskému adresáři a tak dále. Jejich seznam lze zobrazit zadáním příkazu `env` nebo `printenv`. Jejich zkrácený seznam je součástí přílohy této sbírky.

Ukázka

```
echo $HOME
/home/linux
```

2.3 Vytvoření proměnné

Proměnnou lze vytvořit zadáním jejího názvu a hodnoty. V názvu proměnné lze použít velká i malá písmena abecedy, čísla a některé speciální znaky. Je doporučeno udávat název proměnné malými písmeny z důvodu nepřepsání jedné ze systémových proměnných, jejichž název je uveden velkými písmeny. Název nesmí začínat číslicí a měl by smysluplně popisovat účel proměnné.

Syntaxe

```
NAZEV_PROMENNE=HODNOTA
```

Proměnnou lze v Bashi vytvořit (deklarovat) velmi jednoduchým způsobem. Jedná se o pouhé přiřazení hodnoty do proměnné pomocí operátoru =. U tohoto způsobu je potřeba dbát zvýšené opatrnosti při psaní, jelikož je nutné, aby před a za operátorem = nebyla žádná mezera (viz. ukázka použití). Takto nadeklarovanou proměnnou lze poté použít pouhým napsáním názvu proměnné, kterému předchází znak $.

Ukázka použití

```
#!/bin/bash
PROMENNA1=123
PROMENNA2='Text.'
echo $PROMENNA1
```

2.4 Pole

Pole je seřazená kolekce položek zvaných elementy. Každý element má svou hodnotu a je identifikován svým klíčem, který jednoznačně identifikuje prvek v daném poli. Na rozdíl od většiny ostatních programovacích jazyků, které povolují

jako hodnotu klíče i textové řetězce o různé délce, Bash umožňuje pouze hodnoty číselné. Pokud není stanoveno jinak, tak hodnota klíče prvního prvku v poli je nula.

Vytvoření pole

```
jmena=("Bob" "Peter" "$USER" "Big Bad John")
```

Toto je nejběžnější způsob vytvoření pole. Skládá se z názvu pole umístěného vlevo od znaménka „=“ a z výčtu prvků uzavřeného v kulatých závorkách na straně pravé. Tyto prvky jsou vždy odděleny mezerou.

2.5 Konstanty

Konstanta je označení pro speciální druh proměnné, jejíž hodnota je v průběhu provádění programu neměnná. Tato konstanta poté může být v programu použita na více místech. Hlavní výhodou konstanty je možnost nastavit její hodnotu pouze jednou a na tuto hodnotu se poté odkazovat pomocí jejího názvu. Odpadá tak starost měnit potřebné hodnoty na více místech.

Konstantu lze v Bashi vytvořit příkazem `readonly` nebo `declare`.

Ukázka použití

```
readonly DATA=/home/ubuntu/obrazky/auto.png
echo $DATA
```

2.6 Souhrn

- Proměnné slouží pro dočasné ukládání dat.
- Jsou uloženy v operační paměti a lze k nim přistupovat rychleji.
- Proměnná typu pole slouží k ukládání více hodnot v jedné proměnné. Obsahuje položky které mají svou hodnotu a jsou identifikovány klíčem.
- Konstanta je typ proměnné, jejíž hodnota je nadefinována pouze jednou a poté se již v průběhu skriptu nemění.

2.7 Příklady

Příklad 2.7.1

Zadání

Vytvořte skript, který pomocí příkazu `read` od uživatele načte název internetové domény a tuto hodnotu uloží do proměnné s názvem „IDN“. Příkazem host poté zjistí a vypíše IP adresu, která se skrývá pod danou doménou.

Řešení

```
#!/bin/bash
echo "Zadejte nazev internetove domeny"
read IDN
host $IDN
```

Vysvětlení

1) Pomocí příkazu `echo` vypíšeme textový řetězec s nápovědou, aby uživatel věděl, co má dělat.
2) Příkaz `read` čeká na vstup od uživatele, přičemž zadanou hodnotu poté uloží do proměnné, která je specifikována právě za příkazem `read`.
3) Nakonec se spuštěním příkazu `host` vyhledá IP adresa pro danou doménu a vypíše se na standardní výstup.

Příklad 2.7.2

Zadání

Vytvořte skript, který postupně načte tři uživatelem zadaná čísla (rozměry objektu v metrech). Tento skript poté vypočítá a zobrazí celkový povrch a objem objektu.

Řešení

```
#!/bin/bash
echo "Zadejte delku prvni strany (m2)"
read a
echo "Zadejte delku druhe strany (m2)"
read b
echo "Zadejte vysku (m2)"
read c

povrch=$(( 2 * ($a * $b + $b * $c + $a * $c) ))
objem=$(( $a * $b * $c ))

echo "Povrch objektu je:" $povrch "m2"
echo "Objem objektu je:" $objem "m2"
```

Vysvětlení

1) Skript postupně od uživatele načte tři hodnoty, které si postupně uloží do proměnných `a`, `b` a `c`, přičemž u každé pomocí příkazu `echo` zobrazí malou nápovědu.
2) Poté se provede výpočet povrchu a objemu, jehož výsledek se rovněž uloží do vlastní proměnné.
3) Aritmetický výpočet a vyhodnocení je provedeno tak, že celý výraz je uzavřen mezi znaky `$(( … ))`.
4) Nakonec se zobrazí pouze výsledné hodnoty.

Příklad 2.7.3

Zadání

Vytvořte skript, který spočítá spropitné a to tak, že postupně načte dvě uživatelem zadané hodnoty. První z nich bude celková utracená hodnota a druhá procentuální hodnota spropitného.

Řešení

```
#!/bin/bash
read -p "Zadejte celkovou utratu (Kc): " utrata
read -p "Zadejte vysi spropitneho (%): " vyse_spropitneho

spropitne=$(( ($utrata * $vyse_spropitneho) / 100 ))
celkem=$(( $spropitne + $utrata ))

echo "Utrata:" $utrata "Kc"
echo "Spropitne ("$vyse_spropitneho"%):" $spropitne "Kc"
echo "Celkem k zaplaceni:" $celkem "Kc"
```

Vysvětlení

1) Na začátku si skript pomocí příkazu `read` od uživatele načte požadované hodnoty.
2) Namísto použití příkazu `echo` pro zobrazení nápovědy se zde využívá přepínače `-p` dostupného u příkazu `read`.
3) Poté se provede aritmetický výpočet hodnot, který je opět uzavřen mezi znaky `$(( … ))`.
4) Nakonec se vypíší všechny potřebné údaje a skript je u konce.

Příklad 2.7.4

Zadání

Vytvořte skript, který pomocí příkazu `read` načte od uživatele jednu hodnotu. Tato hodnota musí být celé kladné číslo, které bude reprezentovat počet stupňů Celsia. Tuto hodnotu poté převede na stupně Fahrenheita a zobrazí ji na standardní výstup.

Řešení

```
#!/bin/bash
read -p "Zadejte pocet stupnu celsia: " stupne_celsia

stupne_fahrenheita=$(( $stupne_celsia * 9/5 + 32 ))
echo $USER "rika, ze" $stupne_celsia "C =" $stupne_fahrenheita "F"
```

Vysvětlení

1) Skript od uživatele načte požadovanou hodnotu pro převod a uchová si ji v proměnné `stupne_celsia`
2) Poté pomocí vzorce provede výpočet a jeho výsledek uloží do proměnné `stupne_fahrenheita`.
3) Na závěr vypíše výsledek, přičemž ve výsledné zprávě zahrne hodnotu globální proměnné `USER`.

Příklad 2.7.5

Zadání

Vytvořte skript pro práci s polem. Tento skript si na začátku načte do pole všechny konfigurační soubory (s koncovkou „`.conf`“) nacházející se v adresáři „`/etc/`“ a dále bude pracovat s tímto polem. Jeho prvním úkolem bude vypsat celkový počet, první, druhý a poslední prvek v poli. Dále pomocí cyklu `for` projde všechny prvky v tomto poli a vypíše je na standardní výstup. Posledním úkolem bude pomocí libovolného cyklu odebrat prvních pět prvků pole a následně, pro ověření, toto celé pole opět zobrazit.

Řešení

```
#!/bin/bash
soubory=(/etc/*.conf)
pocet=${#soubory[*]}
echo "Celkovy pocet souboru:" $pocet
echo "Prvni soubor:" ${soubory[0]}
echo "Druhy soubor:" ${soubory[1]}
echo "Posledni soubor:" ${soubory[$(( $pocet-1 ))]}
for soubor in "${soubory[@]}"
do
      echo "$soubor"
done
echo ""
```

```
for (( i=0; i<=4; i++ ))
do
   soubory[$i]=''
done
echo ${soubory[@]}
```

Vysvětlení

1) Tento skript si nejdříve načte do proměnné „`soubory`“ všechny soubory s koncovkou „`.conf`“ nacházející se v adresáři „`/etc/`“.
2) Poté pomocí příkazu `echo` vypíše požadované informace.
3) Na první a druhý prvek v poli se odkazuje pomocí indexů 0 a 1.
4) Poslední prvek v poli je označen indexem s hodnotou o jednu menší než je celkový počet prvků v poli, poněvadž indexování začíná od nuly.
5) Jednoduchým cyklem `for` je poté pole procházeno a jeho prvky jsou postupně vypisovány.
6) Pomocí dalšího průchodu cyklem `for` je odebráno prvních pět prvků. Prvky jsou odebrány nastavením hodnoty na prázdný řetězec.
7) Nakonec je toto pole pro ověření smazání prvků znovu vypsáno.

3 Řízení toku programu (rozhodování)

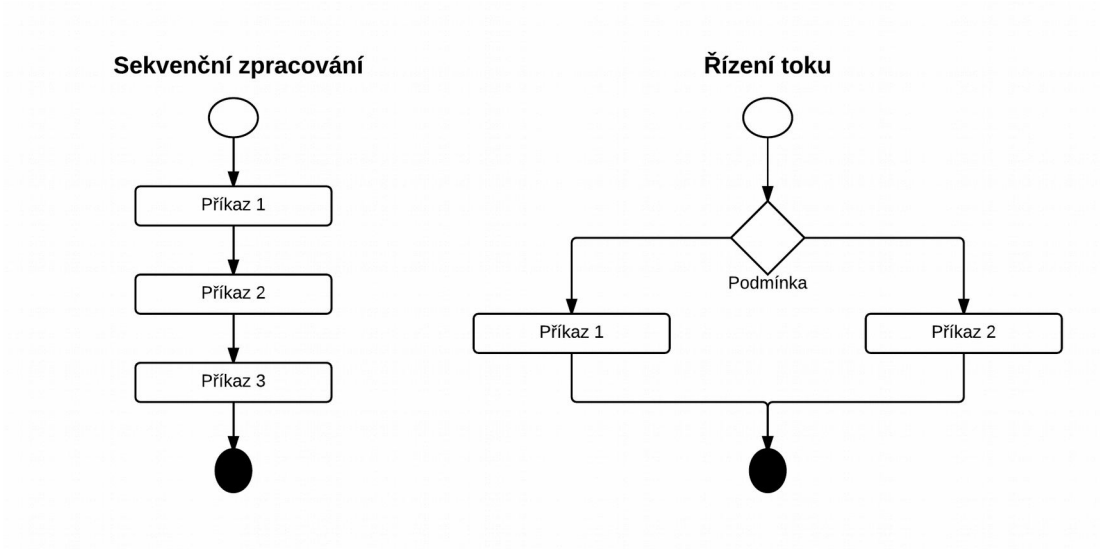
Co se v této kapitole naučíte

- co je to řízení toku programu a jaké se k němu používají struktury
- vyhodnocovat podmínky pomocí příkazu `test`
- řídit tok skriptu pomocí struktury `if` a `case`
- používat logické operátory `AND`, `OR` a `NOT`
- porovnávat čísla, textové řetězce a také soubory a jejich atributy
- předávat skriptu vstupní parametry ovlivňující jeho běh

3.1 Řízení toku

Doposud byl náš skript prováděn pouze takzvaně sekvenčně. To znamená, že každý příkaz v tomto skriptu byl proveden pouze jednou a to v pořadí, ve kterém byl napsán. Tento přístup je velice omezený a nelze s ním vytvářet více komplexní (inteligentní) skripty.

Pomocí řídících struktur `if` a `case` lze docílit provádění určitého příkazu, či dokonce skupiny příkazů, na základě dané podmínky a tím tak rozdělit průběh skriptu na více větví, čímž se stane více interaktivním.



Ilustrace 3: Zpracování skriptu (sekvenčně, pomocí řízení toku)

3.2 Podmínka

V předchozí části o řízení toku byl zmíněn pojem podmínka. Podmínka není nic jiného než výraz, jehož výsledkem je logický datový typ (angl. Boolean), který může nabývat pouze dvou hodnot a to - je pravda (angl. True, v prostředí shellu hodnota 1) či nepravda (angl. False, v prostředí shellu hodnota 0). Používá se především ve spojení s řídícími strukturami `if`, `case` a se smyčkami, které jsou náplní další kapitoly.

Ukázka jednoduché podmínky

```
echo $(( 4 < 8 ))
```

Tento příkaz jednoduše porovná dvě zadaná čísla a vrátí výsledek porovnání. V tomto případě je to hodnota 1, protože číslo 4 je opravdu menší než číslo 8.

Ukázka podmínky ve struktuře if

```
if [ existuje soubor /home/seznam.txt ]
then
      vytvoř kopii
else
      zobraz chybove hlaseni
fi
```

3.3 Příkaz test

Příkaz `test` je vnitřní příkaz shellu, který slouží k porovnávání čísel, textových řetězců, typu a dalších atributů souborů. Používá se zejména ve spojení s podmínkami a strukturami pro řízení toku.

Syntaxe příkazu test

```
test podminka && prikaz_kdyz_pravda || prikaz_kdyz_nepravda
```

Ukázka použití příkazu test

```
test 5 -gt 2 && echo "Je vetsi" || echo "Je mensi"
```

Tento příkaz porovná, zdali je hodnota 5 větší než hodnota 2 a na základě výsledku provede daný příkaz. V tomto případě vypíše text „Je vetsi“.

Více o příkazu **test**, jeho operátorech a použití se můžete dozvědět v manuálových stránkách (příkaz **man test**) nebo v příloze této sbírky.

3.4 Řídící struktura if

Příkaz `if` je základní a pravděpodobně i nejpoužívanější řídící struktura. Jeho úkolem je provádět jeden či více příkazů na základě vyhodnocení podmínky (logického výrazu). Často se používá ve spojení s příkazem `test`.

Syntaxe

```
if podminka
then
      prikaz1
      prikaz2
      ...
      prikazN
fi
```

Tato konstrukce provede příkazy `prikaz1`, `prikaz2` až `prikazN` pouze v případě, že daná podmínka byla vyhodnocena jako pravdivá. Pokud bychom chtěli provést nějaké příkazy i v případě, že daná podmínka byla vyhodnocena jako nepravdivá, můžeme přidat klíčové slovo `else`.

Syntaxe

```
if podminka
then
      pravda
else
      nepravda
fi
```

Pokud potřebujeme vyhodnotit více než jednu podmínku v dané `if` konstrukci, můžeme použít klíčové slovo `elif`.

Syntaxe

```
if podminka1
then
      prikaz1
elif podminka2
then
      prikaz2
fi
```

Tato konstrukce nejdříve vyhodnotí první podmínku a pokud je pravdivá, provede příkazy v dané větvi. Pokud je výsledkem první podmínky nepravda, přejde se k vyhodnocení druhé podmínky, kde se celý proces znovu opakuje. Takto lze v jedné konstrukci vyhodnotit neomezené množství podmínek.

Vnořené příkazy if

Všechny typy konstrukcí `if`, které byly doposud představeny, lze libovolně zanořovat do sebe. To znamená, že v jedné větvi se může nacházet další konstrukce `if` a v té zase další a tak dále. Z hlediska čitelnosti, a hlavně porozumění kódu, není přílišné zanořování doporučeno.

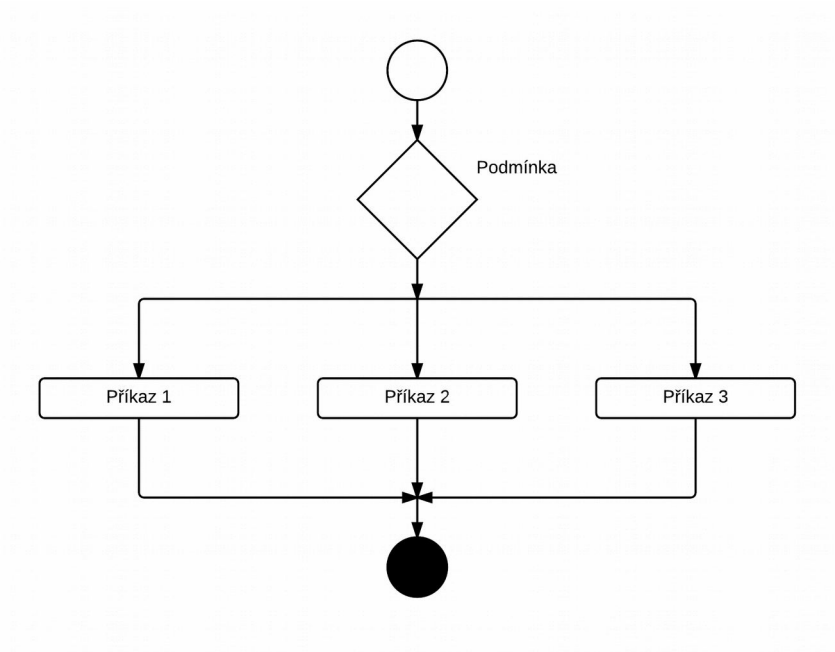
Příklad

```
if podminka1
then
      if dalsiPodminka1
      then
            dalsiPrikaz1
      fi
elif podminka2
then
      if dalsiPodminka2
      then
            dalsiPrikaz2
      else
            dalsiPrikaz3
      fi
fi
```

3.5 Řídící struktura case

Tato řídící struktura se používá jako adekvátní náhrada za víceúrovňový příkaz `if`, který se při větším počtu porovnání stává nepřehledným. Umožňuje nám porovnat několik hodnot vůči jedné proměnné a na základě nalezené shody provést

různé příkazy nebo skupiny příkazů. Syntaxe této struktury je jednoduší na pochopení i vytvoření.



Ilustrace 4: Řídící struktura case

Syntaxe

```
case $promenna in
      vzor1)
            prikazy;;
      vzor2)
            prikazy;;
      ...
      vzorN)
            prikazy;;
      *)
            prikazy;;
esac
```

Syntaxe příkazu case začíná slovy `case slovo in`, kde místo slova vkládáme nejčastěji hodnotu proměnné. Poté následuje seznam hodnot zakončených zavírací závorkou, kterým, když proměnná vyhovuje, vykoná se soubor příkazů uvedených pod danou hodnotou.

Každá takováto sada příkazů musí být vždy zakončena dvěma středníky. Pokud chceme provést nějaký příkaz i v případě, že žádná z daných podmínek nevyhovovala, můžeme vložit tzv. obecnou podmínku (znak hvězdička). Jako u předchozích konstrukcí je i tato zakončena svým obráceným názvem. V případě konstrukce `case` je to `esac`.

Ukázka použití

```
case $operace in
      "scitani") echo $a + $b;;
      "odcitani") echo $a - $b;;
      "nasobeni") echo $a * $b;;
      "deleni") echo $a / $b;;
      *) echo "Neznama operace!";;
esac
```

3.6 Logické operátory

Logické operátory slouží k vytváření tzv. logických výrazů, které jsou součástí podmíněných příkazů a podmínek. Mezi nejpoužívanější logické operátory patří logická negace (NOT), logický součin (AND) a logický součet (OR).

3.7 Souhrn

- Řízení toku slouží k rozdělení běhu skriptu na několik možných cest (posloupností příkazů) na základě dané podmínky.
- Podmínka je logický výraz, jehož výsledkem je buď pravda či nepravda. Podle této hodnoty se skript rozhoduje, jakou cestou se vydá.
- Nejpoužívanější konstrukce pro větvení skriptu je příkaz `if`.
- Pro vícečetné porovnávání je vhodná konstrukce `case`, která je jednodušší a přehlednější nežli použití `if-elif-else` konstrukce.
- Příkaz `test` slouží k porovnávání čísel, textových řetězců, typu a dalších atributů souborů a je úzce spojen s konstrukcemi pro řízení toku.

3.8 Příklady

Příklad 3.8.1

Zadání

Upravte skript z příkladu 2.7.4. Přidejte možnost určit směr převodu, tzn. zdali se bude převádět ze stupňů Celsia na stupně Fahrenheita či naopak. Tuto změnu realizujte přidáním dalšího příkazu read, pomocí kterého získáte od uživatele informaci o směru převodu. Podle této hodnoty poté proveďte daný výpočet a hodnotu zobrazte na standardní výstup.

Řešení

```
#!/bin/bash
echo "Zvolte smer prevodu"
echo "Hodnota 1: C -> F"
echo "Hodnota 2: F -> C"
read smer

if [ $smer -eq 1 ]
then
      read -p "Zadejte pocet stupnu celsia: " stupne_c
      stupne_f=$(( $stupne_c * 9/5 + 32 ))
      echo $stupne_c "C =" $stupne_f "F"
elif [ $smer -eq 2 ]
then
      read -p "Zadejte pocet stupnu fahrenheita: " stupne_f
      stupne_c=$(( 5/9 * $stupne_f - 32 ))
      echo $stupne_f "F =" $stupne_c "C"
else
      echo "Nespravny smer (povoleny jsou pouze hodnoty 1 a 2)"
      exit 1
fi
```

Vysvětlení

1) Na začátku je příkazem `echo` vypsána nápověda.
2) Příkaz `read` načte uživatelem zadanou možnost a uloží ji do proměnné `smer`.
3) Poté se pomocí konstrukce `if-elif-else` rozhodne, jaká možnost byla vybrána a provede se daný úsek kódu.
4) Byla-li vybrána možnost „1“, tak se první podmínka vyhodnotí jako pravdivá a provede se převod ze stupňů Celsia na stupně Fahrenheita. V opačném případě se pokračuje dále.
5) Nebyla-li vybrána možnost „1“, tak skript pokračuje další podmínkou. Tentokrát kontroluje, zdali se hodnota nerovná „2“. Pokud ano, provede převod ze stupňů Fahrenheita na stupně Celsia. V opačném případě pokračuje dále.
6) Pokud ani druhá podmínka nebyla vyhodnocena jako pravdivá, provede se kód ukrývající se v části `else`. Tento blok vypíše na standardní výstup zprávu o tom, že byla zadána nesprávná hodnota. Tímto skript končí.

Příklad 3.8.2

Zadání

Vytvořte skript, který bude jako svůj jediný argument přijímat celé číslo, které bude reprezentovat kalendářní rok a poté zjistí, zdali je tento rok přestupný.

Řešení

```
#!/bin/bash
if [ $# -ne 1 ]
then
        echo "Pouziti: "$(basename $0) "cislo_roku"
        exit 1
fi
rok=$1
```

```
if [ $(($rok % 400)) -eq "0" ]
then
        echo "Rok" $rok "je prestupny."
elif [ $(($rok % 4)) -eq 0 ]
then
        if [ $(($rok % 100)) -ne 0 ]
        then
                echo "Rok" $rok "je prestupny."
        else
                echo "Rok" $rok "neni prestupny."
        fi
else
        echo "Rok" $rok "neni prestupny."
fi
```

Vysvětlení

1) První konstrukce `if` v tomto skriptu kontroluje, zdali se počet argumentů skriptu rovná jedné. Pokud tomu tak není, je uživateli zobrazena nápověda s ukázkou správného použití a skript končí.
2) Pokud skript prošel úvodní kontrolou, tak se jeho jediný parametr uloží do proměnné rok.
3) Pomocí konstrukce `if-else` a zanořené konstrukce `if-else` se provede vyhodnocení přestupného roku.
4) Rok je přestupný v případě, že je beze zbytku dělitelný 400, nebo je beze zbytku dělitelný 4, ale už ne číslem 100.
5) Po vyhodnocení podmínek se uživateli vypíše zpráva o tom zdali je rok přestupný či nikoliv a skript končí.

Příklad 3.8.3

Zadání

Vytvořte skript, který bude přijímat jediný vstupní argument. Tento argument musí nabývat jedné z hodnot „`-d`“, „`-m`“ nebo „`-c`“. Na základě vybrané možnosti poté

zobrazí název aktuálního dne (`-d`), měsíce (`-m`) nebo přímo aktuální čas (`-c`). Pokud nebyla vybrána ani jedna z možností, zobrazí se zpráva s nápovědou pro správné použití. Využijte konstrukce `case`.

Řešení

```
#!/bin/bash
case "$1" in
        -d)
                echo $(date +"%A")
                ;;
        -m)
                echo $(date +"%B")
                ;;
        -c)
                echo $(date +"%H:%M:%S")
                ;;
        *)
                echo "Nespravna nebo nezadana moznost."
                echo "Pouziti:" $(basename $0) "-d|-m|-c"
                ;;
esac
```

Vysvětlení

1) Tento skript je velmi jednoduchý. Využívá konstrukce `case`, pomocí které kontroluje proměnnou `$1` a porovnává ji se specifikovanými možnostmi.
2) Pokud nebyla zadána možnost, nebo tato možnost není správná, zobrazí se uživateli nápověda s informací o správném použití.

4 Cykly (smyčky)

Co se v této kapitole naučíte

- co jsou to cykly a jak lze s jejich pomocí opakovaně provádět danou úlohu
- procházet sadu souborů či parametry skriptu
- využívat základní cykly `for`, `while` a `until`
- vytvářet komplexní menu pomocí cyklu `select`

4.1 Cyklus

Cyklus, neboli také smyčka, je řídící struktura, která umožňuje opakované provádění jednoho či více daných příkazů. Součástí všech cyklů je podmínka, která ovlivňuje běh cyklu a určuje, kdy se má provádění příkazů ukončit. Dokud je podmínka pravdivá, cyklus se opakuje. Stejně tak jako se dají zanořovat podmínky, můžeme do sebe libovolně zanořovat i cykly.

4.2 Cyklus for

Prvním ze zde probíraných cyklů je cyklus `for`. Tento cyklus se vyskytuje téměř ve všech programovacích jazycích a je jedním z nejjednodušších. Tento cyklus postupně prochází seznam hodnot, přičemž si tuto hodnotu vždy uloží do proměnné. Pro každou takto uloženou hodnotu pak provede příkazy uvedené v těle celé konstrukce.

Syntaxe

```
for promenna in polozka1 polozka2 ... plozkaN
do
      prikaz1
      prikaz2
      ...
      prikazN
done
```

Cyklus for začíná klíčovým slovem `for`. Následuje název proměnné, do které se v každém opakování uloží hodnota ze seznamu uvedeného za slovem `in`. Pro každou takto načtenou hodnotu se poté provede tělo cyklu, které je ohraničeno slovy `do` – `done`.

Ukázka použití

```
echo "Nakupni seznam:"
for polozka in hruska pomeranc banan kiwi
do
      echo $polozka
done
```

4.3 Cyklus while

Cyklus `while`**,** stejně jako cyklus for, používá pro opakované provádění jednoho či skupiny příkazů. Tělo cyklus se provádí tak dlouho, dokud podmínka uvedená za klíčovým slovem `while` je vyhodnocena jako pravdivá. Pokud je podmínka vyhodnocena jako nepravdivá, skript pokračuje ve svém běhu dále.

Syntaxe

```
while [ podminka ]
do
      prikaz1
      prikaz2
      ...
      prikazN
done
```

Syntaxe této konstrukce je velmi jednoduchá. Začíná klíčovým slovem `while`, po němž následuje podmínka, která se znovu vyhodnocuje při každém opakování. Dokud je tato podmínka pravdivá, provádí se příkazy v těle cyklu, které je stejně jako u cyklu for uvozeno klíčovými slovy `do – done`.

Ukázka použití

```
x=0
while [ $x < 10 ]
do
        echo $x
        x=$((x+1))
done
```

4.4 Cyklus until

Cyklus `until` je velmi podobný cyklu `while`. Oba tyto cykly plní stejnou funkci, avšak liší se ve způsobu vyhodnocení podmínky. Zatímco u konstrukce `while` se tělo cyklu provádí stále dokola pouze pokud je podmínka pravdivá (tzn. cyklus skončí, byla-li podmínka vyhodnocena jako nepravdivá), u cyklu `until` platí, že se tělo cyklu opakuje, dokud podmínka není pravdivá (tzn. dokud je podmínka vyhodnocována jako nepravdivá, cyklus pokračuje).

Dalším rozdílem oproti cyklu `while` je fakt, že tělo cyklu `until` se vždy provede minimálně jedenkrát. Tento cyklus je vhodný zejména pro skripty, které čekají na nějakou událost.

Syntaxe

```
until [ podminka ]
do
      prikaz1
      prikaz2
      ...
      prikazN
done
```

Syntaxe cyklu `until` je prakticky totožná se syntaxí cyklu `while`. Jediným rozdílem je klíčové slovo `until`.

Ukázka použití

```
until test -f "/etc/mylog"
do
      echo "Soubor /etc/mylog není platny soubor. Cekam na zmenu."
      sleep 10
done
```

4.5 Cyklus select

Při vytváření interaktivních skriptů v Bashi je často užitečné nabídnout uživateli na výběr z více možností provedení. Přesně pro tento účel je navržen cyklus `select`. Tento cyklus zobrazí seznam možností (menu), kde každá položka tohoto seznamu je očíslována. Uživatel poté zadáním jedné z této číselných hodnot vybere požadovanou akci. Často se používá ve spojení s konstrukcí `case`.

Syntaxe

```
select promenna in seznam
do
      prikaz1
      prikaz2
      ...
      prikazN
done
```

Syntaxe tohoto cyklu začíná klíčovým slovem `select`. Za tímto slovem je uveden název proměnné, do které se uloží vybraná hodnota ze seznamu hodnot uvedených za klíčovým slovem `in`. Poté už následuje jen tělo cyklu, které je uzavřeno mezi slovy `do` a `done`.

Ukázka použití

```
select jazyk in anglictina nemcina rustina
do
      case $jazyk in
            anglictina)
                  echo "Vas oblibeny jazyk je anglictina.";;
            nemcina)
                  echo "Vas oblibeny jazyk je nemcina.";;
            rustina)
                  echo "Vas oblibeny jazyk je rustina.";;
            *)
                  echo "Chyba: Vyberte moznost 1 - 3";;
      esac
done
```

4.6 Souhrn

- Cyklus je řídící struktura, která na základě vyhodnocení podmínky umožňuje opakované provádění jednoho či více daných příkazů.
- Mezi cykly se řadí konstrukce `for`, `while`, `until` a `select`.
- Rozdíl mezi cyklem `while` a `until` je ten, že cyklus `until` proběhne vždy minimálně jednou.
- Cyklus `select` se využívá zejména pro vytváření interaktivních menu.

4.7 Příklady

Příklad 4.8.1

Zadání

Vytvořte skript, který bude přijímat dva vstupní argumenty. Oba tyto argumenty musí být platné adresáře. Poté pomocí cyklu for projděte první zadaný adresář a připojte koncovku „`duplicate`“ ke všem souborům v tomto adresáři, jejichž název se

vyskytuje i v adresáři druhém (tzn. soubor se stejným názvem se vyskytuje i ve druhém adresáři).

Řešení

```
#!/bin/bash
PRVNI_ADRESAR=$1
DRUHY_ADRESAR=$2
if [ $# -ne 2 ]
then
      echo "$(basename $0) prvni_adresar druhy_adresar"
      exit 1
fi
if [ ! -d $PRVNI_ADRESAR ]
then
      echo "Adresar $PRVNI_ADRESAR neexistuje"
exit 2
fi
if [ ! -d $DRUHY_ADRESAR ]
then
      echo "echo "Adresar $DRUHY_ADRESAR neexistuje""
exit 2
fi
for soubor1 in $PRVNI_ADRESAR/*
do
      if [ -f $ soubor1 ]
      then
            soubor2="$SRC/$(basename $soubor1)"
            if [ -f $soubor2 ]
            then
                  /bin/mv $soubor1 $soubor1.duplicate
            fi
      fi
done
```

Vysvětlení

1) Tento skript si nejprve načte vstupní argumenty do proměnných `PRVNI_ADRESAR` a `DRUHY_ADRESAR`.
2) Poté pomocí jednoduchých `if` konstrukcí zkontroluje, zdali počet zadaných argumentů odpovídá dvěma a zdali jsou to oba platné adresáře.
3) Pokud skript prošel úvodním ověřením, přechází k cyklu `for`.
4) Tento cyklus postupně projde všechny soubory a složky v prvním adresáři, přičemž v každém průchodu si aktuální hodnotu uloží do proměnné `soubor1`.
5) V těle cyklu se ověří, jestli `soubor1` je obyčejným souborem a následně také zdali stejný soubor existuje i ve druhém specifikovaném adresáři.
6) Pokud ano, k názvu souboru v prvním adresáři se přidá koncovka „`duplicate`“.

Příklad 4.8.2

Zadání

Vytvořte skript, který jako svůj jediný parametr bude přijímat jméno uživatele. Skript bude poté sledovat, zdali se daný uživatel nepřihlásil a pokud ano, tak jej opět odhlásí.

Poznámka

K tomu, aby bylo možné odhlásit jiného uživatele, je zapotřebí tento skript spouštět s právy správce systému (root).

Řešení

```
#!/bin/bash
if [ $# -ne 1 ]
then
      echo "$(basename $0) jmeno_uzivatele"
      exit 1
fi
```

```
until who | grep -s "$1" >/dev/null
do
      sleep 5
done
pkill -KILL -u "$1"
```

Vysvětlení

1) Na začátku se opět provede ověření na počet zadaných argumentů s případným zobrazením nápovědy.
2) Skript pokračuje konstrukcí `until`, která za použití příkazů `who` (slouží k vypsání aktuálně přihlášených uživatelů) a `grep` (slouží k vyfiltrování daného řetězce).

5 Funkce

Co se v této kapitole naučíte

- co jsou to funkce a proč je dobré je využívat
- jak je definovat a používat
- předávat funkci parametry
- co je vráceno jako výsledek funkce

5.1 Definice funkce

Funkce jsou velmi užitečným nástrojem v každém programovacím jazyce. Rozdělují skript do menších částí. Nadefinováním specifického úseku kódu jako funkce můžeme tento kód opakovaně využívat v různých částech skriptu, či dokonce ve skriptech jiných, což zvyšuje jeho konzistenci, čitelnost a správu.

Častým jevem je sdružování funkcí do knihoven, které tak poskytují sadu souvisejících funkcí na jednom společném místě. Tyto knihovny bývají často uloženy ve vlastním souboru a funkce v něm nadefinované jsou poté volány z vlastního skriptu.

5.2 Deklarace funkce

Stejně jako proměnné, tak i funkce musí být nadeklarovány předtím, než mohou být použity. Je dobrým, ale ne nutným zvykem definovat funkce na začátku souboru. Blok kódu definovaný jako funkce se skládá z názvu funkce následovaným párem kulatých a složených závorek.

Syntaxe

```
nazev_funkce(){ }
```

5.3 Volání funkce

Po nadeklarování funkce ji lze použít velmi lehkým způsobem. Stačí pouze zadat název této funkce.

Ukázka použití

```
vcerejsi_datum(){
      date --date='1 day ago';
}

vcerejsi_datum
```

5.4 Předávání parametrů funkci

Na rozdíl od většiny programovacích jazyků, které parametry funkce uvádějí v kulatých závorkách za názvem funkce, se v Bashi parametry funkce předávají podobně, jako je tomu u předávání parametrů skriptu, tedy pomocí speciálních proměnných k tomu určených.

Ukázka použití

```
vypis_argumenty(){
      echo "Prvni argument:" $1
      echo "Druhy argument:" $2
}

vypis_argumenty 5 10
```

5.5 Návratová hodnota

Funkce v Bashi na rozdíl od funkcí v jiných programovacích jazycích neumožňuje vracet jakoukoliv hodnotu. Po ukončení bashové funkce je navrácen pouze číselný status. Pokud je navrácena hodnota „0“ znamená to, že funkce

proběhla v pořádku. Jakákoliv jiná hodnota znamená neúspěch.

Pro ukončení funkce slouží klíčové slovo „`return`“ následované číselnou hodnotou. Pokud toto klíčové slovo není zadáno, je navrácen status na základě úspěchu či neúspěchu posledního příkazu v dané funkci.

5.6 Souhrn

- Funkce je sekce kódu, která může být opakovaně volána v průběhu skriptu.
- Její deklaraci provedeme zadáním jména funkce následovaným sérií kulatých a složených závorek.
- Použít nadeklarovanou funkci lze pouhým napsáním jejího názvu.
- Předávání parametrů funkci funguje na stejném principu, jako předávání parametrů skriptu, tedy pomocí speciálních proměnných `$1`, `$2` atd.
- Funkce v Bashi nevrací hodnotu, ale pouze jen číselný status, který určuje, zdali byla daná funkce provedena úspěšně či nikoliv.

5.7 Příklady

Příklad 5.7.1

Zadání

Vytvořte skript, který bude přijímat jeden vstupní argument. Tento argument bude kladné celé číslo. Poté pomocí libovolného cyklu projděte čísla od jedné do sta a pro každé toto číslo vypište informaci, zdali je beze zbytku dělitelné právě tímto zadaným číslem. Je-li číslo dělitelné zjišťujte pomocí samostatné oddělené funkce.

Řešení

```
#!/bin/bash
jeDelitelne(){
	return $(( $1 % $2 ))
}
for cislo in {1..100}
do
	if jeDelitelne $cislo $1
	then
		echo $cislo "je delitelne cislem" $1
	fi
done
```

Vysvětlení

1) Hned na začátku je nadeklarována funkce pro zjištění skutečnosti o tom, zdali je číslo beze zbytku dělitelné.
2) Tato funkce přijímá dva vstupní parametry, které jsou předány při jejím volání a jsou následně použity při výpočtu.
3) Pomocí cyklu `for` procházíme čísla od jedné do sta.
4) Pro každé číslo vyhodnocuje pomocí konstrukce `if` dělitelnost čísla.
5) Jako podmínka konstrukce `if` je uvedeno volání funkce určené pro zjištění dělitelnosti.
6) Pokud byla daná funkce v pořádku provedena, je podmínka vyhodnocena jako pravdivá a vypíše se informativní zpráva.

Příklad 5.7.2

Zadání

Vytvořte funkci s názvem „rozbal“, která bude přijímat jeden vstupní argument. Tento argument bude název existujícího souboru. Funkce poté na základě typu (koncovky) souboru rozhodne, jaký program pro dekompresi použije a daný archiv rozbalí do aktuálního adresáře.

Řešení

```
function rozbal()
{
    if [ -f $1 ] ; then
        case $1 in
            *.tar.bz2)   tar xvjf $1     ;;
            *.tar.gz)    tar xvzf $1     ;;
            *.bz2)       bunzip2 $1      ;;
            *.rar)       unrar x $1      ;;
            *.gz)        gunzip $1       ;;
            *.tar)       tar xvf $1      ;;
            *.tbz2)      tar xvjf $1     ;;
            *.tgz)       tar xvzf $1     ;;
            *.zip)       unzip $1        ;;
            *.Z)         uncompress $1   ;;
            *.7z)        7z x $1         ;;
            *)           echo "Neznamy typ souboru" ;;
        esac
    else
        echo $1 "není platny soubor"
    fi
}
```

Vysvětlení

1) Tato funkce nejprve zkontroluje, zda se jedná o obyčejný soubor. Pokud ano, pokračuje dále a pokud ne, tak zobrazí chybové hlášení a skript končí.
2) Pokud se jedná o soubor, tak skript pomocí konstrukce `case` a koncovky souboru vyhodnotí správný program pro dekompresi souboru a tento soubor rozbalí do aktuálního adresáře.
3) Pokud koncovka souboru není v seznamu, tak je zobrazeno chybové hlášení.

6 Neřešené úlohy

Příklad 6.1

Vytvořte skript, který bude kontrolovat sílu hesla. Vstupním argumentem bude heslo a výstupem bude oznámení, zdali je dostatečně silné, či nikoliv. Heslo musí být minimálně 8 znaků dlouhé, bude obsahovat malá i velká písmena, číslice a speciální znaky.

Příklad 6.2

Vytvořte skript, který bude mít jeden vstupní argument. Tento vstupní argument bude emailová adresa. Skript zkontroluje, jestli se jedná o platnou adresu.

Příklad 6.3

Vytvořte skript pro zálohování souborů, který po spuštění zobrazí na standardní výstup (obrazovku) menu s výběrem možností pro zazálohování daného typu souboru. Základně bude skript nabízet tyto tři možnosti: hudba, obrázky, dokumenty. Po výběru jedné z těchto možností bude uživatel požádán o zadání cesty.

Příklad 6.4

Vytvořte skript, který zkontroluje dostupné místo na pevném disku a pošle email v případě, že je zaplněn více, než je nadefinovaná hodnota.

Příklad 6.5

Vytvořte skript, který bude jako svůj jediný argument přijímat textový soubor. Tento skript poté pomocí cyklu while projde celý dokument a očísluje všechny řádky.

Příklad 6.6

Vytvořte skript, který bude jako svůj jediný argument přijímat cestu k adresáři. V tomto adresáři poté najde všechny textové soubory, které daný přihlášený uživatel vlastní a vypíše celkový počet všech znaků, slov a řádek v těchto souborech.

Příklad 6.7

Vytvořte skript, který zazálohuje všechny databáze na MySQL serveru. Název serveru a přihlašovací údaje budou přijímány jako parametry skriptu při jeho spouštění. Výsledná záloha bude jeden zkomprimovaný soubor uložený v domovském adresáři uživatele, který daný skript spustil.

Příklad 6.8

Za použití libovolného cyklu vytvořte skript, který vygeneruje následující výstup.

1

22

333

4444

55555

7 Příloha

7.1 Základní linuxové příkazy

Název	Účel
cat	zobrazí obsah souboru na standardní výstup
cd	změna adresáře
cp	zkopírování souboru/složky
df	zobrazení využitého místa na disku
ln	vytvoření symbolického odkazuje
locate	vyhledávání souborů/složek
logout	odhlášení ze systému
ls	vypsání obsahu složky
mv	přesunutí nebo přejmenování souboru/složky
pwd	zobrazí název aktuálního adresáře
who	zobrazení aktuálně přihlášených uživatelů
shutdown	vypnutí systému
mkdir	vytvoření adresáře
rm	odstranění souboru
rmdir	odstranění adresáře
chmod	změna oprávnění
cal	zobrazení kalendáře
kill	ukončení procesů

7.2 Systémové proměnné

Název	Účel
BASH_VERSION	verze Bashe
HOSTNAME	jméno počítače
HISTFILE	jméno souboru, kde je uložena historie příkazů
HISTFILESIZE	maximální počet záznamů v souboru s historií příkazů
HISTSIZE	počet příkazů, které budou uchovány v historii
HOME	domovský adresář přihlášeného uživatele
LANG	jazykové nastavení
PATH	seznam adresářů se spustitelnými programy a skripty
PS1	nastavení příkazové řádky
TERM	typ přihlašovacího terminálu
SHELL	cesta k shellu přihlášeného uživatele
EDITOR	výchozí textový editor
USER	jméno aktuálně přihlášeného uživatele

7.3 Možnosti příkazu test

Název	Účel
-d soubor	soubor existuje a je to adresář
-e soubor	soubor existuje
-f soubor	soubor existuje a je to normální soubor
-L soubor	soubor existuje a je to symbolický odkaz
-x soubor	soubor existuje a je spustitelný
-r soubor	soubor existuje a je čitelný
-s soubor	soubor existuje a má délku větší než nula
-w soubor	soubor existuje a je zapisovatelný