ČÁST I

Příručka uživatele

Originál: http://www.ibiblio.org/pub/Linux/docs/linux-doc-project/users-guide/

Tato příručka obsahuje vše, co potřebujete znát pro práci s operačním systémem Linux, což je volně šířitelná obdoba operačního systému Unix pro osobní počítače. Popisuje základní příkazy operačního systému Unix a rovněž se zabývá některými specifickými vlastnostmi Linuxu. Manuál je určen zejména pro začátečníky, avšak zkušenější uživatelé zde rovněž naleznou spoustu užitečných informací.

Poděkování

Autor by rád poděkoval následujícím lidem za jejich neocenitelnou pomoc při psaní a korekci této příručky. Jsou to:

Linus Torvalds, autor operačního systému Linux, který poskytl řadu podkladů.

Karl Fogel mi pomohl s vytvářením dokumentace k operačnímu systému Linux a napsal podstatné části kapitol 8 a 9.

Maurizio Codogno napsal podstatnou část kapitoly 11.

David Channon napsal dodatek A věnovaný editoru `vi`.

Společnost **Yggdrasil Computing, Inc.** poskytla štědrou podporu pro realizaci této příručky.

Společnost **Red Hat Software** rovněž dobrovolně podpořila realizaci této příručky.

Typografické konvence použité v této části

Tučné písmo	Používá se pro nové pojmy, varovné poznámky a klíčová slova u programovacích jazyků.
Italika	Používá se ke zdůraznění textu.
`Skloněné neproporcionální písmo`	Používá se k označení meta-proměnných, zejména v příkazovém řádku. Například „`ls -l foo`“, kde foo by mělo být nahrazeno jménem souboru, jako například `/bin/cp`.
`Neproporcionální písmo`	Používá se k reprezentaci textů vypisovaných na obrazovce. Dále se používá při výpisu kódů (zpravidla vytvořených v jazyku C), skriptů a výpisu obecných souborů, jako jsou konfigurační soubory. Aby nemohlo dojít k nedorozumění, budou tyto výpisy uváděny v rámečcích.
Výrazy v rámečku	Označují klávesy, které se mají stisknout. Uvidíte je nejčastěji ve formě: „Pro pokračování stiskněte klávesu Enter.“
	Tento symbol bude signalizovat zvláštní upozornění nebo nebezpečí. Odstavce takto označené čtěte se zvláštní pozorností.
	Tento symbol bude označovat text týkající se zvláštních pokynů pro uživatele systému X Window.
	Tímto symbolem jsou označeny odstavce, které obsahují důležité informace a měly by být čteny obzvlášť pečlivě.

KAPITOLA 1

Úvod

Kdo by si měl přečíst tuto knihu

Patříte mezi ty, kteří by si měli přečíst tuto knihu? Pokud neznáte přímou odpověď, pokuste se odpovědět na jiné otázky. Získali jste někde operační systém Linux a nevíte, co máte dělat dál? Jste uživateli jiného operačního systému než Unix a uvažujete o používání systému Linux? Chcete se dozvědět, co lze od tohoto operačního systému očekávat?

Máte-li k dispozici tuto knihu a přitom jste na některou z předcházejících otázek odpověděli „ano“, pak byste si ji měli přečíst. Každý, kdo si nainstaloval na svůj osobní počítač operační systém Linux (což je volně šiřitelná obdoba operačního systému Unix vytvořená panem Linusem Torvaldsem) a kdo neví, jak jej má efektivně využívat, by si měl tuto knihu přečíst. Je v ní je popsána většina základních příkazů operačního systému Unix a rovněž jsou zde uvedeny pokročilejší techniky pro práci se systémem. Rovněž se zmíníme o výkonném editoru GNU Emacs a několika dalších důležitých aplikacích.

Co byste měli udělat, než začnete tuto knihu číst

V této knize se předpokládá, že máte zajištěn přístup k operačnímu systému typu Unix a že tímto operačním systémem je Linux, tedy operační systém běžící na osobních počítačích s procesorem Intel.* Posledně jmenovaný předpoklad však není nezbytný – budeme upozorňovat na případy, kdy se ostatní unixové operační systémy od systému Linux liší.

Operační systém Linux je dostupný v mnoha formách, jež se nazývají distribuce. Předpokládá se, že máte nainstalovánu kompletní distribuci, jako je SlackWare, Redhat nebo Debian. Mezi jednotlivými distribucemi jsou jisté rozdíly, ale většinou nejsou důležité. Může se stát, že příklady uvedené v této knize se budou lišit od vaší distribuce. Ničeho se neobávejte, většinou půjde o odlišnosti zanedbatelné, se kterými si snadno poradíte. Pokud byste narazili na závažné rozdíly, neváhejte a napište o nich autorovi.

Pokud máte přístupová práva jako superuživatel (tedy například jako správce operačního systému nebo ten, kdo provedl instalaci), měli byste si pro sebe vytvořit normální uživatelský účet.

Potřebné informace najdete v instalační příručce. Jestliže nemáte přístupová práva jako super uživatel, pak požádejte o vytvoření účtu správce systému.

* Poznámka korektora: Operační systém Linux je v současné době k dispozici nejen pro procesory Intel, ale i SPARC, Alpha, ARM, PowerPC, pro stanice SGI a jiné.

Na čtení této knihy byste měli mít dostatek času a trpělivosti. Naučit se pracovat s operačním systémem Linux není jednoduché – většina lidí považuje ostatní operační systémy za jednodušší (jako například Macintosh Operating System). Jakmile se však naučíte základům, budete pracovat velmi snadno a efektivně. Unix je výkonný operační systém a umožňuje snadno provádět i velmi složité úlohy.

Navíc se v této knize předpokládá, že jste seznámeni se základní počítačovou terminologií. I když není tento předpoklad nezbytným, bude se vám kniha číst snadněji. Měli byste například vědět, co je to spustitelný program nebo co je to textový soubor. Pokud to nevíte, budete potřebovat někoho, kdo vám při studiu tohoto operačního systému poradí a pomůže.

Kdy se můžete čtení této knihy vyhnout

Jakýkoliv počítačový program se nejlépe naučíte tak, že si jej budete zkoušet na počítači. Většina lidí přijde na to, že čtení manuálů bez možnosti bezprostředně program nebo operační systém používat není příliš užitečné. Operační systém Unix se nejlépe naučíte tak, že jej budete používat. Nevyhýbejte se experimentování – můžete sice leccos pokazit, ale stále máte možnost provést novou instalaci. Většině havárií způsobených vaší nezkušeností se můžete vyhnout důsledným zálohováním.

Operační systém Unix nelze používat tak intuitivně, jako většinu ostatních operačních systémů. Proto byste si měli přečíst alespoň kapitoly 4, 5 a 6.

Jednu z možností, jak se vyhnout čtení této knihy, představují tzv. manuálové stránky, jež poskytují dokumentaci on-line.

Jak číst tuto knihu

Při čtení této knihy vám doporučujeme, abyste si bezprostředně vyzkoušeli to, co si právě přečtete. Pokud jste s některou tématikou již seznámeni, můžete příslušné odstavce přeskočit. Některá témata vám budou připadat zajímavá, jiná triviální, či dokonce nudná. Možná máte tolik sebejistoty, že budete některé příkazy zkoušet přesto, že přesně nevíte, jakou mají funkci. I to je způsob, jak se naučit pracovat s operačním systémem, i když poněkud riskantní.

Mnozí lidé ztotožňují operační systém Unix s příkazovým interpretem. Příkazový procesor je program, kterým lze řídit vše, co se v operačním systému Unix odehrává. Operační systém Unix je ve skutečnosti velmi složitý – většina jednoduše zadaných příkazů spustí spoustu procesů, o nichž uživatel zpravidla ani neví. V této knize probereme základní pravidla pro používání příkazového procesoru a také se zmíníme o důležitých programech, které jsou součástí operačního systému Unix nebo Linux.

Tato kapitola je spíše pseudokapitolou, která předkládá informace o obsahu knihy. Další kapitoly se zabývají tématy dle následujícího seznamu:

Druhá kapitola se zabývá vznikem operačních systémů Unix a Linux. Dále předkládá informace o nadaci Free Software Foundation a o projektu GNU.

Ve **třetí kapitole** je vysvětleno, jak začít a skončit práci s počítačem, co se stane při startování a při ukončení činnosti operačního systému. Většina těchto informací není pro práci s operačním systémem Linux důležitá. Jsou to ale informace užitečné a zajímavé.

Kapitola 4 představuje úvod k práci s příkazovým procesorem operačního systému Unix. Jedná se o prostředí, v němž se provádějí příkazy a spouštějí programy. Dozvíte se zde o základních příkazech a programech, které musíte znát, pokud máte používat operační systém Unix.

V **kapitole 5** se budeme zabývat systémem X Window. Ten představuje primární grafické uživatelské rozhraní pro operační systémy typu Unix a některé distribuce jej používají jako základní uživatelské prostředí.*

V **kapitole 6** se budeme zabývat pokročilejšími technikami při práci s příkazovým procesorem, zejména těmi, které vaši práci zefektivní.

V **sedmé kapitole** jsou stručně popsány příkazy operačního systému Unix. Čím více nástrojů budete umět používat, tím efektivnější bude vaše práce.

Kapitola 8 popisuje věhlasný editor Emacs. Jedná se o velký program, který většinu nástrojů operačního systému Unix integruje do jediného prostředí.

Devátá kapitola je věnována konfiguraci operačního systému Unix. Pomocí této konfigurace si můžete nastavit vlastnosti systému tak, aby se vám co nejlépe pracovalo.

V **desáté kapitole** se zmíníme o možnostech, které má uživatel operačního systému Unix k dispozici, chce-li komunikovat s ostatními počítači, případně se sítí Internet. Najdete zde informace o elektronické poště a o systému World Wide Web.

Jedenáctá kapitola popisuje některé komplikovanější příkazy.

V **kapitole 12** se dozvíte, jak se při práci s operačním systémem Unix nebo Linux snadno vyhnout chybám.

Dokumentace k operačnímu systému Linux

Tato uživatelská příručka (původní název zní: *„The Linux Users' Guide“*) je určena začátečníkům. V rámci projektu *„Linux Documentation Project“* se však připravují knihy pro pokročilejší uživatele.

Další knihy týkající se operačního systému Linux

Dokumentace k operačnímu systému Linux dále obsahuje tyto knihy: *„Installation and Getting Started“* – příručka popisující instalaci systému Linux. *„The Linux System Administrator's Guide“* – dokumentace týkající se správy operačního systému Linux. *„The Linux Kernel Hackers' Guide“* – kniha popisující jádro systému Linux a způsoby jeho modifikace. *„The Linux Network Administration Guide“* – příručka týkající se instalace, konfigurace a používání síťových spojení.

Praktické návody – dokumenty HOWTO

Kromě knih napsaných v rámci projektu *„Linux Documentation Project“* byla vytvořena řada krátkých dokumentů ve formě souborů týkajících se jednotlivých témat kolem operačního systému Linux. Tyto soubory s ev originále nazývají *„HOWTO“* a do této knihy byly zařazeny na dvě desítky nejzajímavějších a nejdůležitějších z nich. Najdete je v části IV jako *„Praktické návody“*. Například dokument *„Sdílení modemů“* popisuje možnosti sdílení modemu pod Linuxem.

Soubory s praktickými návody jsou dostupné v několika formách – jako ucelené knihy (například *„The Linux Bible“* nebo *„Dr. Linux“*), nebo je můžete získat prostřednictvím diskusní skupiny comp.os.linux.answers. Soubory se rovněž nacházejí na mnoha serverech FTP nebo WWW. Centrálním místem obsahujícím informace o operačním systému Linux je *http://www.linux.org*.

---

* Poznámka korektora: V současné době používá systém X Window většina používaných distribucí operačního systému Linux.

Co je to „Linux Documentation Project"?

Cílem projektu „*Linux Documentation Project*" je shromáždit, utřídit a v jednotné formě prezentovat dokumentaci týkající se operačního systému Linux. Pracují na něm lidé po celém světě. Impuls k projektu dal Lars Wirzenius.

Předpokládá se, že v rámci projektu „*Linux Documentation Project*" budou postupně vydány knihy týkající se všech témat kolem operačního systému Linux. Pokud budete mít jakékoliv náměty na zlepšení této dokumentace, dejte prosím vědět autorovi této knihy.

Operační systémy

Primární funkce operačního systému spočívá v tom, že poskytuje podporu pro realizaci počítačových programů. Proto například můžete používat svůj oblíbený editor a vytvářet dokumenty. Bez podpory operačního systému by editor nemohl pracovat – editor potřebuje ke své činnosti interakci s vaším terminálem, s vašimi soubory a dalším technickým vybavením počítače.

Nyní si položme otázku: Proč je nutné psát a číst knihy o operačních systémech, které jsou pouhou podporou vašich aplikací? Každý operační systém obsahuje spoustu programů pro správu, konfiguraci, řízení spojení s ostatními systémy, zálohování a podobně. Pokud jde o operační systém Linux, najdete zde řadu „miniaplikací", jež vám pomohou pracovat efektivněji. Jestliže tedy nepoužíváte váš počítač pouze k práci s jednou nebo několika málo aplikacemi, je znalost operačního systému velmi důležitá.

Operační systém (nejčastěji označován zkratkou „OS") může být jednoduchý a minimalizovaný (například DOS) nebo velký a složitý (například OS/2 nebo VMS). Operační systémy typu Unix patří ke středně velkým systémům. Poskytují o něco více zdrojů a prostředků než první jednoduché operační systémy, a to v takové formě, aby s jejich pomocí bylo možné řešit prakticky všechny úlohy.

Původně byl operační systém Unix navržen jako zjednodušení operačního systému Multics. Filosofie operačního systému Unix spočívá v tom, že by se veškeré funkce měly rozdělit do malých částí, tedy relativně jednoduchých programů.[1] Nové funkční vlastnosti lze získat vhodnou kombinací těchto jednoduchých programů. Samozřejmě se stále objevují nové a nové obslužné programy, které lze snadno integrovat do vašich nástrojů, a tak můžete váš operační systém neustále rozšiřovat. Když jsem například psal tento dokument, používal jsem následující programy: `fvwm` pro obsluhu a konfiguraci systému X Window, editor `Emacs` pro vytvoření textu, `xdvi` pro prohlížení textu před tiskem, `LATEX` pro formátování textu, `dvips` pro přípravu tisku a `lpr` pro vlastní tisk. Kdyby například existoval jiný program pro prohlížení textu před tiskem, pak bych jej mohl použít, aniž bych změnil ostatní programy. V tomto okamžiku na mém počítači běží současně třicet osm programů (většina z nich se nachází v neaktivním stavu „sleep", dokud nebudou aktivovány k provedení nějakého úkolu).

Při používání operačního systému Unix budete jistě chtít minimalizovat množství práce potřebné k realizaci každého běžně prováděného úkolu. Operační systém Unix vám pro tento účel nabízí spoustu nástrojů. Abyste je mohli efektivně používat, budete muset poměrně přesně vědět, jakou funkci každý nástroj má. Pokud byste nad uskutečněním každého jednoduchého úkolu strávili ho-

1 Tato filosofie byla ve skutečnosti určena technickým vybavením počítačů, na kterých běžely první verze operačního systému Unix. Časem se ukázalo, že Unix dobře funguje i na počítačích s vyspělejším technickým vybavením. I dnes, po dvaceti pěti letech, je původní filosofie operačního systému Unix zcela vyhovující.

dinu nebo dokonce více, pak by vaše práce nebyla příliš produktivní. V této knize se dozvíte, jak a v které situaci využívat nástroje obsažené v operačním systému Unix a jak tyto nástroje kombinovat za účelem řešení složitějších úloh.

Klíčovou částí operačního systému je tzv. **jádro**. Ve většině operačních systémů, jako je Unix, OS/2 nebo VMS, plní jádro systému takové funkce, jako je spouštění programů, přidělování systémových zdrojů, přidělování času procesoru současně běžícím programům a podobně. Samozřejmě platí, že i jádro systému je program. Ten běží na vašem počítači jako první program po jeho nastartování, kdy realizuje všechny konfigurační funkce, a jako poslední program před vypnutím počítače, kdy realizuje všechny potřebné funkce k zastavení systému.

KAPITOLA 2

Co je Unix

Historie operačního systému Unix

V roce 1965 pracovaly společnosti Bell Telephone Laboratories (divize AT&T) a General Electric na projektu „MAC of MIT“, jehož cílem bylo vytvořit operační systém Multics. Později se společnost Bell Telephone Laboratories rozhodla od spolupráce odstoupit, ale v důsledku toho neměla k dispozici kvalitní operační systém.

Pánové Ken Thompson a Dennis Ritchie se rozhodli navrhnout operační systém, který by společnosti Bell Telephone Laboratories vyhovoval. Ken Thompson tento návrh realizoval při vytváření vývojového prostředí na počítači PDP-7. Další výzkumný pracovník společnosti Bell Telephone Laboratories, pan Brian Kernighan, dal novému operačnímu systému název Unix.

Později zveřejnil pan Dennis Ritchie programovací jazyk C. V roce 1973 byl Unix kompletně přepsán do jazyka C (původní systém byl vytvořen v assembleru[1]). V roce 1977 byl operační systém Unix převeden z počítače PDP na nový počítač s použitím procesu, jež se nazývá „**porting**“. Tato akce byla uskutečnitelná právě proto, že byl operační systém Unix přepsán v jazyce C.

Koncem sedmdesátých let byla společnosti AT&T protimonopolním úřadem zakázána činnost v oblasti počítačového průmyslu. Proto se společnost rozhodla za velmi výhodných finančních podmínek převést licenci na operační systém Unix na některé university. Unix se tedy stal populárním především v akademických kruzích, avšak postupem času se začal prosazovat i v komerční sféře. Dnešní podoba Unixu se zcela liší od verze z roku 1970. Existují dvě základní varianty: System V od společnosti USL (Unix System Laboratories, dnes jej vlastní Novell[2]) a BSD (Berkeley Software Distribution).

Poslední verze USL má označení SVR4[3] (čtvrtá verze), zatímco poslední verze od BSD má označení 4.4. Kromě těchto základních verzí však existuje spousta dalších verzí operačního systému Unix. Komerční verze jsou zpravidla odvozeny od jedné z verzí USL nebo BSD. Existuje však spousta verzí operačního systému Unix, které kombinují vlastnosti obou základních verzí.

Ceny současných komerčních verzí operačního systému Unix pro počítače s procesorem Intel se pohybují od 500 do 2000 dolarů.

1 Assembler je základní programovací jazyk, který je úzce spjat s konkrétním typem počítače. Programy napsané v jazyce assembler zpravidla nejsou přenositelné mezi různými počítačovými platformami.

2 Tato verze operačního systému Unix byla nedávno prodána společnosti Novell. Předtím byla verze USL vlastněna společností AT&T.

3 SVR4 je zkratkou pro „System V“ verze 4. (System V Release 4)

Historie operačního systému Linux

Autorem operačního systému Linux je pan Linus Torvalds. Jeho původní verze byla v průběhu asi deseti let zdokonalována bezpočtem lidí na celém světě. Linux představuje verzi operačního systému Unix pro osobní počítače s procesorem Intel, Alpha a další a byl kompletně vytvořen znovu – na jeho vývoji se společnosti Unix System Laboratories a Berkeley Software Distribution vůbec nepodílely. Zajímavé je, že se na vývoji operačního systému Linux podíleli lidé na celém světě – od Austrálie po Finsko a všichni doufali, že se Linux podaří uvést do podoby schopné konkurovat ostatním operačním systémům typu Unix.

Vlastní projekt operačního systému Linux začal výzkumem vlastností procesoru 386. Systém je navržen tak, aby maximálně využíval všech vlastností tohoto procesoru.

Na operační systém Linux se vztahují licenční podmínky GPL (GNU General Public Licence). Tato licence se vztahuje na veškeré programové vybavení produkované nadací Free Software Foundation a jejím cílem je zabránit komukoliv omezovat distribuční práva ostatních. Jinými slovy, podle licenčních podmínek GPL si můžete účtovat za distribuci programového vybavení GNU kolik chcete, ale nesmíte nikomu nařizovat, za jaký poplatek je má distribuovat dál. Dále musíte s každou distribucí programového vybavení GNU zpřístupnit zdrojové kódy[4], což je velmi užitečné pro programátory. Pak si totiž každý může například modifikovat operační systém Linux a dále distribuovat tuto modifikovanou verzi – opět za předpokladu, že ji bude distribuovat podle licenčních podmínek GPL.

Operační systém Linux podporuje většinu programového vybavení napsaného pro Unix, včetně systému X Window. Tento systém byl vytvořen na Massachusetts Institute of Technology tak, aby umožňoval operačním systémům Unix vytvářet grafická okna a interaktivně spolupracovat mezi sebou. Dnes platí, že je systém X Window implementován pro každou verzi operačního systému Unix.

Kromě standardů, které představují verze System V a BSD, existuje sada standardizačních dokumentů publikovaná úřadem pro standardizaci IEEE, která má označení **POSIX**. Operační systém Linux splňuje většinu podmínek uvedených v dokumentech POSIX-1 a POSIX-2. Přitom má v mnoha aspektech vlastnosti podobné systému BSD, ale má také vlastnosti, které najdete u systému System V. Stručně řečeno, pokud jde o standard, představuje operační systém Linux kombinaci (a většina lidí se domnívá, že velmi dobrou) všech tří standardů.

Většina programového vybavení dodávaná s operačním systémem Linux pochází od nadace Free Software Foundation a je součástí projektu GNU. Hlavním cílem projektu GNU je jednak zpřístupnit programové vybavení původně napsané pro operační systém Unix uživatelům ostatních operačních systémů, jednak vytvořit přenositelný operační systém, který bude mít prakticky stejné vlastnosti jako Unix. Přenositelný znamená, že tento operační systém poběží na všech počítačových platformách (ať jsou vybaveny procesorem Intel, PowerPC, Alpha či jiným). Tento operační systém má označení Hurd. Hlavní rozdíl mezi operačním systémem Linux a operačním systémem Hurd nespočívá v uživatelském rozhraní, ale v programátorském rozhraní – Hurd představuje moderní operační systém, zatímco Linux je operačním systémem založeným na filosofii stařičkého Unixu.

4 Zdrojový kód programu je textový soubor obsahující příkazy pro konkrétní programovací jazyk. Příslušným překladačem jej lze přeložit do strojového kódu pro konkrétní počítač. Strojový kód je soubor, který již není snadno čitelný a modifikovatelný tak jako zdrojový kód.

Předcházející odstavce by mohly vzbudit dojem, že se o operační systém Linux stará pouze pan Linus Torvalds. V raných dobách existence tohoto systému se například o překladač GCC (základní překladač jazyka C pro Linux) a o knihovny Linux C staral H. J. Lu. V každé distribuci operačního systému Linux naleznete v adresáři `/usr/src/linux/` soubor `CREDITS` obsahující seznam všech lidí, kteří se významnou měrou podíleli na jeho vývoji.

Dnešní podoba operačního systému Linux

První číslo ve verzi operačního systému Linux představuje číslo hlavní revize. V době, kdy byla napsána tato kniha (únor 1996) je k dispozici teprve první verze. Druhé číslo představuje méně podstatné revize. Pokud je druhé číslo sudé, jedná se o stabilní verzi. Jestliže je liché, jedná se o vývojovou verzi. Ve vývojových verzích bývá spousta chyb, které „odvážní“ uživatelé postupně odhalují a programátoři postupně opravují. Jakmile jsou všechny závažné nedostatky z vývojové verze odstraněny, prohlásí se vývojová verze za stabilní a začne se pracovat na nové vývojové verzi. V únoru 1996 měla stabilní verze číslo 1.2.11 a poslední vývojová verze 1.3.61.*

Operační systém Linux je velkým systémem obsahujícím spoustu chyb, které se postupně odstraňují. Ve stabilních verzích se však vyskytují chyby velmi zřídka a souvisejí hlavně s nestandardním technickým vybavením počítače. Doposud jsme hovořili pouze o chybách jádra systému. Operační systém však zdaleka není jen jádro systému, ale obsahuje spoustu obslužných programů.

Ani velmi zkušený uživatel často není schopen určit, zda je původ chyby v obslužném programu nebo v jádře systému. Také je velmi nesnadné rozeznat, zda jsou některé „divné“ věci chybou operačního systému, nebo novou vlastností. Doufáme, že vám tato kniha pomůže orientovat se právě v takových situacích.

Několik málo otázek a odpovědí na ně

Než se pustíme do další kapitoly, odpovězme si na několik důležitých otázek.

Otázka: Jak se má vyslovovat Linux?

Odpověď: Podle autora operačního systému Linux se má samohláska „i“ vyslovovat krátce (jako například ve slově minimum). Linux by se měl rýmovat se slovem Minix, což je další verze operačního systému Unix. Samohláska „u“ by se měla vyslovovat ostře, jako například ve slově „rule“, a nikoliv měkce, jako například ve slově „ducks“. Obecně by se mělo slovo Linux rýmovat se slovem „cynics“. V našich zemích se Linux vyslovuje „linuks“.

Otázka: Proč by se mělo pracovat s operačním systémem Linux?

Odpověď: Proč ne. Linux je obecně levnější než ostatní operační systémy a je méně problematický než většina komerčních operačních systémů. Nemusí být pravda, že zrovna Linux je tím nejlepším operačním systémem pro vaši konkrétní aplikaci. Pokud má však někdo zájem o používání operačního systému typu Unix na osobním počítači a o aplikace vytvořené pro Unix, pak je Linux pravděpodobně nejlepším vysoce výkonným systémem.

Komerční programové vybavení pro operační systém Linux

Pro operační systém Linux existuje spousta komerčního programového vybavení. Nejdůležitější roli hraje systém Motif, což je uživatelské rozhraní pro systém X Window připomínající Microsoft Windows. Na základě systému Motif byla vytvořena řada komerčního programového vybavení.

* Poznámka korektora: V současné době (únor 2003) je poslední stabilní verze 2.4.20 a vývojová verze 2.5.59.

Dnes můžete koupit prakticky cokoliv ve verzi pro operační systém Linux – od oblíbeného textového procesoru Word Perfect až po Maple, univerzální nástroj pro matematické a fyzikální modelování.

Možná se budete divit, jak lze skloubit operační systém Linux, jenž je distribuován podle licenčních podmínek GPL (GNU General Public Licence, viz dodatek B), s komerčním programovým vybavením. Podmínky GPL platí pouze pro jádro systému, zatímco na ostatní aplikace vytvořené pro Linux se vztahují jiné podmínky GNU, „*Library General Public Licence*“ (viz dodatek B). Podle těchto podmínek smějí poskytovatelé programového vybavení prodávat své aplikace a přitom nemusejí distribuovat zdrojové kódy.

Uvědomte si prosím, že uvedené dva dokumenty představují něco jako autorská práva, ale v žádném případě nepředstavují licenci na užívání. Tyto dokumenty nikterak nevymezují způsoby, kterými můžete dané programové vybavení používat, ale vymezují pravidla, kterými se musíte řídit při distribuci tohoto programového vybavení. V tom spočívá hlavní myšlenka filosofie nadace Free Software Foundation a platí i pro Linux: na používání operačního systému Linux neexistuje žádná licence.

KAPITOLA 3

Začínáme pracovat s operačním systémem Linux

Pravděpodobně máte jisté zkušenosti s používáním jednouživatelského operačního systému, jako je DOS nebo OS/2. Chcete-li pracovat v těchto operačních systémech, nemusíte se identifikovat – předpokládá se, že jste jediným uživatelem operačního systému, který má přístup ke všem jeho zdrojům. Na druhé straně, operační systém Unix je víceuživatelským systémem. To znamená, že v daném okamžiku může mít k systému přístup více uživatelů najednou.

Aby mohl operační systém Unix komunikovat s každým uživatelem odděleně, musí jej identifikovat procesem, který se nazývá „**přihlášení se**“ (logging in). Když zapnete počítač, proběhne spousta inicializačních procesů a až potom je počítač schopen komunikovat s uživatelem. Protože je tato příručka věnována operačnímu systému Linux, vysvětlíme si, co se odehrává v průběhu zavádění tohoto operačního systému Linux.

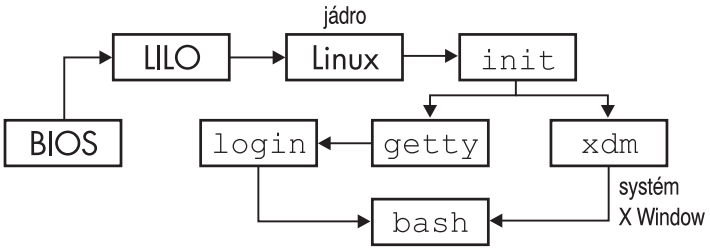
Pokud nepoužíváte operační systém Linux na osobním počítači s procesorem Intel, pak se vás nebudou některé informace v této kapitole týkat. Většinu užitečných informací najdete v oddílu Zapnutí počítače.

Jestliže se pouze zajímáte o používání vašeho počítače, pak kromě oddílu Činnost uživatele nemusíte tuto kapitolu číst.

Zapnutí počítače

Po zapnutí počítače s procesorem Intel se jako první provede program zvaný BIOS. BIOS je zkratkou pro Basic Input/Output System. Tento program je trvale uložen ve zvláštní paměti počítače. Realizuje některé základní testy a identifikuje technické vybavení, jako je pevný disk či disketová jednotka. Po identifikaci disku najde tzv. zaváděcí sektor (boot sector) a spustí kód, který v něm nalezne. Jestliže byl nalezen disk, ale žádný zaváděcí sektor, vypíše BIOS následující hlášení:

```
Non-system disk or disk error
```



Obrázek 3.1 – Cesta, po které dospěje osobní počítač s procesorem Intel k příkazovému řádku. Proces `init` může, ale také nemusí realizovat zavedení systému X Window. Pokud systém X Window zavádí, spustí nejdříve program `xdm`. Jinak spustí příslušný počet procesů `getty`.

Pak ovšem stačí disk bez zaváděcího sektoru vyjmout, stisknout kteroukoliv klávesu a zaváděcí proces bude pokračovat.

Pokud není v disketové jednotce vložena disketa, BIOS vyhledá hlavní zaváděcí záznam MBR (master boot record) na pevném disku (závisí na nastavení BIOSu). Ten obsahuje kód (zaváděcí program), jehož úkolem je zavést operační systém. Pro operační systém Linux se tento program jmenuje LILO (LInux LOader). Linux Loader je nejpoužívanějším bootovacím programem pro Linux, ale je možno použít i GRUB Loader, SYSLinux nebo i NT Loader. To znamená, že program LILO je uložen právě v diskové oblasti MBR. Nyní budeme předpokládat, že se zavádí operační systém Linux. (U vaší konkrétní distribuce může proces zavádění operačního systému Linux probíhat poněkud odlišně – proto si pečlivě pročtěte dokumentaci k vaší distribuci. Další užitečné informace naleznete v dokumentaci k zaváděcímu programu LILO.)

Jak Linux přebírá kontrolu nad počítačem

Nejdříve tedy předá BIOS kontrolu zaváděcímu programu LILO a ten pak předá kontrolu **jádru** operačního systému Linux (Linux kernel). Jádro je centrálním programem celého systému a řídí všechny ostatní programy. V prvním kroku jádro realizuje přepnutí procesoru do tzv. chráněného módu. Procesor 80386[1], jež řídí váš počítač, může pracovat ve dvou módech: v tzv. „reálném módu“ a v tzv. „chráněném módu“. Operační systém DOS, stejně jako BIOS, funguje v reálném módu. Vyspělejší operační systémy však pracují v chráněném módu. Proto platí, že Linux po svém zavedení zcela nahradí BIOS.

Pokud jde o počítače s jinými procesory, probíhá tato fáze zavádění systému Linux poněkud jinak. Zpravidla nedochází k přepínání do chráněného módu a jen několik málo procesorů vyžaduje komplikované zavádění operačního systému podobné kombinaci BIOS – LILO. Po zavedení jádra pracuje Linux na všech počítačích stejně bez ohledu na procesor.

V dalším kroku se Linux pokouší identifikovat technické vybavení počítače, na kterém běží. Musí znát informace o pevném disku, zda je nebo není k dispozici myš, zda je nebo není váš počítač připojen k počítačové síti a podobně. Linux si není schopen po vypnutí počítače tyto údaje pamatovat, proto je pokaždé zjišťuje znovu. Naštěstí nemusíte údaje o technickém vybavení vašeho počítače zadávat – Linux si je zjistí sám.

1 Pod pojmem procesor 80386 budeme dále rozumět všechny 32bitové procesory Intel, tedy i 80486, Pentium, Pentium Pro, Pentium MMX a další. Místo 80386 budeme také používat označení 386.

V průběhu zavádění vypisuje operační systém Linux řadu hlášení. Podrobněji se jimi budeme zabývat v oddílu Hlášení jára systému. Asi by se vám nelíbilo, pokud byste museli při každém startu odpovídat na spoustu dotazů týkajících se technického vybavení a konfigurace vašeho počítače. Na několik důležitých dotazů tohoto druhu však budete muset odpovědět při instalaci systému. Budete-li mít jakékoliv problémy, pročtěte si dokumentaci k vaší distribuci.

Jak jsme se již zmínili, jádro řídí všechny ostatní programy. Jestliže tedy úspěšně identifikuje veškeré technické vybavení počítače, spustí jiný program, který již začne dělat něco užitečného. Tento program má název `Proces init`. (Všimněte si, že pro název programu jsme použili jiný font. `Neproporcionální font` budeme používat k označení programů, adresářů i obecných souborů.) Po nastartování programu `Proces init` se jádro stane jakýmsi „manažerem“, avšak již není aktivním programem.

Jestliže chceme vědět, co dělá počítač po zavedení jádra, musíme studovat Proces init. Ten prochází mnoha komplikovanými stádii, která nejsou na všech počítačích stejná. Operační systém Linux má mnoho verzí programu `Proces init` – každá z nich realizuje své cíle různými způsoby. Také záleží na tom, zda je váš počítač zapojen do sítě a jakou distribuci operačního systému Linux používáte. Nyní si uveďme některé základní procesy, jež `init` realizuje:

- Zpravidla proběhne kontrola souborového systému. Co je to souborový systém? Je to způsob organizace a ukládání souborů na pevném disku. Linux při kontrole souborového systému zjišťuje, které části disku jsou již použity a které jsou volné. Souborový systém se podobá rejstříku nebo štítkovému katalogu v knihovně. Bohužel se stává, že například při výpadku elektrické energie dojde k poškození některých souborů a pak je používání některých částí pevného disku konfliktní. `Proces init` proto spouští další program označený jako `fsck`, jenž se pokouší vadné soubory a konfliktně obsazené části pevného disku opravit.
- Pokud je váš počítač připojen k síti, spustí se několik speciálních programů, jež zajišťují komunikaci vašeho počítače s jinými počítači.
- Na pevném disku mohou zůstat některé dočasné soubory, které jsou jinak neužitečné. Ty mohou být programem `Proces init` zrušeny.
- Zpravidla také dojde k aktualizaci systémového času. Operační systém Unix předpokládá, že je čas definován jako UTC (Universal Coordinated Time), známý také jako střední greenwichský čas. Přitom je čas ve vašem počítači pravděpodobně nastaven na lokální – to znamená, že některý program musí přečíst lokální čas nastavený ve vašem počítači a korigovat jej na čas UTC.

Jakmile `Proces init` ukončí všechny uvedené aktivity, přejde do stavu, jenž lze označit jako „rodič“ všech procesů v systému Unix. Pod pojmem proces si lze jednoduše představit běžící program. Protože jeden program může být spuštěn dvakrát nebo vícekrát, mohou existovat dva procesy nebo více procesů realizujících konkrétní program.

V operačním systému Unix tedy platí, že proces je instancí programu. Vytváří se systémovým voláním (službou poskytovanou jádrem systému) nazvaným „fork“ (rozvětvení). Pojem „fork“ se používá proto, že se původně jeden proces při spuštění dalšího procesu rozdělí na dva oddělené procesy. Typickým příkladem programu, který běží jako několikanásobný proces, je program `getty`. `getty` je důležitý program, jenž voláním programu `login` umožňuje uživateli přihlásit se do systému.

Činnost uživatele

Jak se přihlásit do systému

Než začnete používat operační systém Unix, musíte se identifikovat. Procesu identifikace se říká „**přihlášení do systému**“ (`login`). Jedná se o proces, ve kterém se operační systém přesvědčí, zda má uživatel oprávnění systém používat. Systém se vás zeptá na jméno účtu (account name) a heslo (password). Jméno účtu je zpravidla podobné vašemu jménu – pravděpodobně jste jej obdrželi od vašeho systémového správce nebo jste si vytvořili vlastní, pokud jste sami systémovým správcem. (Informace o vytváření jména účtu naleznete v příručce *„Příručka správce operačního systému Linux“.)*

Po dokončení všech zaváděcích procesů byste měli na svém monitoru spatřit hlášení podobné následujícímu. Nejnovější verze Linuxu se hlásí takto (první řádek je pouze uvítací zprávou – může zde být cokoliv jiného):

```
RedHat Linux release 5.1 (Manhattan) with all updates. Kernel 4.1.108 on an Intel
mousehouse login:
```

Je však možné, že vás systém uvítá něčím úplně jiným. Místo nudné textové obrazovky se vám může představit nějaká grafická obrazovka. Avšak i v tomto případě budete vyzváni k přihlášení se do systému a proběhnou stejné procedury identifikace uživatele. Pokud se vám objevila zmíněná grafická obrazovka, pak jste zřejmě uživateli systému X Window. To znamená, že budete pracovat v systému oken, o kterém se podrobněji zmíníme v kapitole 5. Všimněte si, že veškerý výklad o systému X Window bude v této knize označován velkým písmenem X na levém okraji textu.

Jako jméno účtu budeme používat smyšlené jméno `larry`. Kdykoliv uvidíte v následujícím textu jméno `larry`, měli byste místo něj dosadit své vlastní jméno účtu. Jména účtu by měla být založena na skutečném jménu uživatele; ve větších operačních systémech typu Unix se zpravidla používají příjmení uživatelů, kombinace jména a příjmení nebo kombinace jména a číslic. Pro uživatele se jménem Larry Greenfield by se mohly použít například tyto kombinace: `larry`, `greenfie`, `lgreenfi` nebo `lg19`.

Pro bližší vysvětlení, jméno `mousehouse` je „jménem“ počítače, na kterém pracuji. Je pravděpodobné, že jste při instalaci operačního systému Linux použili jiné, podobně duchaplné jméno. Jméno počítače není důležité; v této knize budeme používat již uvedené jméno `mousehouse`, případně `lionsden`. Jestliže tedy zadáte své jméno účtu a stisknete klávesu Enter, objeví se výzva k zadání hesla:

```
mousehouse login: larry
Password:
```

Na druhém řádku očekává operační systém Linux zadání hesla (password). Při zadávání hesla neuvidíte na obrazovce, co píšete, proto zadávejte heslo pečlivě. Chybně zadaný znak můžete zrušit, ale opět nebudete vidět, který znak jste zrušili. Jestliže se na vás zrovna někdo dívá, nezadávejte heslo příliš pomalu. Mohlo by tak dojít k vyzrazení vašeho hesla a nepovolaná osoba by jej mohla zneužít. Jestliže zadáte heslo chybně, budete mít možnost zadat je znovu.

Pokud jste zadali heslo správně, objeví se zpravidla na vaší obrazovce nějaká zpráva, jež se většinou nazývá „zpráva dne“. Tato zpráva může obsahovat cokoliv – její znění zadává systémový správce.

Jako další se objeví tzv. výzva příkazového řádku (prompt). Výzva příkazového řádku je řetězec indikující skutečnost, že systém očekává zadání příkazu. Mohla by vypadat takto:

```
/home/larry#
```

Pokud jste uživateli systému X Window, pak zřejmě uvidíte výzvu podobnou výše uvedené v nějakém okně na obrazovce. Okno je obdélníková orámovaná oblast obrazovky, která v tomto případě imituje terminál. Pokud chcete zadat příkaz v příkazovém řádku v okně, musí být okno aktivní – stačí například do tohoto okna přesunout kurzor myši.

Jak ukončit práci s počítačem

Chcete-li skončit práci s počítačem, pak rozhodně nestačí pouze počítač vypnout. Pokud to uděláte v operačním systému Linux, s největší pravděpodobností ztratíte nějaká data nebo poškodíte nějaké důležité soubory.

Na rozdíl od operačního systému DOS, kde lze počítač vypnout vypínačem nebo restartovat tlačítkem Reset, je v operačním systému Linux nutné provést řadu kroků, které zajistí bezpečné ukončení systému. Uveďme si hlavní důvod. Operační systém Linux používá tzv. **paměť cache** pro diskové operace. To znamená, že jistá část souborů uložených na pevném disku je umístěna do paměti RAM[3]. K synchronizaci mezi pamětí RAM a pevným diskem dochází přibližně každých třicet sekund. Jestliže se má vypnout nebo restartovat počítač, pak je nutné, aby se část paměti RAM obsahující disková data uložila.

Máte-li tedy ukončit práci s počítačem a přitom jste normálně přihlášeni (t.j. zadali jste jméno uživatelského účtu a heslo), pak se musíte odhlásit. K tomuto účelu slouží příkaz `logout`. Napište v příkazovém řádku `logout` a stiskněte klávesu Enter. Klávesou Enter se odesílá každý příkaz. Než tuto klávesu stisknete, můžete chybně zadané znaky opravovat, případně celý příkaz zrušit a zadat nový.

```
RedHat Linux release 5.1 (Manhattan) with all updates.
Kernel 4.1.108 on an Intel
mousehouse login:
```

Nyní se může do systému přihlásit další uživatel.

Vypnutí počítače

Jste-li jediným uživatelem počítače, pak po odhlášení můžete počítač odpojit od zdroje elektrické energie.[4] V takovém případě se přihlašujte jako uživatel se speciálním jménem účtu root. Účet root je účtem systémového správce, který má zajištěn přístup ke každému souboru v systému. Pokud hodláte vypnout počítač, pak musíte obdržet heslo od systémového správce. (Jste-li jediným uživatelem počítače, pak jste systémovým správcem právě vy! Proto nikdy nezapomeňte heslo!)

---

3 Rozdíl mezi pamětí RAM a pevným diskem lze přirovnat ke krátkodobé a dlouhodobé paměti. Po vypnutí počítače se veškeré informace v paměti RAM ztratí, zatímco informace uložené na pevném disku zůstávají uchovány. Na druhé straně platí, že přístup k informacím na pevném disku je nesrovnatelně pomalejší, než přístup k informacím v paměti RAM.

4 Chcete-li šetřit některé komponenty technického vybavení vašeho počítače, pak vypínejte počítač jen když nebudete na počítači pracovat delší dobu (například večer). Časté zapínání a vypínání počítače zbytečně namáhá některé jeho součásti.

Přihlášení a odhlášení by mohlo vypadat takto:

```
mousehouse login: root
Password:
Last login: Sun 5 11:14:57 on tty1
/# shutdown -h now

Broadcast message from root (tty1) Sun 14:52:32 1998...

The system is going down for system halt NOW!
INIT:Switching to runlevel: 0
INIT: sending processes the TERM signal
The system is halted
System Halted
```

Po zadání příkazu `shutdown -h now` proběhnou jisté procedury, které připraví systém na vypnutí nebo reset.

Na závěr ještě uveďme krátkou poznámku pro pohodlnější uživatele. Místo procesu přihlašování a odhlašování jako uživatel root lze použít příkaz `su`. Jako běžný uživatel zadejte v příkazovém řádku příkaz `su` a stiskněte klávesu Enter. Systém si vyžádá heslo uživatele root a pak vám přidělí všechna jeho privilegia. Nyní můžete bez problémů systém ukončit příkazem `shutdown -h now`.

Hlášení jádra systému

Když poprvé startujete počítač, objeví se na obrazovce spousta hlášení popisující technické vybavení vašeho počítače. Uvedená hlášení vypisuje jádro operačního systému Linux. V tomto oddílu si některá důležitá hlášení podrobněji popíšeme.

Přirozeně platí, že se hlášení jádra systému budou lišit, což je dáno jednak typem počítače a jednat distribucí operačního systému Linux. V následujícím textu budou popsána hlášení platná pro můj počítač. Některá mají obecnou platnost, jiná jsou více specifická. Musím poznamenat, že můj počítač má minimální konfiguraci, pokud jde o technické vybavení, proto dále neuvidíte příliš mnoho informací týkajících se speciálního technickéh vybavení. Následující hlášení pocházejí z verze operačního systému Linux 1.3.55. Tato verze patří v době, kdy píši tuto knihu, k nejnovějším.

1. V prvním kroku operační systém Linux identifikuje grafickou kartu, aby mohl vybrat nejvhodnější font a jeho velikost. (Čím menší font je zvolen, tím větší počet hlášení se vejde na jednu obrazovku.) Linux se vás může zeptat, jaký font má používat, nebo použije standardní „kompilovaný“ font (compiled font).[5]

   ```
   Console: 16 point font, 400 scans
   Console: colour VGA+ 80x25, 1 virtual console (max 63)
   ```

   V předcházejícím příkladu se vlastník počítače v době kompilace rozhodl pro větší, standardní font. Také si všimněte zastaralého tvaru slova „colour“. Linus se evidentně naučil špatnou verzi angličtiny.

2. V dalším kroku zobrazí jádro operačního systému zprávu o výkonnosti vašeho počítače. Výkonnost se měří v jednotkách „BogoMIPS“. Zkratka MIPS (million instructions per se-

5 Kompilace je proces, při kterém se instrukce srozumitelné člověku přeloží do instrukcí srozumitelných počítači. Pojmem kompilovaný se zde rozumí „začleněný do programu“.

cond) označuje počet instrukcí, které je počítač schopen provést za sekundu. „BogoMIPS“ je zkratkou „bogus MIPS“ (bogus=podvodný, falešný) a označuje, kolikrát za sekundu neudělá počítač absolutně nic. Protože cyklus, kterým se výkonnost počítače měří, nedělá ve skutečnosti nic, nelze uvedené číslo považovat za objektivní ukazatel výkonnosti.* Operační systém Linux toto číslo používá v případě, kdy potřebuje vyčkat na odezvu nějakého zařízení.

```
Calibrating delay loop.. ok - 33.28 BogoMIPS
```

3. V třetím kroku vypíše jádro systému informace o použití paměti:

```
Memory: 23180k/24576k available (544 kernel code, 384k reserved,468k data)
```

Z této informace vyplývá, že má počítač 24 MB operační paměti. Část této paměti byla rezervována pro jádro. Zbytek může být využíván ostatními programy. Uvedené informace se týkají paměti **RAM**, která může být používána k uchovávání dat, jen když je počítač zapnut. Počítač však má k dispozici permanentní paměť zvanou **pevný disk**. Obsah pevného disku zůstává uchován i tehdy, když je váš počítač vypnut.

4. V průběhu zavádění operačního systému provádí Linux řadu testů technického vybavení a o výsledcích těchto testů vypisuje průběžné zprávy.

```
This processor honours the WP bit even when in supervisor mode. Good.
```

5. V dalším kroku se Linux zabývá konfigurací sítě. Následující zpráva by měla být popsána v manuálu *„Příručka správce sítě“*.

```
Swansea University Computer Society NET3.033 for Linux 1.3.50
IP Protocols: ICMP, UDP, TCP
```

Popis konfigurace sítě přesahuje rámec této dokumentace, a proto se jím nebudeme zabývat.

6. Operační systém Linux podporuje jednotku pro výpočet v pohyblivé řádové čárce FPU (floating point unit). Jedná se o speciální integrovaný obvod (nebo přímo součást procesoru 80486DX), jenž realizuje aritmetické operace s neceločíselnými operandy. Některé z těchto integrovaných obvodů mohou být špatné a když se je operační systém Linux pokusí detekovat, dojde k jeho zhroucení – počítač přestane být funkční. Pokud nastane tento případ, uvidíte na obrazovce zprávu:

```
Checking 386/387 coupling...
```

V případě, že používáte procesor 486DX, uvidíte tuto zprávu:

```
Checking 386/387 coupling... Ok, fpu using exception 16 error reporting.
```

Máte-li starší procesor 386 s matematickým koprocesorem 387, uvidíte zprávu:

```
Checking 386/387 coupling... Ok, fpu using irq13 error reporting.
```

7. Nyní proběhne další test, jenž testuje instrukci „halt“.

* Poznámka korektora: Velikost tohoto čísla může být např. ovlivněna typem procesoru, jeho schopností předvídat cykly a také např. místem v paměti, kde se testovací cyklus provádí.

```
Checking 'hlt' instruction... Ok.
```

8. Po dokončení počáteční konfigurace vypíše operační systém Linux řádek, kterým identifikuje sám sebe. Na tomto řádku je informace o verzi systému, verzi překladače GNU C Compiler, kterým bylo jádro přeloženo, a kdy byl překlad realizován.

```
Linux version 1.3.55 (root@mousehouse) (gcc version 2.7.0)
Sun Jan 7 14:56:26 EST 1996
```

9. Jako další se nastartuje ovladač sériového portu, který zjistí informace o příslušném technickém vybavení. **Ovladač** (driver) je součást jádra operačního systému, která řídí nějaké zařízení, zpravidla periferní. Ovladač je odpovědný za detaily komunikace mezi procesorem a periferním zařízením. Ovladače umožňují, aby se programátoři mohli při psaní aplikací soustředit na vlastní aplikaci a nemuseli se starat o takové problémy, jako je například tisk na tiskárně.

```
Serial driver version 4.11 no serial optins enabled
tty00 at 0x03f8 (irq = 4) is a 16450
tty01 at 0x02f8 (irq = 3) is a 16450
tty02 at 0x03e8 (irq = 4) is a 16450
```

V tomto případě byly nalezeny tři sériové porty. Sériový port je ekvivalentní portu COM v operačním systému DOS. Jedná se o zařízení běžně používané ke komunikaci s modemy nebo myší.

Uvedený výpis sděluje, že sériový port s pořadovým číslem 0 (`COM1`) má adresu `0x03f8`. Když port přeruší činnost jádra (zpravidla proto, aby sdělil systému, že chce přenášet data), použije přerušení IRQ 4. Přerušení IRQ představuje další prostředek pro komunikaci mezi programovým vybavením a periferním zařízením. Každý sériový port má také svůj řídící integrovaný obvod. Často se používají obvody 16450, mohou se však také vyskytnou obvody 16550 nebo 820.**

10. Nyní přichází na řadu paralelní port. Paralelní port se nejčastěji připojuje k tiskárně a v operačním systému Linux začínají jména paralelních portů dvojicí písmen lp. lp je zkratkou pro Line Printer (řádková tiskárna), i když v poslední době by se mělo spíše jednat o zkratku pro Laser Printer (laserová tiskárna). Samozřejmě platí, že je operační systém Linux schopen komunikovat s každým typem paralelní tiskárny – jehličkovou, bublinkovou i laserovou.***

```
lp0 at 0x03bc, (polling)
```

Uvedená zpráva říká, že v systému byl nalezen jeden paralelní port a že pro něj bude použit standardní driver.

11. V dalším kroku identifikuje operační systém Linux ovladače pevných disků. Podle následující zprávy byly identifikovány dva pevné disky IDE:

```
hda: WDC AC2340, 325MB, w/127KB Cache, CHS=1010/12/55
hda: WDC AC2850F, 814MB, w/64KB Cache, LBA, CHS=827/32/63
```

---

* Poznámka korektora: V novějších jádrech Linuxu je obsažen i test na procesory, které obsahují i tzv. chybu „F00F".

** Poznámka korektora: V novějších počítačích je téměř vždy obvod 16550A.

*** Poznámka korektora: V poslední době se objevují i tzv. GDI-tiskárny, jejichž podpora není ještě zcela dokonalá, protože jejich výrobci nejsou ochotni poskytovat programátorům specifikaci komunikačního protokolu.

12. Dále provede jádro kontrolu disketových jednotek. Podle následujícího příkladu má počítač dvě disketové jednotky: jednotku „A“ pro diskety 5 1/4 palce a jednotku „B“ pro diskety 3 1/2 palce. Operační systém Linux přidělí disketové jednotce „A“ jméno `fd1` a disketové jednotce „B“ jméno `fd0`.

```
Floppy drive(s): fd0 is 1.44M, fd1 is 1.2M
floppy: FDC 0 is a National Semiconductor PC87306
```

13. Dalším ovladačem v případě mého počítače je ovladač SLIP. Vypíše následující zprávu o své konfiguraci:

```
SLIP: version 0.8.3-NET3.019-NEWTTY (dynamic channels, max=256)
  (6 bit encapsulation enabled)
CSLIP: code copyright 1989 Regents of the University of
  California
```

14. Dále jádro ověří všechny pevné disky, které byly identifikovány. Prohledá různé diskové oddíly a identifikuje ty části, na kterých se nachází operační systém, aby zabránil případné interferenci s jinými operačními systémy. V následujícím příkladě jsou identifikovány dva pevné disky: `hda` se čtyřmi oddíly a `hdb` s jedním oddílem:

```
Partition check:
hda: hda1 hda2 hda3 hda4
hdb: hdb1
```

15. V konečné fázi provede operační systém Linux připojení (mount) hlavního souborového systému platného pro kořenový oddíl (root partition). Kořenový oddíl je část pevného disku, ve které je umístěn operační systém. Když operační systém připojuje hlavní souborový systém, zpřístupní jej uživateli:

```
VFS: Mounted root (ext2 filesystem) readonly.
```

KAPITOLA 4

Příkazový interpret operačního systému Unix

Příkazy operačního systému Unix

Když se poprvé přihlásíte do operačního systému Unix, objeví se na obrazovce **výzva příkazového řádku** (prompt), která by mohla vypadat takto:

```
/home/larry#
```

Podle názvu „výzva příkazového řádku“ lze usuzovat, že systém na tomto místě očekává zadání příkazu. Každý příkaz operačního systému Unix představuje posloupnost písmen, čísel a znaků. Nepatří sem však mezery. Platnými příkazy operačního systému Unix jsou například: `mail`, `cat` nebo `CMU_is_Number-5`. Některé znaky není v příkazech povoleno používat – zmíníme se o nich později. Poznamenejme, že Unix má vlastnost označovanou jako „**case-sensitive**“ (doslova citlivý na malá a velká písmena).[1] To znamená, že `cat` a `Cat` jsou různé příkazy.

Příkazový řádek je zobrazován speciálním programem, který se nazývá **příkazový interpret** (shell). Příkazový interpret přijímá příkazy a realizuje je. Příkazové interprety mají svůj vlastní programovací jazyk a programy napsané v tomto jazyce se nazývají skripty příkazového interpretu (shell scripts).

Pro operační systémy Unix existují dva hlavní typy příkazových interpretů: Bourne shell a C shell. (V současnosti existuje více než desítka různých příkazových interpretů – např. v distribuci Red Hat Linux 5.1 je pro každého uživatele k dispozici pět příkazových interpretů – `ash`, `bash`, `tcsh`, `zsh` a `pdksh` – ze kterých si může vybrat příkazem `chsh`). Příkazový procesor Bourne shell byl nazván podle svého autora, kterým je Steven Bourne. Ten vytvořil původní příkazový procesor pro operační systém Unix a většina příkazových procesorů vytvořených od té doby končí písmeny sh. Tím se indikuje, že další příkazové procesory jsou rozšířením původního příkazového interpretu. Dnes existuje spousta implementací původního příkazového interpretu Bourne shell.

1 Některé operační systémy, jako je OS/2 nebo Windows NT, sice zachovávají v názvech velká a malá písmena, ale jinak mezi nimi nerozlišují. V praxi platí, že se v operačních systémech Unix nepoužívají příkazy nebo obecná jména, která by se lišila pouze malými a velkými písmeny (například různé příkazy cat a Cat).

Jiný typ představují tzv. C shelly (původně je navrhl pan Bill Joy), jež se rovněž hojně používají. Obecně platí, že příkazové interprety typu Bourne shell se používají z důvodu kompatibility s originálním příkazovým interpretem, zatímco C shell se používá pro interaktivní zpracování příkazů.

Příkazové interprety typu C shell se vyznačují tím, že mají lepší vlastnosti pro interaktivní práci, ale mají poměrně nepružné vlastnosti z hlediska programátorského.

Operační systém Linux se distribuuje s příkazovým interpretem `bash`, který byl vytvořen nadací Free Software Foundation. `bash` je zkratkou pro Bourne Again Shell, což tradičně představuje spíše špatnou slovní hříčku typickou pro operační systém Unix. Jedná se o vylepšený příkazový procesor Bourne shell. Má podobné programátorské vlastnosti jako Bourne shell a také má spoustu dobrých vlastností převzatých z příkazového interpretu C shell. `bash` je implicitním příkazovým interpretem používaným v operačním systému Linux.

Když se přihlásíte do systému Linux, pak je to právě program `bash`, který zobrazuje výzvu příkazového řádku. Příkazový procesor `bash` vás bude provázet po celou dobu práce s počítačem.

Typické příkazy operačního systému Unix

Prvním příkazem, se kterým se seznámíme, je příkaz `cat`. Máte-li jej použít, napište `cat` a stiskněte klávesu Enter.

```
/home/larry# cat
```

Jestliže je nyní kurzor na následujícím řádku, pak jste zadali příkaz `cat` správně. Existuje několik způsobů, jak příkaz `cat` zadat. Některé jsou správné, jiné nikoliv.

- Předpokládejme, že jste se spletli:

  ```
  /home/larry# ct
  ct: command not found
  /home/larry#
  ```

Tímto způsobem vás příkazový interpret upozornil na to, že nemůže najít program, který by se jmenoval „`ct`“. Nabídne vám další výzvu příkazového řádku a očekává další příkaz. Připomeňme, že operační systém Unix je citlivý na velká a malá písmena. Příkaz `CAT` je rovněž chybný.

- Před příkaz můžete uvést libovolný počet mezer, například:[2]

  ```
  /home/larry# _ cat
  ```

  Takto zadaný příkaz je v pořádku a program `cat` se spustí.

- V příkazovém řádku také můžete pouze stisknou klávesu Enter. Ničeho se neobávejte, nic se nestane.

Předpokládejme, že jste spustili program `cat`. Asi jste zvědavi, co dělá. Pokud si myslíte, že se jedná o nějakou hru, pak asi budete zklamáni. Tento program je velice užitečným nástrojem, i když se to na první pohled nezdá. Napište jakýkoliv text a stiskněte Enter. Pak na obrazovce uvidíte:

```
/home/larry# cat
Help! I'm stuck in a Linux program!
Help! I'm stuck in a Linux program!
```

2 _ indikuje mezeru.

(Skloněným písmem je označeno to, co jsem v programu `cat` napsal.) Zdá se, že program pouze kopíruje text. To je někdy užitečné, ale program `cat` toho umí mnohem více. Pro tento okamžik program cat ukončíme a později si uvedeme příklady, kdy je mnohem užitečnější.

K ukončení většiny příkazů operačního systému Unix se používá kombinace kláves Ctrl+D[3]. Touto kombinací odešlete znak konce souboru EOF (end-of-file). Alternativně lze znak EOF považovat za konec textu, avšak v této knize jej budeme považovat za konec souboru. Znak EOF signalizuje programům operačního systému Unix, že jste ukončili zadávání dat. Pro program `cat` to znamená, že další data nemá zpracovávat a ukončí se.

Také si můžete vyzkoušet program sort. Jak napovídá jeho název, jedná se o program pro třídění. Jestliže zapíšete několik řádků a pak stisknete Ctrl+D, zobrazí se vám tyto řádky uspořádány. Programy tohoto druhu se nazývají **filtry**. Fungují tím způsobem, že nejdříve akceptují nějaký text, provedou s ním nějaké operace (například třídění) a pak modifikovaný text vypíší na obrazovce (nebo na jiném výstupním zařízení). Programy cat a sort jsou poněkud neobvyklé filtry, cat je neobvyklý v tom, že přečte text a neprovádí v něm žádné změny. Program `sort` je neobvyklý v tom, že čte řádky a neprovádí nic, dokud nepřečte znak EOF. Mnoho filtrů zpracovává data řádek po řádku – přečtou řádek, provedou nějaké modifikace a zobrazí řádek modifikovaný.

Pomozte si sami

V operačních systémech Unix existuje příkaz `man`,[4] jenž zobrazuje tzv. manuálové stránky. Originální manuálové stránky jsou pouze v anglickém jazyce, ale na jejich překladu se již pracuje. Zadejte například příkaz:

```
/home/larry# man cat
CAT(1)                                                    CAT(1)
NAME
   cat - concentrate files and print on the standard output
SYNOPSIS
   cat [-benstuvAET] [--number] [--number-nonblank]
   [--squeeze-blank] [--show-nonprinting] [--show-ends]
   [--show-tabs] [--show-all] [--help] [--version] [--file...]
DESCRIPTION
   This manual page documents the GNU version of cat.       cat
```

Uvedený výpis představuje asi jednu stránku informací o příkazu `cat`. Nepředpokládejte, že budete okamžitě všemu rozumět. Manuálové stránky zpravidla předpokládají, že má jejich čtenář jisté znalosti o operačním systému Unix, tedy znalosti, které vy zatím nemáte. Když čtete manuálovou stránku, pak se v dolním řádku objeví blok s textem jako „`--more--`“ nebo „`Line 1`“. Jedná se o nápovědu, že k danému tématu existuje více informací.

Man po zobrazení první stránky vyčkává, co se rozhodnete dělat dále. Chcete-li pokračovat dále v prohlížení manuálových stránek, stiskněte klávesu Spacebar – pak se vám zobrazí následující stránka. Pokud chcete program man ukončit, stiskněte klávesu Q. Pak se vrátíte do příkazového řádku příkazového procesoru, který bude očekávat zadání dalšího příkazu.

---

3 Podržte stisknutou klávesu Ctrl a stiskněte klávesu D.

4 Program `man` také zobrazí informace o systémovém volání, podprogramu, formátu souboru a mnoho dalších informací. V originálních verzích operačního systému Unix obsahuje manuálová stránka stejné informace jako tištěná dokumentace. Pro tento okamžik se spokojíme s nápovědou k syntaxi příkazu.

Programu man lze také předat jisté argumenty, jež řídí jeho funkce. Předpokládejme například, že si chcete přečíst informace o všem, co souvisí s formátem Postscript (Postscript je řídící jazyk pro tisk na tiskárnách od firmy Adobe). Pak zadejte příkaz `man -k ps` nebo `man -k Postscript`. Zobrazí se vám seznam všech příkazů, systémových volání a další dokumentace pojednávající o tomto tématu. Tento postup je velmi užitečný v případě, kdy chcete nalézt nějaký nástroj, ale nevíte, kde jej hledat, nebo nevíte, zda vůbec existuje.

Ukládání informací

Programy patřící do kategorie filtrů jsou velmi užitečné, zejména pokud patříte mezi pokročilejší uživatele. Zůstává zde však jeden problém. Jak ukládat, uchovávat a aktualizovat informace? Za tímto účelem nabízí operační systém Unix **soubory** a **adresáře**.

Adresář je něco jako pořadač, který místo svazků papíru obsahuje soubory. Velké pořadače mohou obsahovat několik dalších menších pořadačů. V operačním systému Unix se systém adresářů a souborů nazývá souborový systém. Zpočátku je každý souborový systém tvořen jediným adresářem, který se nazývá kořenový nebo také hlavní adresář. Uvnitř tohoto adresáře se mohou vyskytovat další adresáře a uvnitř nich se opět mohou vyskytovat adresáře (ty se někdy nazývají podadresáře). V každém adresáři mohou být uloženy soubory.[5]

Každý soubor a každý adresář má své vlastní jméno. Jméno může být buď krátké a může se shodovat se jménem adresáře nebo souboru uloženého jinde, nebo dlouhé (tzv. jméno včetně cesty), které je v rámci souborového systému na daném disku unikátní. Příkladem krátkého jména může být `joe`, zatímco odpovídajícím dlouhým jménem (t.j. jménem včetně cesty) je `/home/larry/joe`. Posloupnost znaků v dlouhém jménu ukončená znakem / se nazývá cesta. Cesta může být dekódována do posloupnosti adresářů. Následující příklad ilustruje, jak systém dospěje k souboru `/home/larry/joe`:

```
/home/larry/joe
```

První znak / indikuje kořenový adresář.

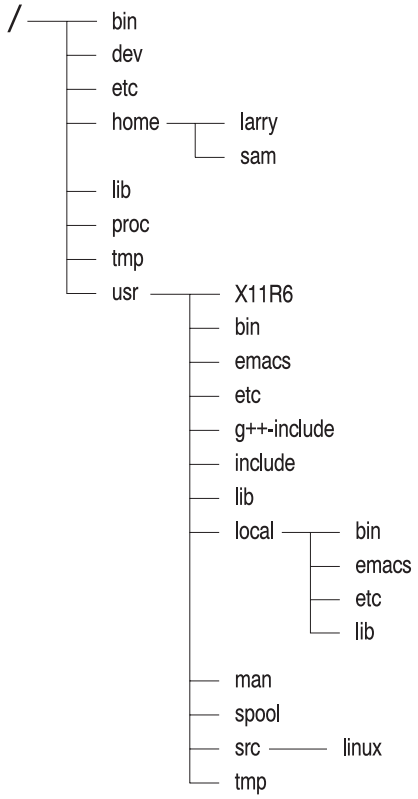
Následuje adresář se jménem `home`, který je podadresářem kořenového adresáře.

Další je adresář `larry`, který je podadresářem adresáře `home`.

Soubor `joe` je uvnitř adresáře `larry`. Cesta může odkazovat buď na adresář, nebo na soubor – `joe` tedy může být jak adresářem, tak i souborem. Všechny položky před krátkým jménem musejí být adresáře.

Existuje jednoduchý způsob, jak strukturu adresářů vizualizovat. Tato vizualizace se nazývá stromový diagram a většina uživatelů asi zná stromové diagramy z operačního systému DOS (vytvářejí je známé programy pro správu souborů, jako je NORTON COMMANDER). Stromový diagram můžete vytvořit programem tree nebo v programu Midnight Commander, linuxové obdobě Norton Commandera. Typickou stromovou strukturu adresářů používanou v operačním systému Linux znázorňuje obrázek 4.1. Poznamenejme, že zde uvedený diagram není kompletní. Úplná struktura operačního systému Linux obsahuje přes 8000 souborů! Zrovna tak platí, že zde mohou být uvedeny adresáře, jež se nevyskytují ve vaší instalaci a naopak, ve vaší instalaci mohou být adresáře, které v uvedeném obrázku uvedeny nejsou.

5 Počet podadresářů může být omezen, ale v některých systémech nemusí. V operačním systému Linux jsem bez problémů vytvořil podadresáře až do úrovně 10.



Obrázek 4.1 – Typická (zkrácená) stromová struktura adresářů v operačním systému Unix

Prohledávání adresářů pomocí příkazu ls

Nyní máte jistě představu o tom, jak jsou v operačním systému Unix organizovány adresáře a soubory. Samozřejmě budete potřebovat nějaké nástroje, pomocí nichž budete moci se soubory a adresáři manipulovat. Prvním důležitým nástrojem je příkaz `ls`. Jedná se o příkaz, který zobrazuje seznam souborů. Nyní zadejte příkaz `ls` na úrovni příkazového řádku:

```
/home/larry# ls
/home/larry#
```

Žádný seznam se nezobrazil, což je v pořádku. Operační systém Unix pracuje přesně. Příkazu `ls` nebyly předány žádné parametry, proto se žádný seznam nezobrazil.

Jak jsme se již zmínili, při instalaci operačního systému Linux se nainstaluje více než 8000 souborů. Kde jsou tyto soubory uloženy? Abychom pochopili používání příkazu `ls`, musíme si vysvětlit pojem aktuální adresář. Podle výzvy příkazového řádku poznáte, že aktuálním adresářem je `/home/larry`. Zde však nejsou uloženy žádné soubory, proto příkaz `ls` nezobrazil žádný seznam. Zkuste si zobrazit seznam souborů a adresářů uložených v hlavním adresáři:

```
/home/larry# ls /
bin     etc     lostfound       proc    tmp
boot    home    misc            root    usr
dev     lib     mnt             sbin    var
/home/larry#
```

V předcházejícím příkazu „ls /“ jsme programu ls předali parametr „/“. Prvním slovem v příkazu je jméno příkazu a další slova jsou parametry. Obecně platí, že parametry modifikují funkce příkazů. V případě příkazu ls se jako první parametr uvádí jméno adresáře, jehož obsah má příkaz ls zobrazit. Některé příkazy mají speciální parametry, kterým se říká volby (options) nebo přepínače (switches). Vyzkoušejte si následující příkaz:

```
/home/larry# ls -F /
bin/    etc/    lostfound/      proc/   tmp/
boot/   home/   misc/           root/   usr/
dev/    lib/    mnt/            sbin/   var/
/home/larry#
```

Parametr `-F` se nazývá **volba**. Volba je speciální parametr, který začíná pomlčkou a modifikuje způsob provádění programu. V případě příkazu `ls` je parametr `-F` určen k rozlišení položek seznamu na adresáře, speciální soubory, programy a ostatní soubory. Každá položka končící znakem / je adresář. Později budeme hovořit o příkazu ls, který je opravdu velmi univerzální, podrobněji.

Nyní byste si měli udělat malé cvičení. Především se naučte, jak příkaz ls pracuje. Vyzkoušejte si zobrazit obsahy jiných adresářů podle obrázku 4.1. Některé z nich budou prázdné, jiné budou obsahovat spoustu souborů. Doporučuji, abyste si vyzkoušeli příkaz ls s parametrem -F i bez něj. Například adresář /usr/local by mohl vypadat takto:

```
/home/larry# ls /usr/local
Acrobat3        com     etc     include lib     man     share   teTeX
bin             doc     games   info            libexec sbin    src
/home/larry#
```

Druhé cvičení bude více obecné. Spousta příkazů v operačním systému Unix se spouští podobně jako příkaz `ls`. Také mají nějaké parametry a nějaké volby, jež se zpravidla uvádějí jako jednoznakové řetězce za pomlčkou. Na rozdíl od příkazu `ls` však některé příkazy vyžadují parametry a/nebo volby. K vyjádření syntaxe příkazů budeme používat následující formu:

`ls [-aRF] [`*`adresář`*`]`

Když budeme popisovat nový příkaz, použijeme k jeho definici podobný syntaktický zápis. Prvním slovem je příkaz (v tomto případě `ls`). To, co následuje za příkazem, jsou parametry. Volitelné parametry jsou uzavřeny do hranatých závorek. Tzv. meta-proměnné jsou vyznačeny skloněným písmem – jsou to slova, která musejí být nahrazena skutečnými parametry. V uvedeném příkladu je meta-proměnnou *`adresář`*. Ta by měla být nahrazena jménem skutečného adresáře.

V případě voleb platí jiná pravidla. V syntaktické definici příkazu jsou uzavřeny v hranatých závorkách. V příkazu můžete použít kteroukoliv volbu nebo kteroukoliv jejich kombinaci. V případě příkazu `ls` máte tedy možnost použít osm kombinací voleb. Porovnejte výstup příkazů `ls -R` a `ls -F`.

Aktuální adresář a příkaz cd

`pwd`

Používání adresářů může být poněkud těžkopádné – asi by se vám nelíbilo, kdybyste museli pokaždé vypisovat celou cestu, kdykoliv byste si chtěli zpřístupnit nějaký soubor. V operačním systému Unix je zaveden pojem aktuálního adresáře. V příkladech, které jsme doposud použili, je aktuálním adresářem `/home/larry`. Pokud chcete zjistit, jaký je skutečný aktuální adresář, zadejte příkaz `pwd` (print working directory). V některých případech zobrazuje výzva příkazového řádku i jméno počítače. To je užitečné pouze tehdy, když je počítač zapojen do sítě, ve které se vyskytuje více počítačů.

```
mousehouse> pwd
/home/larry
mousehouse>
```

`cd [`*`adresář`*`]`

Jak jste se mohli přesvědčit, příkaz `pwd` zobrazí aktuální adresář.[6] Příkaz `pwd` je tedy velmi jednoduchý. Většina příkazů v operačním systému Unix funguje implicitně v aktuálním adresáři, který lze změnit pomocí příkazu `cd`. Vyzkoušejte si například příkazy:

```
/home/larry# cd /home
/home# ls -F
larry/ sam/ shutdown/ steve/ user1/
/home#
```

Pokud vynecháte volitelný parametr *`adresář`*, pak se navrátíte do vašeho domovského adresáře, kterým je v našem případě `/home/larry`. Jinak se dostanete do adresáře specifikovaného prostřednictvím parametru. Například:

```
/home# cd
/home/larry# cd /
/# cd home
/home# cd /usr
/usr# cd local/bin
/usr/local/bin#
```

Jak jste si asi všimli, příkaz `cd` umožňuje specifikovat buď **absolutní cestu,** nebo **relativní cestu**. Absolutní cesta začíná znakem / a musí obsahovat všechny adresáře vedoucí k výslednému adresáři, do kterého se chcete přepnout. Relativní cesta se vztahuje k vašemu aktuálnímu adresáři. V předcházejícím příkladu jsme v adresáři `/usr` zadali relativní adresář `local/bin` – `local` je adresář pod adresářem `usr` a `bin` je adresář pod adresářem `local`. Podobně jsme použili relativní adresář v příkazu `cd home`.

V operačním systému Unix (podobně jako v operačním systému DOS) existují dva speciální adresáře, jež se používají pouze jako relativní. Jsou to adresáře „`.`“ a „`..`“. Adresář „`.`“ specifikuje aktuální adresář a adresář „`..`“ specifikuje nadřazený adresář. Jedná se tedy o zkratky, které existují v každém adresáři, ale ne vždy mají přesně ten význam, jak jsme jej popsali. Například hlavní adresář má jako nadřazený adresář sebe sama.

6 Někdy se také používá termín pracovní adresář. V této knize se budeme držet termínu aktuální adresář.

Soubor `./chapter-1` v aktuálním adresáři by měl být identický se souborem `chapter–1`. Některé příkazy vyžadují zadání typu `./chapter-1`, ale to není častý případ. Ve většině případů jsou `./chapter-1` a `chapter-1` identické specifikace souboru.

Adresář „`..`“ je velmi užitečný při „pohybu“ v adresářové struktuře směrem nahoru:

```
/usr/local/bin# cd ..
/usr/local# ls -F
archives/      bin/    emacs@ etc/    ka9q/   lib/    tcl@
/usr/local# ls -F ../src
cweb/          linux/ xmris
/usr/local#
```

V tomto případě jsme se přepnuli do nadřazeného adresáře pomocí `cd ..` a pak jsme zobrazili obsah adresáře `/usr/src` z adresáře `/usr/local` pomocí příkazu `ls -F ../src`.

Pro domovský adresář existuje zkratka ~. Vyzkoušejte si následující příkaz:

```
/usr/local# ls -F ~/
/usr/local#
```

Opět jsme se přesvědčili, že v domovském adresáři nejsou žádné soubory. Užitečnost zkratky ~ pro domovský adresář vynikne zejména v případě, kdy začneme se soubory manipulovat.

Vytváření a odstraňování adresářů

`mkdir` *`adresář1`* [*`adresář2 ... adresářN`*]

Vytváření vlastních adresářů je v operačním systému Unix opravdu jednoduché a představuje velmi užitečný nástroj pro organizaci a údržbu souborů. K vytvoření adresáře slouží příkaz `mkdir` (make directory).

Uveďme si jednoduchý příklad, abychom pochopili, jak příkaz `mkdir` funguje.

```
/home/larry# ls -F
/home/larry# mkdir report-1993
/home/larry# ls -F
report-1993/
/home/larry# cd report-1993
/home/larry/report-1993#
```

Příkaz `mkdir` může akceptovat více než jeden parametr. Předpokládá, že každý z nich představuje jméno adresáře, který se má vytvořit. Opět můžete specifikovat celou cestu nebo relativní adresář. V předcházejícím příkladu jsme použili relativní adresář `report-1993`.

```
/home/larry/report-1993# mkdir /home/larry/report-1993/chap1
  ~/report-1993/chap2
/home/larry/report-1993# ls -F
chap1/  chap2/
/home/larry/report-1993#
```

`rmdir` *`adresář1`* [*`adresář2 ... adresářN`*]

Jakýmsi opakem k příkazu `mkdir` je příkaz `rmdir` (remove directory). `rmdir` funguje přesně jako `mkdir`. Podívejte se na následující příklad:

```
/home/larry/report-1993# rmdir chap1 chap3
rmdir: chap3: No such file or directory
/home/larry/report-1993# ls -F
chap2/
/home/larry/report-1993# cd ..
/home/larry# rmdir report-1993
rmdir: report-1993: Directory not empty
/home/larry#
```

Jak jste si zřejmě všimli, příkaz `rmdir` odmítnul odstranit adresář, který buď neexistuje nebo není prázdný. Uvědomte si, že adresář `report-1993` obsahuje podadresář `chap2`, proto nelze adresář `report-1993` odstranit.

Manipulace se soubory

Práce s adresáři je sice zajímavá, ale stále neřeší náš problém s ukládáním, aktualizací a udržováním informací. Tento problém se řeší prostřednictvím **souborů**.

Vytváření a editování souborů se budeme věnovat v několika následujících kapitolách.

Základními příkazy pro manipulaci se soubory jsou v operačním systému Unix příkazy `cp` (copy), `mv` (move) a `rm` (remove).

Příkaz pro kopírování souborů cp

`cp [-i]` *`zdroj cíl`*
`cp [-i]` *`soubor1 soubor2 ... souborN cílový adresář`*[7]

Příkaz `cp` je v operačním systému Unix velmi důležitým a výkonným příkazem. V jedné sekundě vám umožní přemístit tolik informací, kolik jich středověký mnich přemístil za celý rok.

Jestliže nemáte na svém pevném disku dostatek prostoru, buďte při používání příkazu `cp` opatrní. Nikdo nechce vidět zprávu „Disk full“, když pracuje s důležitými soubory. Příkaz `cp` je dále schopen přepsat důležité soubory bez jakéhokoliv varování – tomuto nebezpečí se budeme věnovat podrobněji.

Nejprve se zaměříme na první definici příkazu `cp`. Prvním parametrem příkazu `cp` je soubor, který se má kopírovat (tzv. zdrojový soubor, source file). Druhým parametrem je soubor, který má být kopií prvního (tzv. cílový soubor, destination file). Při kopírování můžete vytvářet soubory s novým jménem nebo můžete soubory se stejným jménem kopírovat do jiných adresářů. Vyzkoušejte si následující příklady:

```
/home/larry# ls -F /etc/passwd
/etc/passwd
/home/larry# cp /etc/passwd .
/home/larry# ls -F
passwd
/home/larry# cp passwd frog
/home/larry# ls -F
frog passwd
/home/larry#
```

7 Příkaz `cp` má v předloze dva řádky, neboť význam druhého parametru může být rozdílný v závislosti na počtu parametrů.

Prvním příkazem `cp` se z adresáře `/etc` zkopíroval soubor `passwd` (jenž obsahuje jména všech uživatelů systému Unix spolu se zakódovanými hesly) do mého domovského adresáře. Toto platí pouze za předpokladu, že nepoužíváme tzv. stínová hesla, kdy jsou zakódovaná hesla uložena v souboru `/etc/shadow`. Příkaz `cp` neruší zdrojový soubor, proto jsem neudělal nic, co by nějak poškodilo systém. Nyní existují v mém systému dvě kopie souboru `passwd`, jedna v adresáři `/etc` a druhá v mém domovském adresáři `/home/larry` a obě kopie mají jméno `passwd`.

Třetí kopii jsem vytvořil příkazem `cp passwd frog`. Nyní jsou v mém systému tři kopie souboru: `/etc/passwd`, `/home/larry/passwd` a `/home/larry/frog`. Posledně vytvořená kopie má jiné jméno, ale obsahuje stejná data.

Příkaz `cp` je schopen kopírovat soubory mezi adresáři, a to v tom případě, kdy je prvním parametrem jméno souboru a druhým parametrem je jméno adresáře. Pak zůstane jméno cílového souboru shodné se jménem zdrojového souboru.

Kopírovat soubory a přitom měnit jejich jména lze jen tehdy, když se v příkazu `cp` jako parametry uvedou jména souborů. S příkazem `cp` je spojeno jedno nebezpečí. Když například zadáte příkaz `cp /etc/passwd /etc/group`, vytvoří příkaz `cp` nový soubor `group`, jehož obsah bude identický s obsahem souboru `/etc/passwd`. Pokud by však soubor `/etc/group` již existoval, pak jej přepíšete bez jakéhokoliv varování a bez možnosti starou verzi souboru `/etc/group` obnovit.

Podívejme se na další příklad použití příkazu `cp`:

```
/home/larry#   ls -F
frog passwd
home/larry#    mkdir passwd_version
home/larry#    cp frog passwd passwd_version
home/larry#    ls -F
frog           passwd  passwd_version/
home/larry#    ls -F passwd_version
frog passwd
home/larry#
```

Jak jsem nyní použil příkaz `cp`? Je evidentní, že příkaz `cp` může akceptovat více než dva parametry (nyní jsem příkaz `cp` použil podle druhé definice). Příkaz `cp` v předcházejícím příkladu zkopíroval všechny uvedené soubory (`frog` a `passwd`) do adresáře `passwd_version`, který je uveden jako poslední parametr. V podstatě platí, že příkaz `cp` je schopen akceptovat jakýkoliv počet parametrů, přičemž prvních n-1 považuje za jména souborů a poslední považuje za jméno adresáře, do kterého se mají uvedené soubory kopírovat.

Jestliže kopírujete v daném okamžiku více než jeden soubor, pak jej nemůžete přejmenovat – kopírované soubory si uchovají svá původní jména. Co se stane, když zadáte příkaz `cp frog passwd toad`, kde `frog` a `passwd` jsou soubory a přitom `toad` není adresář? Vyzkoušejte si takový příkaz a uvidíte.

Odstranění souborů pomocí příkazu rm

`rm [-i] soubor1 soubor2 ... souborN`

Nyní, když jsme se naučili, jak kopírovat a vytvářet soubory (později si probereme jiné způsoby vytváření souborů), bude jistě užitečné vědět, jak soubory rušit. Ve skutečnosti je to velmi jednoduché. Příkaz pro odstraňování souborů má název `rm` a zruší všechny soubory, jejichž jména se uvedou jako parametry.

Například:

```
/home/larry# ls -F
frog passwd passwd_version/
/home/larry# rm frog toad passwd
rm: toad: No such file or directory
/home/larry# ls -F
passwd_version/
/home/larry#
```

Jak asi vidíte, příkaz `rm` je velmi nevlídný. Nejen, že se vás nezeptá, zda má uvedené soubory skutečně zrušit, ale provede zrušení existujících souborů, i když vlastně příkaz nebyl správně zadán (soubor `toad` neexistoval). Příkaz `rm` může být skutečně nebezpečný. Podívejme se na rozdíl mezi následujícími dvěma posloupnostmi příkazů:

```
/home/larry# ls -F
toad frog/
/home/larry# ls -F frog
toad
/home/larry# rm frog/toad
/home/larry#
```

```
/home/larry# rm frog toad
rm: frog is a directory
/home/larry# ls -F
frog/
/home/larry#
```

Pátý řádek v první posloupnosti a první řádek v druhé se liší jediným znakem. Uvedený příklad má ilustrovat, že opomenutí jediného znaku může v operačním systému Unix způsobit, že provedete něco úplně jiného, než jste původně chtěli. Proto je velmi důležité, abyste raději několikrát zkontrolovali zapsaný příkaz, než stisknete klávesu Enter.

Přesouvání souborů pomocí příkazu mv

`mv [-i]` *`starý nový`*
`mv [-i]` *`soubor1 soubor2 ... souborN adresář`*

Nakonec se podívejme na příkaz pro tzv. přesouvání souborů. Příkaz má název `mv` a je podobný příkazu `cp`. Na rozdíl od příkazu `cp` však zdrojové soubory po jejich zkopírování odstraňuje. Příkaz `mv` je tedy jakousi kombinací příkazů `cp` a `rm`. Uveďme si na následujícím příkladu, jak příkaz `mv` funguje:

```
/home/larry# cp /etc/passwd .
/home/larry# ls -F
passwd
/home/larry# mv passwd frog
/home/larry# ls -F
frog
/home/larry# mkdir report
/home/larry# mv frog report
/home/larry# ls -F
```

```
report/
/home/larry# ls -F report
frog
/home/larry#
```

Jak jste si pravděpodobně všimli, příkaz `mv` soubor přejmenuje, pokud je druhým parametrem jméno souboru. Jestliže je druhým parametrem jméno adresáře, pak se soubor přesune do nového adresáře a přitom zůstane uchováno jeho původní jméno.

Podobně jako v případě příkazu `cp` musíte být při používání příkazu `mv` velmi opatrní. Příkaz neprovádí kontrolu, zda cílový soubor existuje nebo ne. Kdyby například v mém adresáři `report` již existoval soubor `frog`, pak by se příkazem `mv frog report` nejdříve zrušil soubor `/report/frog` a pak by se nahradil obsahem souboru `~/frog`.

Ve skutečnosti existuje možnost, jak přimět příkazy `rm`, `cp` a `mv`, aby vás upozorňovali na možnost přepsání či zrušení souborů. Všechny tyto příkazy akceptují volbu `-i`. Po uvedení této volby bude uživatel před každým zrušením souboru upozorněn. Dokonce můžete použít tzv. **alias** (druhé jméno) a zařídit, aby vás například příkaz `rm` upozorňoval na zrušení souboru, aniž uvedete volbu `-i`. O tom však budeme diskutovat až v oddíle Vytváření aliasů v kapitole 9.

KAPITOLA 5

Systém X Window

Tato kapitola se týká pouze uživatelů systému X Window. Jestliže máte před sebou barevnou grafickou obrazovku s mnoha okny a kurzorem, kterým lze pohybovat pouze prostřednictvím myši, pak jste zřejmě uživateli systému X Window. Pokud však máte před sebou obrazovku s bílým textem na černém pozadí, pak momentálně systém X Window nemáte aktivován. Budete-li chtít systém X Window aktivovat, přečtěte si následující oddíl.

Spuštění a ukončení systému X Window

Spuštění systému X Window

Systém X Window můžete spustit i v případě, kdy se automaticky nenastartuje ihned po zavedení operačního systému. Existují dva příkazy, kterými lze nastartovat systém X Window: `startx` a `xinit`. Nejdříve vyzkoušejte příkaz `startx`. Pokud příkazový procesor ohlásí, že takový příkaz neexistuje, pak zkuste `xinit`. Pokud ani v tomto případě nedojde ke spuštění systému X Window, pak jej pravděpodobně nemáte nainstalován. Prostudujte si příslušnou dokumentaci.

Jestliže se příkaz spustí, ale po nějakém čase se opět vrátíte do příkazového řádku příkazového procesoru, pak je systém X Window sice instalován, ale není nakonfigurován. I v tomto případě si budete muset důkladně přečíst dokumentaci k instalaci systému X Window.

Ukončení systému X Window

V závislosti na tom, jak je váš systém X Window konfigurován, existují dva možné způsoby, jak systém X Window ukončit. První závisí na tom, zda váš systém X Window řídí správce oken či nikoliv. Pokud ano, můžete systém X Window ukončit prostřednictvím nabídky (viz oddíl Nabídky). Chcete-li si zobrazit nabídku, klepněte myší na pozadí pracovní plochy.

Pak se vám mezi jinými objeví položka „Exit Window Manager“ nebo „Exit X“ nebo jiná položka obsahující slovo „Exit“. Pokuste se najít tuto položku (počítejte s tím, že zde může existovat více než jedna nabídka – vyzkoušejte různá tlačítka myši) a ukončete systém X Window.

Druhá metoda ukončení systému X Window spočívá v tom, že se k řízení systému X Window využije speciálního okna `xterm`. V takovém případě byste měli najít okno nazvané „login“ nebo „system `xterm`“. Přesuňte kurzor myši do tohoto okna a zadejte příkaz `exit`.

Jestliže se systém X Window automaticky spustil, když jste se přihlásili do systému, pak by vás jedna z výše uvedených metod ukončení systému X Window měla automaticky odhlásit. Jestliže jste však systém X Window nastartovali z příkazového řádku, pak se do tohoto příkazového řádku po ukončení systému X Window opět vrátíte (pokud se chcete odhlásit, zadejte příkaz `logout`).

Co je systém X Window?

Systém X Window je distribuované grafické uživatelské prostředí původně vyvinuté v akademické instituci Massachusetts Institute of Technology. Později byl předán skupině distributorů s názvem „The X Consortium“. Zde se dodnes udržuje a nadále vyvíjí.

Systém X Window (někdy zkracován jako „X“[1]) se každých několik málo let distribuuje v nové verzi, uváděné jako „release“. Poslední verze má označení X11R6, tedy „release“ 6. Číslo 11 značí hlavní verzi systému X Window, avšak toto označení je trvalé, protože se nová verze neplánuje.

V souvislosti se systémem X Window existují dva důležité pojmy, se kterými byste se měli seznámit. Program, jenž běží pod systémem X Window, se nazývá **klient**. Například `xterm` je klient, který se spustí v okamžiku, kdy se přihlašujete do systému. Program, který poskytuje služby ostatním programům typu klient, se nazývá **server**. Server například kreslí okno pro program `xterm` a zajišťuje komunikaci s uživatelem.

Protože jsou server a klient dva odlišné programy, je možné zajistit, aby server běžel na jednom počítači a klient na jiném. Vzhledem k tomu, že grafické uživatelské rozhraní dodržuje jistý standard, můžete v systému X Window spustit program na vzdáleném počítači (třeba na druhém konci světa) a přitom můžete grafické rozhraní tohoto programu vidět na svém počítači.

Dalším důležitým termínem, se kterým byste se měli seznámit, je **správce oken** (window manager). Jedná se o speciální program typu klient, který určuje pozici jednotlivých oken na pracovní ploše systému X Window a řídí způsoby, kterými uživatel například přemisťuje okna. Samotný server v podstatě pro uživatele nic nedělá. Pouze představuje rozhraní mezi uživatelem a klientem.

Co se nachází na pracovní ploše X Window

Když poprvé nastartujete systém X Window, zároveň se nastartuje řada dalších programů. Jako první se nastartuje server. Další v pořadí se nastartují některé programy typu klient – jejich pořadí a typ závisí na distribuci a není standardizován. Je však pravděpodobné, že mezi těmito klienty je správce oken (buď program `fvwm` nebo `twm`), terminálové okno `xterm` a hodiny `xclock`.

Program XClock

`xclock [-digital] [-analog] [-update` *seconds*`] [-hands` *color*`]`

Program `xclock` (hodiny) funguje přesně tak, jak asi očekáváte. Zobrazuje ciferník s vteřinovou ručičkou a malou i velkou ručičkou v malém okně.

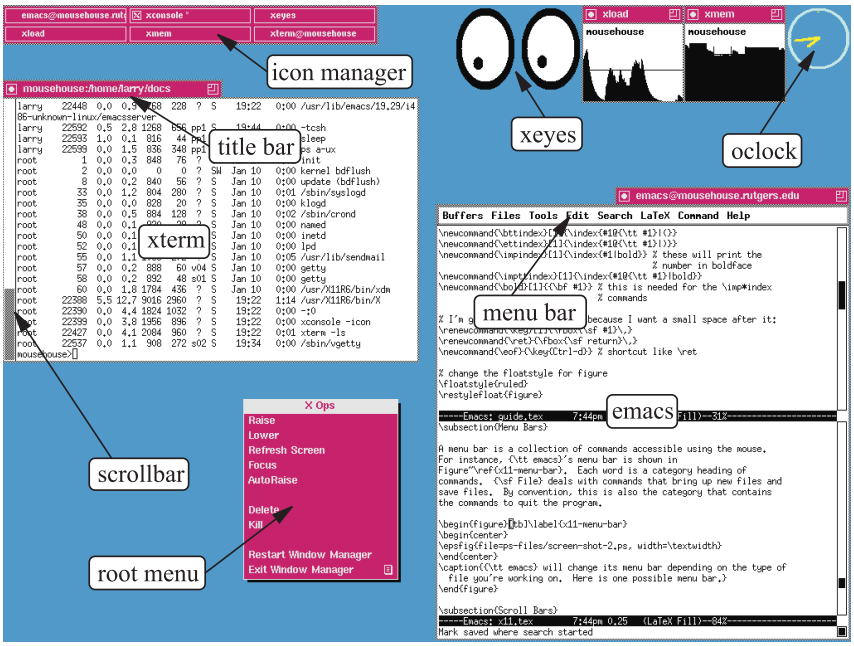
Prostřednictvím myši nemůžete vlastnosti okna s hodinami příliš modifikovat. Nanejvýš můžete měnit jeho velikost. Pokud však program xclock spustíte z příkazového řádku, pak můžete prostřednictvím jeho parametrů nastavit různé vlastnosti. Když zadáte příkaz `xclock -digital`, zobrazí se digitální hodiny. Pomocí parametru `-update` můžete nastavit, zda se má vteřinová ručička aktualizovat každou sekundu (`-update 1`) nebo například každých pět sekund (`update -5`).

Chcete-li vědět více o programu `xclock` a jeho parametrech, podívejte se do manuálových stránek. Stačí v příkazovém řádku zadat `man xclock`. Budete-li chtít spustit více programů `xclock` najednou, přečtěte si oddíl Současný běh úloh, ve kterém budeme diskutovat o provozování více úloh současně.

Jestliže máte program `xclock` spuštěn na popředí (což je běžný způsob v provozování programů) a chcete jej ukončit, stiskněte kombinaci kláves Ctrl-C.

---

1 Existuje několik přijatelných způsobů, jak nazývat systém X Window. Nejčastěji se vyskytuje zkratka „X“ nebo název „X Window“.



Obrázek 5.1 – Příklad standardní obrazovky systému X Window. V tomto příkladu uživatel spustil program twm. Standardní hodiny byly nahrazeny jiným programem zvaným `oclock`.

Program xterm

Okno s příkazovým řádkem uvnitř (tedy v našem případě okno s výzvou `/home/larry#`) je řízeno programem, jenž se nazývá xterm. Což je zvláštní a přitom komplikovaný program. Na první pohled se zdá, že toho moc neumí, ale jeho prostřednictvím lze vykonat spoustu užitečné práce. Emuluje terminál, a proto v něm lze spouštět všechny textově orientované aplikace operačního systému Unix. Také obsahuje vyrovnávací paměť uchovávající jistou množinu příkazů – proto se můžete snadno vracet k dříve zadaným příkazům (podrobněji se tímto tématem budeme zabývat v oddílu Posuvné lišty).

Celá tato kniha je zaměřena na příkazy zadávané v příkazovém řádku terminálu, proto je program xterm v systému X Window tak důležitý. Chcete-li něco zadávat v okně tohoto programu xterm, musí být toto okno aktivní. To znamená, že musíte do okna terminálu přesunout kurzor myši. Způsob, kterým se nastavuje aktivní okno obecně, závisí na správci oken.

Program xterm představuje jednu z možností, jak v systému X Window spouštět více programů současně. Programy běžící v systému X Window jsou standardními programy operačního systému Unix, proto mohou být spuštěny z terminálového okna. Protože dlouhodobé programy spouštěné z terminálového okna zablokují program `xterm` po celou dobu svého běhu, spouští se takové programy zpravidla na pozadí. Více informací o tomto tématu uvedeme v oddílu Současný běh úloh.

Správce oken

V operačním systému Linux existují dva běžně používané programy pro správu oken.* První z nich, twm, je zkratkou pro „Tab Window Manager“. Druhý program je menší a má název fvwm („F(?) Virtual Window Manager“). Oba programy jsou konfigurovatelné, což znamená, že si uživatel může definovat význam ovládacích klíčů a způsob práce s myší.

Konfiguraci programu twm je věnován oddíl Konfigurace programu twm a konfiguraci programu fvwm probereme v oddílu Konfigurace programu fvwm.

Jak se vytvářejí nová okna

Při vytvoření nového okna existují tři možné varianty, jež realizuje správce oken. V poslední době se objevuje spousta nových správců oken, např. `fvwm95`, `gnome` nebo `kwm`. Správce oken může být nakonfigurován tak, že se zobrazí rám nového okna a uživatel má možnost okno umístit kamkoliv na obrazovku. Tento způsob umístění okna se nazývá **ruční umístění** (manual placement). Jestliže se před vámi objeví rám okna, jednoduše jej pomocí kurzoru myši nastavte na požadovanou pozici a stiskněte levé tlačítko.

Je také možné, že správce oken umístí nové okno na pracovní ploše systému X Window dle „svého uvážení“. Takové umístění okna se nazývá **náhodné umístění** (random placement).

Třetí varianta spočívá v tom, že se okno náležící jisté aplikaci pokaždé otevře na stejné pozici. Správce oken může být nakonfigurován tak, aby pro vybrané aplikace otevíral okna s předem definovanou velikostí na předem definované pozici.

Jako příklad může sloužit program `xclock`, kdy asi budete chtít zobrazovat hodiny v horním levém rohu pracovní plochy systému X Window.

Aktivní okno

Správce oken řídí spoustu důležitých nastavení. Vás bude určitě zajímat, jak nastavit dané okno (například terminálové okno) tak, abyste v něm mohli zadávat příkazy. Aktivní okno je nejčastěji určeno pozicí kurzoru myši. Jestliže je kurzor myši umístěn na jednom z terminálových oken,[2] pak v tomto okně můžete zadávat příkazy z klávesnice. V tom spočívá rozdíl od jiných operačních systémů, jako je OS/2, Microsoft Windows nebo Macintosh, kde musíte na dané okno (chcete-li, aby bylo aktivní) klepnout myší. V systému X Window obvykle platí, že když kurzor myši opustí rámec okna, pak v tomto okně již nemůžete zadávat příkazy.

Poznamenejme, že oba správce oken (tedy program twm a fvwm) lze konfigurovat tak, že se aktivace oken bude realizovat stejně jako například v operačním systému OS/2. Popis konfigurace správce oken najdete v příslušné dokumentaci, nebo můžete požadované konfigurace dosáhnout metodou pokusů a omylů.

Přemisťování oken

V systému X Window lze rovněž konfigurovat způsob, kterým mají být okna přemisťována. Moje osobní konfigurace programu twm obsahuje tři způsoby, jak posouvat okna po pracovní ploše. Nejznámější způsob spočívá v tom, že se kurzor myši umístí na titulní (hlavní) lištu okna, stiskne kterékoliv tlačítko (levé, pravé nebo střední[3]), a pak se okno přesune na novou pozici. Je ovšem velmi pravděpodobné, že je váš systém X Window konfigurován tak, aby se okna přemisťovala pomocí levého tlačítka (tak, jak je tomu u ostatních operačních systémů).

---

2 V daném okamžiku můžete spustit několik programů `xterm` současně.

3 Některé osobní počítače mají dvoutlačítkovou myš. Pak můžete prostřední tlačítko simulovat současným stisknutím pravého a levého tlačítka.

* Poz. korektora: V současné době je jich již mnohem více.

Jiný způsob přemisťování oken může spočívat v tom, že se využívá některé klávesy na klávesnici. Například moje vlastní konfigurace umožňuje stisknout klávesu Alt, nastavit kurzor myši kamkoliv do rámce okna a pak po stisknutí levého tlačítka okno přesunout na novou pozici.

I zde platí, že si konfiguraci správce oken můžete měnit metodou pokusů a omylů, nebo si prostudujte příslušnou dokumentaci. Budete-li se pokoušet interpretovat konfigurační soubor správce oken, přečtěte si oddíl Konfigurace programu twm nebo oddíl Konfigurace programu fvwm.

Překrývání oken

Protože v systému X Window může být otevřeno několik oken najednou, má smysl hovořit o překrývání oken. Přestože okna a samotná pracovní plocha systému X Window jsou dvourozměrné, mohou se okna navzájem překrývat, jako by představovaly třírozměrné útvary. To znamená, že vybrané okno může být zobrazeno v popředí a částečně nebo úplně překrývat okna na pozadí. V souvislosti s překrýváním oken existuje několik operací:

- **Vyzvednutí** (raising) okna do popředí. Toho lze zpravidla dosáhnout klepnutím jistým tlačítkem myši na titulní lištu vybraného okna. Podle konfigurace správce oken to může být kterékoliv tlačítko; také je možné dosáhnout vyzvednutí okna do popředí pomocí více než jednoho tlačítka.
- **Zasunutí** (lowering) okna do pozadí. Toho lze obvykle dosáhnout klepnutím jiného tlačítka myši na titulní lištu okna. Dokonce lze nakonfigurovat správce oken tak, že k vyzvednutí i zasunutí okna se použije totéž tlačítko myši – je-li okno v popředí, pak se zasune do pozadí a je-li v pozadí, vyzvedne se do popředí.
- **Cyklické** procházení všech oken. Správce oken může být konfigurován tak, že po stisknutí jisté klávesy se postupně objevuje ve stanoveném pořadí každé otevřené okno v popředí.

Transformace okna do ikony

Nyní se podívejme na další operace, které umožňují transformovat (nebo také skrýt) okno do ikony. V závislosti na konfiguraci správce oken lze tohoto efektu dosáhnout několika různými způsoby. Při používání správce oken twm si většina lidí nakonfiguruje tzv. **správce ikon** (icon manager). Jedná se o speciální okno, které obsahuje seznam jmen všech ostatních oken otevřených na pracovní ploše. Pokud klepnete jistým tlačítkem myši na konkrétní jméno, dané okno se transformuje na ikonu. Okno je stále aktivní, ale nevidíte jej. Pomocí jiného tlačítka můžete provést inverzní operaci – ikona se zpět transformuje na okno.

Uvedené vlastnosti mohou být velmi užitečné. Předpokládejme například, že máte otevřenu spoustu terminálových oken pro příležitostnou komunikaci se vzdálenými počítači. Kdyby zůstala všechna okna otevřena, měli byste na pracovní ploše nepřehledno. Protože však s těmito terminálovými okny pracujete pouze příležitostně, můžete je transformovat do ikon a získat tak větší přehled. Jediné malé nebezpečí spočívá v tom, že zapomenete na některé okno transformované do ikony a zbytečně si otevřete pro stejný účel nové okno.

Pokud jde o umisťování ikon, lze správce oken nakonfigurovat tak, aby se ikony automaticky řadily na spodní okraj pracovní plochy.

Jak měnit velikost oken

V systému X Window existuje několik způsobů, jak měnit velikost oken. Tyto způsoby opět závisejí na konfiguraci správce oken. První a asi nejpoužívanější metoda spočívá v tom, že klepnete na rámeček ohraničující okno levým tlačítkem myši, držíte je stisknuté a rozměr okna nastavíte na požadovanou velikost (tento způsob je běžný i u ostatních operačních systémů, jako je Microsoft Windows nebo OS/2).

Další metoda je založena na speciálním tlačítku umístěném v titulní liště okna. Na obrázku 5.1 si můžete všimnout malého tlačítka na pravé straně titulní lišty každého okna. Stačí klepnout levým tlačítkem myši na toto tlačítko a nastavit rozměr okna na požadovanou velikost. Jakmile má okno příslušné rozměry, levé tlačítko myši uvolněte.

Nastavení maximální velikosti okna

Většina správců oken podporuje tzv. maximalizaci oken, tedy nastavení rozměrů okna tak, aby pokrylo celou obrazovku. Při použití správce oken `twm` lze dokonce nastavit takovou konfiguraci, aby bylo možné maximalizovat pouze vertikální rozměr, horizontální rozměr nebo oba rozměry najednou. Tento proces je v programu `twm` označován jako „zooming" (zvětšování), ale častěji se dává přednost termínu „maximization". Různé aplikace reagují na změnu rozměrů okna různě. Například `xterm` vám poskytne větší pracovní prostor a nezvětšuje přitom velikost fontů.

Obecně lze říci, že proces maximalizace oken není bohužel v systému X Window standardizován.

Nabídky

Dalším úkolem správce oken je řídit funkci nabídek a umožnit tak uživateli rychle realizovat úkony, které se stále dokola opakují. Například lze vytvořit nabídku, která umožní automaticky a rychle spustit editor emacs nebo terminál `xterm`. Obecně platí, že programy běžící v systému X Window můžete spouštět buď z terminálového okna, což je pomalejší a nepohodlné, nebo z vhodně nakonfigurovaných nabídek, což je rychlé a pohodlné.

Různé nabídky mohou být zpřístupněny klepnutím myši v základním okně (často se pro toto okno používá označení pracovní plocha), tedy okně, které nelze přemisťovat a které obsahuje všechna ostatní okna. Pozadí základního okna je implicitně šedé[4]. Chcete-li vyvolat nabídku, stačí umístit kurzor myši na pozadí základního okna a stisknout tlačítko. V nabídce si opět pomocí kurzoru myši vyberte požadovanou položku (aniž uvolníte tlačítko myši) a uvolněním tlačítka ji spusťte.

Atributy systému X Window

Existuje spousta programů, jež využívají vlastností systému X Window. Některé programy, jako je editor emacs, mohou být spuštěny jako textově orientované aplikace a zároveň mohou být spuštěny v prostředí systému X Window. Existuje však řada programů, které mohou běžet pouze v prostředí X Window.

Geometrie

Existuje několik aspektů, které mají programy běžící pod systémem X Window společné. První z nich lze označit jako geometrii. Atributy geometrie popisují velikost a umístění okna a tvoří je čtyři komponenty.

- Horizontální rozměr, zpravidla měřený v grafických bodech. (Grafický bod je nejmenší jednotka na obrazovce, které lze přiřadit barvu. Systém X Window běžně pracuje s rozlišovací schopností 1024 bodů horizontálně a 768 bodů vertikálně.) Některé aplikace (`xterm` nebo `emacs`) měří velikost okna ve znacích. Například šířka okna je dána počtem znaků na řádku (nejčastěji osmdesát), který může aplikace zobrazit.

4 Existuje program xfishtank, který na pozadí základního okna simuluje akvárium a také spousta dalších programů pro změnu pozadí v systému X Window.

- Vertikální rozměr, opět obvykle měřený v grafických bodech. Rovněž je možné stanovit vertikální rozměr ve znacích.
- Horizontální vzdálenost od okrajů obrazovky. Například číslo `+35` znamená, že levý okraj okna bude vzdálen 35 grafických bodů od levého okraje obrazovky. Na druhé straně číslo `-50` bude znamenat, že pravý okraj okna bude vzdálen padesát grafických bodů od pravého okraje obrazovky. Obecně platí, že není možné inicializovat okno mimo rámec obrazovky, i když je pak možné okno mimo rámec obrazovky přesunout.
- Vertikální vzdálenost od horního nebo dolního okraje obrazovky. Kladná vertikální vzdálenost je měřena od horního okraje obrazovky a záporná je měřena od spodního okraje obrazovky. Všechny čtyři komponenty definující geometrii okna se uvádějí prostřednictvím jediného řetězce. Například `503x73-78+0` znamená, že okno bude mít šířku 503 grafických bodů, výšku 73 grafických bodů a že bude umístěno vpravo bezprostředně pod horním okrajem obrazovky. Obecný tvar řetězce pro specifikaci geometrie okna je tedy: `hsizexvsize+hplace+vplace`.

Display

Každé aplikaci v systému X Window je přiřazen tzv. display, což je jméno obrazovky, kterou řídí server systému X Window. Displej se skládá ze tří komponent:

- Jméno počítače, na kterém server běží. Při samostatné instalaci operačního systému Linux běží server vždy na stejném systému jako klient. Ve většině případů může být jméno počítače vynecháno.
- Číslo serveru, který na daném počítači běží. Na každém jednotlivém počítači může běžet několik serverů systému X Window současně (v případě operačního systému Unix to není příliš častý případ). Pak každý z nich musí mít unikátní číslo.
- Číslo obrazovky. Systém X Window podporuje servery řídící v daném okamžiku více než jednu obrazovku. Některé aplikace je účelné provozovat na více než jednom monitoru a pak je nutné, aby server řídil více monitorů. Není totiž efektivní, aby byl pro každý monitor spuštěn zvláštní server.

Uvedené tři atributy lze specifikovat prostřednictvím jediného řetězce v následujícím formátu: *`machine:server-number.screen-number`*.

Například moje aplikace mají nastavenu hodnotu pro display jako `:0.0`. To znamená, že systém pracuje s první obrazovkou, řídí jej první server a běží na lokálním počítači. Kdybych však používal vzdálený počítač, pak bude display nastaven jako `mousehouse:0.0`.

Implicitně se řetězec definující hodnotu display čte ze systémové proměnné `DISPLAY` (viz oddíl Systémové proměnné) a může být předefinován parametry příkazového řádku pro spuštění systému X Window (viz tabulka 5.2). Chcete-li se přesvědčit, zda je systémová proměnná nastavena, zadejte příkaz `echo $DISPLAY`.

Společné vlastnosti

Systém X Window představuje uživatelské grafické rozhraní, které lze prakticky ve všech aspektech konfigurovat. Je téměř nemožné říci, jak jeho jednotlivé komponenty fungují, protože to především závisí na konfiguraci.

jméno	následováno čím	příklad
`-geometry`	geometrií okna	`xterm -geometry 60x24+0+90`
`-display`	display, ve kterém se má program objevit	`xterm -display lionsden:0.0`
`-fg`	primární barva popředí	`xterm -fg yellow`
`-bg`	primární barva pozadí	`xterm -bg blue`

Tabulka 5.2 – Standardní volby pro programy běžící pod systémem X Window

Každá vlastnost může být překonfigurována, změněna nebo úplně nahrazena jinou. Proto je obtížné jednoznačně popsat chování jednotlivých prvků tvořících toto rozhraní. Zatím jsme popsali to, co popsat lze: různé správce oken a možnosti jejich konfigurace.

Situaci ještě více komplikuje skutečnost, že různé aplikace jsou založeny na různých grafických knihovnách. Pro tyto knihovny se vžil název „knihovny přípravků" – widget sets. Spolu se systémem X Window se distribuuje „Athena", grafický systém vyvinutý v Massachusetts Institute of Technology. Systém Athena se běžně používá ve volně šiřitelném programovém vybavení. Má tu nevýhodu, že nevytváří příliš dobře vyhlížející grafické objekty a jeho používání je poměrně těžkopádné.

Další populární knihovnou je „Motif". Motif patří ke komerčním produktům a má podobné vlastnosti jako uživatelské grafické rozhraní používané v operačním systému Microsoft Windows. Tuto knihovnu využívá spousta komerčních a některé volně šiřitelné aplikace. Populární program pro prohlížení stránek WWW Netscape také využívá knihovnu Motif.

Pokusme se nyní popsat některé společné vlastnosti systému X Window.

Tlačítka

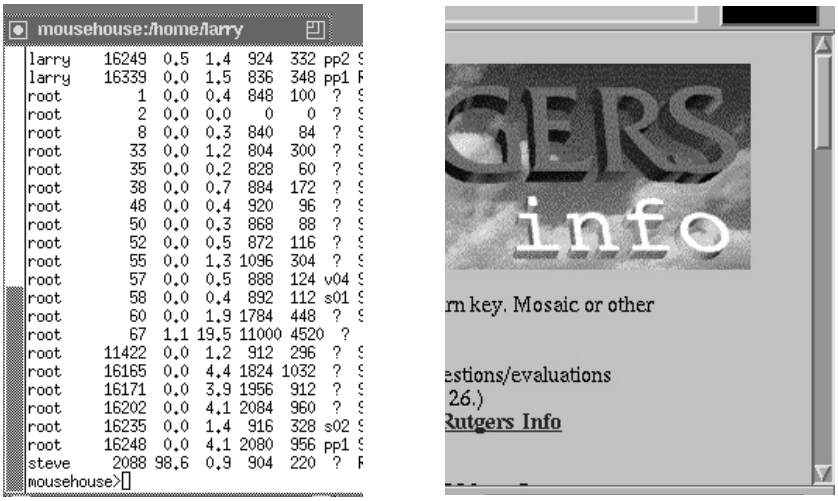
Tlačítka jsou prvky grafického rozhraní, jež se nejsnadněji používají. Stačí na tlačítko přesunout kurzor myši a klepnout levým tlačítkem. Tlačítka systémů Athena a Motif fungují stejně, rozdíly mezi nimi jsou pouze kosmetické.

Nabídkové lišty

Nabídková lišta je soubor příkazů zpřístupněných pomocí myši. Na obrázku 5.3 je například zobrazena nabídka editoru `emacs`. Každé slovo v nabídce představuje záhlaví jisté množiny dalších příkazů. Například položka `File` obsahuje příkazy pro zavádění a ukládání souborů. Podle zažité konvence zde najdete i příkaz k ukončení editoru.

Chcete-li zpřístupnit některý příkaz z nabídky, přesuňte kurzor myši na příslušnou nabídku (například `File`) a stiskněte levé tlačítko myši (držte je stále stisknuté). Pak se vám zobrazí další nabídka příkazů. Chcete-li si jeden z nich vybrat, přesuňte kurzor myši na požadovaný příkaz a uvolněte tlačítko myši. Některé nabídkové lišty vyžadují, abyste na požadované položce klikli myší. Pak program vyčkává, až klepnete na nějaký příkaz. Také můžete klepnout mimo nabídku a tím dáte programu najevo, že žádný příkaz realizovat nemá.

Obrázek 5.3 – Editor Emacs mění nabídkovou lištu podle toho, s jakým typem souboru zrovna pracujete. Zde je ukázka lišty obsahující nabídky.



Obrázek 5.4 – Na levé straně tohoto terminálového okna je ukázka posuvné lišty vytvořené v systému Athena. Na druhé straně je ukázka posuvné lišty (v okně programu Netscape) vytvořené systémem Motif.

Posuvné lišty

Posuvná lišta je nástroj umožňující zobrazit pouze část dokumentu, zatímco zbytek dokumentu je mimo okno. Například na obrázku 5.4 zobrazuje terminálové okno dolní třetinu textu. Díky posuvným lištám se také snadno určí, jaká část dokumentu je právě zobrazena. Tmavá část posuvné lišty ukazuje relativní pozici právě zobrazené stránky textu a zároveň její relativní velikost. Jestliže se celý text vleze na obrazovku, je celá posuvná lišta zobrazena tmavě. Jestliže je na obrazovce zobrazena polovina textu, pak bude zobrazena tmavě polovina posuvné lišty.

Vertikální posuvné lišty mohou být zobrazeny jak v pravém okraji okna, tak i v levém. Horizontální posuvné lišty bývají zpravidla zobrazeny u dolního okraje okna.

Posuvné lišty v systému Athena

Posuvné lišty v systému Athena fungují jinak, než u ostatních systémů. Každé tlačítko myši má jinou funkci. Má-li se text rolovat nahoru (tj. má-li se zobrazit text nad momentálně zobrazeným textem), pak stačí klepnout pravým tlačítkem myši, přičemž je kurzor myši umístěn kdekoliv v posuvné liště. Pokud chcete text rolovat dolů, pak klepněte levým tlačítkem myši.

Dále si můžete pomocí prostředního tlačítka myši zobrazit text v konkrétní pozici. Stačí klepnout prostředním tlačítkem myši na odpovídající pozici v posuvné liště.

Posuvné lišty v systému Motif

Posuvné lišty v systému Motif fungují téměř stejně jako v operačním systému Windows nebo Macintosh. Příklad posuvné lišty je na obrázku 5.4. Všimněte si, že na začátku a konci posuvné lišty jsou šipky. Ty jsou určeny pro „jemné“ rolování textu. Pokud na jedné z těchto šipek klepnete levým nebo prostředním tlačítkem myši, posune se text jen o několik málo řádků. Pravé tlačítko myši v tomto případě neplní žádnou funkci.

Pokud jde o používání myši uvnitř posuvné lišty, existují významné rozdíly mezi systémy Athena a Motif. Pravé tlačítko myši nemá žádný význam. Klepnutí levým tlačítkem myši nad momentální pozicí tmavé části posuvné lišty způsobí posun textu nahoru. Podobně platí, že klepnutí levým tlačítkem myši pod momentální pozicí tmavé části lišty způsobí posun textu dolů. Dále lze zobrazovaným textem přímo manipulovat tak, že se klepne levým tlačítkem myši na tmavou část posuvné lišty, drží se tlačítko stisknuté a nastaví se pozice tmavé části na požadované místo. Jakmile se tlačítko uvolní, požadovaná pozice se zobrazí.

Prostřední tlačítko má podobnou funkci jako u systému Athena. Zobrazí se ta část textu, která svou pozicí relativně odpovídá pozici kurzoru myši v posuvné liště.

KAPITOLA 6

Práce s operačním systémem Unix

Operační systém Unix je velmi výkonný nástroj pro ty, kdo jej umí používat. V této kapitole si popíšeme pokročilejší techniky práce s příkazovým procesorem `bash`.

Pseudoznaky

V předcházející kapitole jsme se naučili pracovat s příkazy pro manipulaci se soubory. Příležitostně se může stát, že budete chtít pracovat s více soubory najednou. Například budete chtít zkopírovat všechny soubory začínající posloupností „data“ do adresáře `~/backup`. Můžete to udělat tak, že použijete několikrát příkaz `cp`, nebo v jednom příkazu `cp` uvedete seznam všech souborů, které se mají kopírovat. Oba způsoby jsou těžkopádné a jistě vám seberou hodně času. Navíc máte velkou šanci udělat chybu.

Na následujícím příkladě si uvedeme daleko elegantnější postup:

```
/home/larry/report# ls -F
1993-1 1994-1 data1 data5
1993-2 data-new data2
/home/larry/report# mkdir ~/backup
/home/larry/report# cp data* ~/backup
/home/larry/report# ls -F ~/backup
data-new data1 data2 data5
/home/larry/report#
```

Z uvedeného příkladu vidíte, že po zadání hvězdičky zkopíroval příkaz `cp` všechny soubory začínající řetězcem data do adresáře `~/backup`. Zkuste uhodnout, jak bude fungovat příkaz `cp d*w ~/backup`

Co se ve skutečnosti stalo?

Rozhodně dobrá otázka. Příkazový interpret `bash` je schopen interpretovat jisté znaky, kterým se říká pseudoznaky (wildcards). Znak hvězdička (*) je interpretován tak, že za něj dosadí jakoukoliv posloupnost znaků. Proto příkaz z našeho příkladu `cp data* ~/backup` byl interpretován jako příkaz `cp data-new data1 data2 data5 ~/backup`.

Abychom ještě lépe ilustrovali interpretaci znaku hvězdička, uvedeme si nový jednoduchý příkaz `echo`, který zkopíruje své parametry jako text na obrazovce.

```
/home/larry# echo Hello!
Hello!
/home/larry# echo How are you?
How are you?
/home/larry# cd report
/home/larry/report# ls -F
1993-1 1994-1 data1 data5
1993-2 data-new data2
/home/larry/report# echo 199*
1993-1 1993-2 1994-1
/home/larry/report# echo *4*
1994-1
/home/larry/report# echo *2*
1993-2 data2
/home/larry/report#
```

Jak sami vidíte, příkazový procesor rozšířil hvězdičku, vybral všechny soubory vyhovující specifikaci s hvězdičkou a předal je programu, který byl zadán ke spuštění. Naskýtá se přirozená otázka. Co se stane, když specifikaci s hvězdičkou nevyhovuje žádný soubor? Vyzkoušejte si například příkaz `echo /rc/fr*og` a uvidíte – příkazový procesor předá specifikaci přesně tak, jak byla zadána (žádnou interpretaci hvězdičky neprovede).

Ostatní příkazové procesory, například tcsh, se chovají jinak – nepředávají přesnou specifikaci, ale vypíší zprávu `No match`. Uveďme si příklad s příkazovým procesorem `tcsh`:

```
mousehouse>echo /rc/fr*og
echo: No match.
mousehouse>
```

Také vás asi napadá otázka, zda je možné pomocí příkazu `echo` zobrazit například `data*` (a nikoliv seznam souborů vyhovujících specifikaci `data*`). Řešení je jednoduché, stačí požadovaný text uzavřít do uvozovek. Například:

```
/home/larry/report# echo "data*"
data*
/home/larry/report#
```

```
mousehouse>echo "data*"
nebo    data*
mousehouse>
```

Znak otazník

Kromě znaku hvězdičky je příkazový procesor schopen interpretovat jako speciální znak také znak otazník. Uvede-li se ve specifikaci otazník, pak to znamená, že má být nahrazen právě jedním znakem. Například příkaz `ls /etc/??` zobrazí všechny soubory v adresáři `/etc`, jejichž jména se skládají právě ze dvou znaků.

Jak ušetřit čas pomocí příkazu bash

Editování příkazového řádku

Někdy se stane, že napíšete v příkazovém řádku příkazového interpretu `bash` dlouhý příkaz a pak si všimnete, že jste udělali chybu. Samozřejmě můžete postupně smazat všechny znaky, až se dostanete k chybně zadanému znaku, ale pak budete muset celý zbytek příkazu znovu psát. Příkazový interpret `bash` umožňuje používat kurzorové klávesy (šipka vlevo a šipka vpravo), pomocí kterých si můžete nastavit kurzor na chybně zadaný znak a opravit jej.

Dále existuje řada speciálních klávesových zkratek, pomocí nichž můžete příkazový řádek editovat. Tyto klávesové zkratky jsou většinou podobné příkazům používaným v editoru Emacs. Například kombinace Ctrl+T provede výměnu dvou sousedních znaků.[1] Popis většiny důležitých klávesových zkratek najdete v kapitole 8, která je věnována editoru Emacs.

Doplňování příkazů a jmen souborů

Další vynikající vlastnost příkazového procesoru bash spočívá v tzv. doplňování příkazových řádků. Podívejme se na následující příklad.

```
/home/larry# ls -F
this-is-a-long-file
/home/larry# cp this-is-a-long-file shorter
/home/larry# ls -F
shorter        this-is-a-long-file
/home/larry#
```

Samozřejmě je velmi nepohodlné a namáhavé vypisovat každý znak jména souboru `this-is-a-long-file` pokaždé, když chcete tento soubor zpřístupnit. Vytvořte si takový soubor, například příkazem `cp /etc/passwd this-is-a-long-file`. Nyní si ukážeme, jak zadat předcházející příkaz rychleji a s menším rizikem, že uděláme chybu.

Místo vypisování celého jména souboru zadejte `cp th` a pak stiskněte klávesu Tab. Jako mávnutím kouzelného proutku se vám celé jméno souboru objeví v příkazovém řádku a vám již stačí napsat `shorter`. Příkazový procesor `bash` neumí bohužel číst vaše myšlenky, proto jméno souboru `shorter` musíte napsat sami.

Když stisknete klávesu Tab, příkazový interpret `bash` se pokusí najít soubor, který začíná znaky, jež jste doposud napsali. Když například napíši `/usr/bin/ema` a pak stisknu Tab, příkazový procesor bash najde soubor `/usr/bin/emacs`. Když však napíši `/usr/bin/ld` a pak stisknu klávesu Tab, příkazový interpret pípne, aby mě upozornil, že našel více souborů. Skutečně, v adresáři `/usr/bin` jsou soubory `ld`, `ldd` a `ld86`.

Jestliže se pokoušíte doplnit jméno souboru a příkazový procesor pípne, pak můžete znovu stisknout klávesu Tab a objeví se vám seznam všech nalezených souborů. Proto nemusíte znát přesně jména svých souborů, příkazový procesor vám je vždy tímto způsobem připomene.

1 Zkratka Ctrl+T znamená, že máte stisknout klávesu Ctrl, držet ji stisknutou, a pak stisknout klávesu T.

Standardní vstup a standardní výstup

Zkuste si provést příkaz, který vypíše seznam všech souborů v adresáři `/usr/bin`: `ls /usr/bin`. Adresář `/usr/bin` obsahuje velké množství souborů, proto po zobrazení jejich jmen zůstane na obrazovce jen několik a ostatní „utečou“ nahoru. Jak tento problém řešit?

Standardní vstup a výstup

Operační systém Unix nabízí programátorům velmi jednoduchý způsob zápisu na terminál. Pokud nějaký program něco vypisuje na vaši obrazovku, pak používá tzv. **standardní výstup** (standard output). Standardní výstup se zkracuje zkratkou stdout a slouží k předávání informací uživateli. Na druhé straně existuje **standardní vstup** (standard input), který slouží k předávání informací od uživatele. Samozřejmě je možné, aby program komunikoval s uživatelem bez použití standardního vstupu a výstupu, ale většina programů, kterými se zabýváme v této knize, standardní vstup a standardní výstup používá.

Například příkaz `ls` vypisuje seznam adresářů a souborů na standardní výstup, jenž je normálně spojen s terminálem. Příkazový interpret `bash` čte vámi zadané příkazy ze standardního vstupu.

Programy také mohou zapisovat do tzv. **standardního chybového výstupu** (standard error). Standardní chybový výstup je téměř výlučně spjat s terminálem, proto mohou být uživatelé informováni o případných chybách.

V následujících odstavcích si ukážeme, jak využívat standardní vstup a standardní výstup v konstrukcích, kterým se říká přesměrování vstupu, přesměrování výstupu a roura vstupu/výstupu.

Přesměrování výstupu

Velmi důležitou vlastností operačního systému Unix je schopnost přesměrovat výstup. Tato vlastnost vám umožní uložit výsledky příkazu do souboru nebo vytisknout na tiskárně. Jestliže chcete například přesměrovat výstup z příkazu `ls /usr/bin` do souboru, přidejte na konec příkazu znak `>` a za něj uveďte jméno souboru (nejlépe neexistujícího). Například:

```
/home/larry# ls
/home/larry# ls -F /usr/bin > listing
/home/larry# ls
listing
/home/larry#
```

Jak vidíte, seznam souborů se nezobrazil na terminálu, ale místo toho se vytvořil ve vašem domovském adresáři nový soubor `listing`. Zkusme se na tento soubor podívat pomocí příkazu `cat`. Jestli si vzpomínáte, příkaz `cat` se jevil jako zbytečný příkaz, který zkopíroval na obrazovku (standardní výstup) to, co jste napsali (standardní vstup). Příkaz `cat` také umí vypsat obsah souboru na standardní výstup, pokud tento soubor uvedete jako parametr:

```
/home/larry# cat listing
 ...
/home/larry#
```

Soubor `listing` nyní obsahuje přesný výstup příkazu `ls /usr/bin`, tedy seznam všech souborů v adresáři `/usr/bin`. To je v pořádku, ale stále to neřeší náš původní problém.[2]

2 Pro netrpělivé čtenáře poznamenejme, že zde mohou použít program `more`. Než se však k němu dostaneme, musíme ještě několik věcí vysvětlit.

Nyní je příkaz `cat` o něco zajímavější – lze jej použít spolu s přesměrováním výstupu. Co asi udělá příkaz `cat listing > newfile`? Přesměrování `> newfile` pro příkazový interpret znamená toto: „vezmi celý výstup z příkazu a ulož jej do nového souboru“. Výstup z příkazu `cat listing` je ovšem soubor `listing`. Jinými slovy, pravě jsme popsali jeden ze způsobů kopírování souborů, i když ne příliš efektivní.

Co asi udělá příkaz `cat > fox`? Příkaz `cat` čte data ze standardního vstupu (v tomto případě terminálu) a kopíruje je na standardní výstup, dokud nenarazí na znak konce souboru Ctrl+D. V tomto případě bude standardní vstup přesměrován do souboru `fox`. Nyní nám příkaz `cat` slouží jako základní editor:

```
/home/larry# cat > fox
The quick brown fox jumps over the lazy dog.
```

Stiskni Ctrl+D

Právě jste vytvořili soubor se jménem `fox` obsahující větu „`The quick brown fox jumps over the lazy dog.`“ Další důležitou funkcí příkazu `cat` je spojování souborů dohromady. Příkaz `cat` bude vypisovat každý soubor, který mu byl předán jako parametr. Proto například příkaz `cat listing fox` nejprve vypíše seznam souborů adresáře `/usr/bin` a nakonec vypíše naši hloupou větu. Příkaz `cat listing fox > listandfox` vytvoří nový soubor tvořený obsahem souborů `listing` a `fox`.

Přesměrování vstupu

Tak, jako je možné přesměrovat standardní výstup, je možné přesměrovat i standardní vstup. Místo toho, aby program četl z klávesnice, bude číst ze souboru. Protože se přesměrování vstupu vztahuje k přesměrování výstupu, je přirozené pro něj vyhradit znak `<`. Tento znak se také uvádí za příkazem, který chcete realizovat.

Přesměrování vstupu je užitečné zejména v případě, kdy máte soubor obsahující data a program, který data čte se standardního vstupu. Na druhé straně platí, že většina programů vyžaduje specifikaci souboru se vstupními daty, proto se přesměrování vstupu nepoužívá tak často, jako přesměrování výstupu.

Roura

Řada příkazů v operačním systému Unix produkuje velké množství informací. Jak jsme si již ukázali, příkaz `sp /usr/bin` vyprodukuje příliš mnoho informací, než aby mohly být zobrazeny na terminálu. Abyste si mohli prohlédnout všechny informace z příkazů, jako je `ls /usr/bin`, budete muset použít další důležitý příkaz operačního systému Unix, který má název `more`.[3] Příkaz `more` způsobí pozastavení výpisu na obrazovku, jakmile se celá obrazovka zaplní. Například příkaz `more < /etc/rc` zobrazí stejným způsobem soubor `/etc/rc` jako příkaz `cat`, ale s tím rozdílem, že výpis pozastaví po každém naplnění obrazovky.

Příkaz `more` také můžete použít ve tvaru `more /etc/rc`, což představuje normální způsob jeho použití.

Co je však platné, že příkaz `more` umožňuje zobrazit více informací, než se vleze na obrazovku? Příkaz `more < ls /usr/bin` nebude fungovat, protože přesměrování vstupu funguje pouze se soubory a nikoliv s příkazy. Budete tedy muset postupovat takto:

3 Název `more` pochází z nápovědy tohoto programu, kterou byl původně řetězec --more--. V mnoha verzích operačního systému Linux se vyskytuje identický příkaz, který postupně programátoři zdokonalovali.

```
/home/larry# ls /usr/bin > temp-ls
/home/larry# more temp-ls
...
/home/larry# rm temp-ls
```

Naštěstí nabízí operační systém Unix mnohem elegantnější způsob. Stačí, když zadáte příkaz `ls /usr/bin | more`. Znak | indikuje tzv. **rouru**. Roura v operačním systému Unix řídí tok dat.

Užitečnost roury ještě více vzrůstá ve spojení s dalšími nástroji, kterým se říká **filtry**. Filtr je program, který čte standardní vstup, nějakým způsobem jej modifikuje a odešle jej na standardní výstup. Příkladem filtru je právě příkaz `more` – čte data ze standardního vstupu, zobrazuje je na standardní výstup (obrazovku) a umožňuje vám prohlédnout celý soubor. `more` však není úplně dokonalý filtr, protože jeho výstup není vhodný pro to, aby mohl být předán jako vstup jinému programu.

Mezi další filtry patří příkazy `cat`, `sort`, `head` a `tail`. Chcete-li například přečíst pouze prvních deset řádků výstupu z příkazu `ls`, můžete použít příkaz `ls /usr/bin | head`.

Současný běh úloh

Jak řídit úlohy

Řízení úloh (job control) představuje možnost zajistit, aby daný proces (proces není v podstatě nic jiného, než běžící program) běžel na pozadí nebo naopak, aby běžel na popředí. Jinými slovy, zřejmě potřebujete nástroje k tomu, abyste mohli dlouhodobě běžící program přesunout do pozadí a tak mohli pracovat na jiných věcech, a přitom měli možnost do programu běžícího na pozadí kdykoliv zasahovat. Tímto nástrojem je v operačním systému Unix příkazový procesor. Ukážeme si, jak jej využívat k řízení úloh.

V souvislosti s řízením úloh jsou v operačním systému Unix vyhrazena dvě důležitá klíčová slova. První z nich je `fg` pro běh programů v popředí a druhé je `bg` pro běh programů na pozadí. Abychom vyzkoušeli, jak fungují, použijeme příkaz `yes`:

```
/home/larry# yes
```

Po spuštění příkazu `yes` se na levé straně obrazovky vypisuje velkou rychlostí písmeno „y“.[4] Pokud byste chtěli příkaz `yes` zastavit, stiskli byste klávesu Ctrl+C. Místo toho však stiskněte klávesu Ctrl+Z. Nyní se vám bude zdát, že se provádění příkazu `yes` zastavilo. Na obrazovce se však objeví následující zpráva:

```
[1]+ Stopped          yes
```

To znamená, že proces `yes` byl pozastaven. Pozastavený proces můžete obnovit pomocí příkazu `fg`, který jej aktivuje v popředí a program poběží dále. Zatímco je program pozastaven, můžete realizovat další příkazy. Než zadáte příkaz `fg`, zkuste si zadat několikrát jiný příkaz, například `ls`.

Jakmile se příkaz `yes` „vrátí“ do popředí, bude pokračovat ve výpisu písmen „y“ stejnou rychlostí jako na počátku po jeho spuštění. Nemusíte mít obavy, že se výstup z programu, který byl pozastaven, někam ztratí. Je-li program pozastaven uvedeným způsobem, pak až do okamžiku, kdy jej obnovíte, žádný výstup neprodukuje. Nyní stiskněte klávesu Ctrl+C a program `yes` zrušte.

4 Možná se vám zdá příkaz `yes` divný. Řada příkazů však potřebuje potvrdit nějakou volbu, proto v operačním systému Unix takový příkaz existuje.

Nyní se vraťme k předcházející zprávě:

```
[1]+ Stopped          yes
```

Číslo v hranatých závorkách je **pořadové číslo úlohy**, na které se můžete odvolávat, kdykoliv budete chtít na běhu této úlohy něco změnit. Číslo úlohy se zavádí z toho důvodu, aby existovalo nějaké rozlišení mezi více současně běžícími úlohami. Znak „+“ za hranatými závorkami indikuje, že uvedená úloha je aktuální úlohou, tedy úlohou, jež byla jako poslední převedena z popředí na pozadí. Jestliže zadáte příkaz `fg`, pak do popředí převedete právě tu úlohu, která je označena znakem „+“. Slovo `Stopped` znamená, že úloha byla pozastavena a nachází se ve zvláštním stavu – není zrušena, ale také není aktivní. Operační systém Linux ji uvedl do speciálního stavu, kdy může pokračovat v realizaci, jakmile bude zadán příslušný příkaz. Jako poslední je uvedeno jméno procesu, tedy v našem případě `yes`.

Než pokročíme dále, zrušme tuto úlohu a nastartujme ji, tentokrát jiným způsobem. Příkaz ke zrušení úlohy se jmenuje `kill` (doslova zabít, zničit) a používá se následujícím způsobem:

```
/home/larry# kill %1
[1]+ Stopped yes
/home/larry#
```

Zpráva, která se objevila na obrazovce (říká, že proces byl pozastaven) je zavádějící. Abychom zjistili, v jakém stavu se proces nachází (t.j. zda je aktivní, zmrazen nebo pozastaven), zadáme příkaz `jobs`:

```
/home/larry# jobs
[1]+ Terminated yes
/home/larry#
```

Až zde máte napsáno, v jakém stavu se úloha `yes` skutečně nachází – byla ukončena a nikoliv pozastavena. Je také možné, že zadáte příkaz `jobs` a vůbec žádná zpráva se nevypíše. To znamená, že na pozadí opravdu neběží žádná úloha. Jestliže jste právě zrušili nějakou úlohu a příkaz `jobs` nic nevypíše, pak jste ji zrušili úspěšně. Obvykle se ale objeví zpráva, že úloha byla ukončena (`Terminated`).

Nyní spusťme příkaz `yes` znovu, tentokrát následujícím způsobem:

```
/home/larry# yes > /dev/null
```

Pokud jste četli oddíl o přesměrování vstupu a výstupu, pak asi tušíte, že příkaz `yes` bude odesílat svůj výstup do souboru `/dev/null`. Soubor `/dev/null` má v operačním systému Unix speciální význam. Představuje „černou díru“, do které se ztratí jakékoliv množství informací, aniž by například hrozilo, že se zaplní váš pevný disk. Nemáte-li dostatek fantazie, pak si představte, že je ve vašem počítači vyvrtána díra, kterou proudí informace ven a ztrácejí se v hlubinách vesmíru.

Po zadání uvedeného příkazu se vám nevrátí nápověda příkazového řádku, ani se nebudou na obrazovce vypisovat písmena „y“. I když je výstup z příkazu `yes` přesměrován do souboru `/dev/null`, úloha stále běží na popředí. Jako obvykle ji můžete pozastavit pomocí klávesy Ctrl+Z. Pak se vám samozřejmě znovu objeví výzva příkazového řádku:

```
/home/larry# yes > /dev/null
[Příkaz "yes" běží, nyní stiskněme Ctrl+Z]
[1]+ Stopped          yes > /dev/null
/home/larry#
```

Je nějaká možnost přesunout úlohu do pozadí tak, aby dále běžela a aby se nám objevila výzva příkazového řádku? K tomuto účelu slouží příkaz `bg`:

```
/home/larry# bg
[1]+ yes > /dev/null &
/home/larry#
```

Nyní byste měli věřit následujícímu tvrzení: po zadání příkazu `bg` běží proces `yes > /dev/null` dále, a to na pozadí. Poznáte to podle toho, že když v příkazovém řádku zadáte nějaký příkaz, například `ls` nebo `stuff`, bude jeho realizace poněkud zpomalena. Jiný efekt úlohy běžící na pozadí nemají. Nadále můžete v příkazovém řádku zadávat jakékoliv příkazy a proces `yes > /dev/null` poběží na pozadí a přitom bude výstup odesílat do „černé díry".

Existují dvě možnosti, jak proces běžící na pozadí zrušit (doslova „zabít"). Buď použijete příkaz `kill`, nebo přesunete proces do popředí a ukončíte jej pomocí klávesy Ctrl+C. Vyzkoušíme si druhý způsob, abychom lépe pochopili vztah mezi příkazy `fg` a `bg`:

```
/home/larry# fg
yes > /dev/null
[Nyní běží proces opět v popředí. Stiskněte klávesu Ctrl+C a ukončete jej.]
/home/larry#
```

Jako další si vyzkoušíme spustit současně více procesů, například:

```
/home/larry# yes > /dev/null &
[1] 1024
/home/larry# yes | sort > /dev/null &
[2] 1026
/home/larry# yes | uniq > /dev/null
[nyní stiskněte klávesu Ctrl+Z]
[3]+ Stopped yes | uniq > /dev/null
/home/larry#
```

Určitě jste si všimli, že jsme na konec prvních dvou příkazů zapsali znak `&`. Pokud na konec příkazu zadáte znak `&`, příkazový procesor spustí úlohu hned od počátku na pozadí. Je to mnohem jednodušší, než spustit úlohu normálním způsobem, stisknout klávesu Ctrl+Z a pak zadávat příkaz `bg`. První dvě úlohy jsme tedy přímo spustili na pozadí. Třetí úloha je v tomto okamžiku pozastavena a tedy neaktivní. Jistě jste zpozorovali, že počítač nyní reaguje na další příkazy pomaleji, protože dvě běžící úlohy spotřebovávají jistý čas procesoru.

Zrušme nyní druhou úlohu, protože ta patrně výrazně snižuje výkon vašeho počítače. Mohli bychom použít příkaz `kill %2`, ale to by bylo příliš jednoduché. Místo toho zadejme následující příkazy:

```
/home/larry# fg %2
yes | sort > /dev/null
[Stiskněte klávesu Ctrl+C]
/home/larry#
```

Uvedený příklad demonstruje, že příkaz `fg` akceptuje parametry začínající znakem `%`. Ve skutečnosti jsme mohli použít následující posloupnost příkazů:

```
/home/larry# %2
yes | sort > /dev/null
[Stiskněte klávesu Ctrl+C]
/home/larry#
```

První příkaz bude opět fungovat, protože příkazový procesor interpretuje číslo úlohy jako požadavek, že má být úloha přenesena do popředí. Čísla úloh odlišuje od ostatních čísel právě pomocí znaku %. Nyní zadejte příkaz `jobs`, abyste věděli, které úlohy momentálně běží:

```
/home/larry# jobs
[1]- Running yes > /dev/null &
[3]+ Stopped yes | uniq > /dev/null
/home/larry#
```

Znak „-“ znamená, že úloha číslo 1 byla do pozadí přesunuta jako první v pořadí a bude jako druhá v pořadí přesunuta do popředí, pokud zadáte příkaz `fg` bez parametrů. Znak „+“ znamená, že úloha číslo 3 bude do popředí přesunuta jako první v pořadí, pokud zadáte příkaz `fg` bez parametrů. Uvedené implicitní pořadí můžete změnit takto:

```
/home/larry# fg %1
yes > /dev/null
[nyní stiskněte klávesu Ctrl+Z]
[1]+ Stopped yes > /dev/null
/home/larry#
```

Pomocí uvedených příkazů jste pozastavili úlohu číslo 1 a změnili jste také pořadí priorit všech úloh. K jakým změnám došlo? Na to vám odpoví příkaz `jobs`:

```
/home/larry# jobs
[1]+ Stopped yes > /dev/null
[3]- Stopped yes | uniq > /dev/null
/home/larry#
```

Nyní jsou obě úlohy pozastaveny (v obou případech jsme použili klávesu Ctrl+Z). Úloha číslo 1 je nyní na pozici první, pokud jde o pořadí, v jakém budou úlohy implicitně přenášeny do popředí. Je tomu tak z toho důvodu, že jsme úlohu nejprve přesunuli do popředí a pak ji pozastavili. Znak „+“ vždy označuje tu úlohu, která byla jako poslední pozastavena, když předtím běžela na popředí. Úlohu číslo 1 můžeme opět aktivovat tímto způsobem:

```
/home/larry# bg
[1]+ yes > /dev/null &
/home/larry# jobs
[1]- Running yes > /dev/null &
[3]+ Stopped yes | uniq > /dev/null
/home/larry#
```

Všimněte si, že úloha 1 nyní běží a u druhé úlohy se objevil znak „+“. Nakonec obě úlohy zrušíme, protože se už nemůžeme dále dívat na to, jak snižují výkon počítače:

```
/home/larry# kill %1 %2
[3] Terminated yes | uniq > /dev/null
/home/larry# jobs
[1]+ Terminated yes > /dev/null
/home/larry#
```

Jistě jste si všimli zpráv upozorňujících na ukončení úloh – jak se zdá, nic neumírá tiše. Tabulka 6.1 obsahuje stručný přehled o příkazech, které se vztahují k řízení úloh.

Teorie řízení úloh

Především je nutné pochopit, že úlohy řídí příkazový procesor. Ve skutečnosti v systému neexistují programy jako fg, bg, jobs, & nebo kill. Tyto programy jsou vnitřními příkazy příkazového procesoru a jsou jeho součástí. (Výjimku někdy tvoří příkaz `kill`, který je v některých operačních systémech Unix implementován jako externí program. Pokud však jde o Linux, příkaz `kill` je integrován v příkazovém procesoru bash). Tento mechanismus řízení úloh je logický. Každý uživatel chce mít k realizaci úloh svůj prostor. Protože každý uživatel pracuje se svým příkazovým procesorem, je logické, aby příkazový procesor úlohy řídil. Odtud dále vyplývá, že čísla úloh se vztahují právě k jednomu uživateli. Moje úloha číslo 1 bude zřejmě zcela odlišná od vaší úlohy číslo 1, a to i v případě, kdy budeme oba přihlášeni k témuž systému. Ve skutečnosti platí, že přihlásíte-li se do systému více než jednou, bude každý spuštěný příkazový procesor vlastnit unikátní datové struktury pro řízení úloh. Jako jediný uživatel tedy můžete mít několik zcela různých úloh s pořadovým číslem 1 běžících v různých příkazových interpretech.

`fg %job`	Tento příkaz je příkazem příkazového procesoru a přesouvá úlohu do popředí. Chcete-li zjistit, která úloha se přesune implicitně jako první, zadejte příkaz `jobs` a podívejte se, která úloha je označena znakem „+“. Parametry: volitelným parametrem je číslo úlohy. Implicitní je úloha označená znakem „+“.
`&`	Pokud se znak „&“ uvede na konec příkazového řádku, bude úloha spuštěna automaticky přímo na pozadí. Pro takto spuštěný proces platí všechny metody řízení úloh, které jsme zde uvedli.
`bg %job`	Tento příkaz je příkazem příkazového procesoru a přesouvá úlohu do pozadí. Chcete-li zjistit, která úloha se přesune implicitně jako první, zadejte příkaz `jobs` a podívejte se, která úloha je označena znakem „+“. Parametry: volitelným parametrem je číslo úlohy. Implicitní je úloha označená znakem „+“.
`kill %job PID`	Tento příkaz je příkazem příkazového procesoru a ukončuje úlohu, ať běží na pozadí nebo byla pozastavena. Pokaždé byste měli použít jako parametr číslo úlohy, kterému předchází znak `%`. Parametry: buď číslo úlohy, kterému předchází znak `%`, nebo PID (identifikační číslo procesu), pak znak `%` není nutné uvádět. Na jednom příkazovém řádku může být uvedeno více úloh, které se mají ukončit.
`jobs`	Tento příkaz je příkazem příkazového procesoru. Vypisuje informace o běžících úlohách a o úlohách, které byly pozastaveny.
Ctrl+C	Klávesová zkratka vyhrazená k bezpodmínečnému ukončení úlohy. Běžně se používá k ukončení úlohy běžící v popředí. Je však nutné upozornit na skutečnost, že ne všechny programy na tuto klávesovou zkratku reagují.
Ctrl+Z	Klávesová zkratka vyhrazená k pozastavení úlohy. Opět platí, že některé programy ji ignorují. Jakmile je jednou úloha pozastavena, může být znovu přesunuta do popředí nebo ukončena.

Tabulka 6.1 – Přehled příkazů a kláves používaných k řízení úloh

Jediný bezpečný způsob, jak identifikovat proces, představuje identifikační číslo procesu (process identification number, zkratka PID). Každá úloha běžící v systému má svoje unikátní identifikační číslo procesu. Dva různí uživatelé mohou k odkazu na proces použít totéž identifikační číslo a pak mají jistotu, že se jedná o tentýž proces (samozřejmě za předpokladu, že jsou přihlášeni k témuž systému).

Podívejme se na další příkaz, který vám pomůže lépe pochopit význam identifikačních čísel procesů. Příkaz `ps` vypíše seznam všech běžících nebo pozastavených procesů, včetně vašeho příkazového procesoru. Tento příkaz má také několik voleb, z nichž nejdůležitější jsou `a`, `u` a `x`. Zadáte-li volbu `a`, pak se vám zobrazí seznam všech procesů, t.j. procesů, které spustili i ostatní uživa-

telé. Pomocí volby `x` se zobrazí seznam těch procesů, které nejsou nijak svázány s terminálem.[5] Poslední volba `u` zajistí, že se vám zobrazí další užitečné informace o běžících nebo pozastavených procesech.

Jestliže chcete získat komplexní představu, co se ve vašem systému odehrává, zadejte příkaz `ps -aux`. (V novějších verzích je možno znaménko „–“ vynechat a použít pouze příkaz `ps aux`.) Pak se můžete podívat, kolik paměti každý proces potřebuje (sloupec `%MEM`) a jak zatěžuje procesor (sloupec `%CPU`). Ve sloupci `TIME` je uveden celkový čas procesoru, který spuštěný proces spotřeboval.

Ještě jedna poznámka o identifikačním číslu procesu. Kromě parametru `%číslo úlohy` můžete v příkazu `kill` použít identifikační číslo procesu. Spusťte příkaz `yes > /dev/null` na pozadí, pak spusťte příkaz `ps` a podívejte se na identifikační číslo náležící procesu `yes`. Toto identifikační číslo můžete také použít ke zrušení procesu `yes`.[6]

Jestliže začnete programovat v jazyku C, pak se brzy dozvíte, že příkazy pro řízení úloh jsou interaktivní verze systémových volání `fork` a `execl`. Jedná se o příliš složité mechanismy, než abychom je zde mohli popsat. Pokud budete vytvářet programy, které jsou schopny spouštět více procesů, pak se s těmito systémovými funkcemi budete muset seznámit v příslušné dokumentaci.

Virtuální konzoly

Operační systém Linux podporuje tzv. virtuální konzoly. Jedná se o metodu, která budí dojem, že váš počítač není jedním počítačem ale několika počítači najednou. Linux je schopen spustit několik terminálů současně a přitom jsou všechny spojeny s jedním jádrem. Naštěstí je používání virtuálních konzol jednou z nejjednodušších záležitostí v operačním systému Linux. Chcete-li si je vyzkoušet, stiskněte klávesu Alt a pak klávesu F2.[7]

Najednou se vám objeví nová výzva k přihlášení se do systému. Nepodléhejte panice. Nyní pracujete s virtuální konzolou číslo 2. Přihlaste se a proveďte několik příkazů, abyste se přesvědčili, že nově nastartovaný příkazový procesor skutečně funguje. Budete-li se chtít vrátit do virtuální konzoly číslo 1, stiskněte opět klávesu Alt a pak F1. Nebo můžete vytvořit třetí konzolu pomocí kláves Alt a F3.

Operační systém Linux má implicitně povoleno šest virtuálních konzol. Pokud budete chtít vědět jak, prostudujte si manuál *„Příručka správce operačního systému Linux“*. Budete muset udělat nějaké úpravy v souborech v adresáři `/etc`. Šest virtuálních konzol však většině uživatelů stačí.

Začnete-li používat virtuální konzoly, brzy zjistíte, že jsou ideálním nástrojem k realizaci mnoha úkonů současně. Na první konzole můžete například provozovat editor Emacs, na druhé programy pro komunikaci s Internetem a na třetí můžete mít spuštěn příkazový procesor, kdybyste chtěli spustit ještě nějaký další program.

---

5 To má smysl jen v případě jistých systémových programů, které nekomunikují s uživatelem prostřednictvím klávesnice.

6 Obecně je mnohem jednodušší zrušit úlohu pomocí jejího čísla.

7 Ujistěte se, že pracujete s textovou konzolou. V případě systému X Window by uvedená kombinace kláves nemusela fungovat.

KAPITOLA 7

Malé a výkonné programy

V čem spočívá síla operačního systému Unix

Síla operačního systému Unix spočívá v používání malých a jednoduchých příkazů, které se sami o sobě zdají neužitečné. Pokud se je naučíte spojovat dohromady, vytvoříte systém, jenž je mnohem pružnější a výkonnější než všechny ostatní operační systémy. V této kapitole se budeme zabývat příkazy `sort`, `grep`, `more`, `cat`, `wc`, `spell`, `diff` a `tail`. Z názvů těchto programů nelze bohužel intuitivně předpokládat, jakou funkci plní.

Nejdříve se budeme věnovat každému příkazu odděleně a nakonec uvedeme několik příkladů, jak je spojovat a používat dohromady.[1]

Práce se soubory

Kromě příkazů `cd`, `mv` a `rm`, o kterých jsme se zmínili v kapitole 4, existují další příkazy určené k manipulaci se soubory (a nikoliv s daty, jež soubory obsahují). Patří mezi ně `touch`, `chmod`, `du` a `df`. Žádný z těchto příkazů se nestará o obsah souborů – pouze mění některé jejich atributy, se kterými operační systém Unix pracuje.

Uveďme si seznam atributů, se kterými uvedené příkazy manipulují:

- Časové údaje. S každým souborem jsou spjaty tři časové údaje.[2] První časový údaj obsahuje informaci o době vytvoření souboru, druhý o době jeho poslední modifikace a třetí o době, kdy byl naposledy zpřístupněn.
- Vlastník souboru. Každý soubor v operačním systému Unix je vlastněn některým uživatelem.
- Skupina. Každý soubor je také spojen se skupinou uživatelů. Nejčastěji se tato skupina nazývá `users`, což znamená, že soubor je sdílen každým uživatelem přihlášeným do systému.
- Přístupová práva. Každý soubor má přístupová práva (pro která se někdy používá označení privilegia). Jedná se o mechanismus, jenž určuje, kdo může k danému souboru přistupovat, kdo může spouštět dané programy a podobně. Každé z přístupových práv může být odděleně přiřazeno vlastníku souboru, skupině nebo všem uživatelům operačního systému.

1 Poznamenejme, že výklad uvedených příkazů nebude zcela vyčerpávající. Podrobnosti naleznete v manuálových stránkách.

2 Starší souborové systémy v operačním systému Linux uchovávaly pouze jeden časový údaj, protože byly odvozeny od operačního systému Minix. Pokud máte takový souborový systém, pak pro vás budou některé informace neaktuální.

`touch` *`soubor1 soubor2 ... souborN`*

Příkaz `touch` provede aktualizaci časových údajů každého souboru uvedeného jako parametr – nastaví čas vytvoření na aktuální čas. Pokud soubor uvedený jako parametr neexistuje, program touch jej vytvoří. Je také možné specifikovat čas, který má být souborům přiřazen. Podrobnosti si nastudujte v manuálových stránkách.

`chmod [-Rfv]` *`mód soubor1 soubor2 ... souborN`*

Příkaz umožňující změnit přístupová práva se nazývá `chmod` (change mode). Než se pustíme do jeho popisu, musíme si probrat, jaká přístupová práva operační systém Unix bere v úvahu. Každý soubor je spojen se skupinou přístupových práv. Podle těchto přístupových práv operační systém Unix určuje, zda může být soubor čten, zda do něj může být zapisováno, nebo, jedná-li se o spustitelný program, může být spuštěn. V následujících odstavcích budeme hovořit o přístupových právech uživatelů. Každý program, jenž uživatel spustí, má shodná přístupová práva. Pokud přesně nevíte, co program dělá, mohou nastat problémy s bezpečností systému.

Operační systém Unix rozlišuje tři různé typy uživatelů. Prvním z nich je vlastník souboru. Tím je osoba, která má právo v souvislosti s tímto souborem používat příkaz `chmod`. Druhý typ představuje skupina. Většina souborů ve vašem systému by měla mít přiřazen atribut „`users`“, a tak by měla být přístupna každému normálnímu uživateli. Chcete-li znát hodnotu atributu skupina, zadejte příkaz `ls -l` *`file`*.

Nakonec operační systém identifikuje ty osoby, které nejsou ani vlastníky souboru, ani členové jeho skupiny. Pro ty je vyhrazen atribut „`other`“ (ostatní).

Typické nastavení přístupových práv je: pro vlastníka právo číst soubor a zapisovat do souboru, pro skupinu právo číst soubor a pro ostatní jsou všechna práva potlačena. Může se však stát, že skupina má právo soubor číst i do něj zapisovat a přitom vlastník nemá žádná práva.

Nyní si vyzkoušejme, jak pomocí příkazu `chmod` změnit některá přístupová práva. Nejdříve si vytvořte nový soubor, řekněme pomocí příkazu `cat` nebo pomocí editoru Emacs. Implicitně je vám přiděleno právo soubor číst i právo do něj zapisovat. Práva ostatních uživatelů jsou určena podle toho, jak je nastaven váš systém a jak je nastaven proces přihlašování. Ujistěte se, že nově vytvořený soubor jste schopni číst pomocí příkazu `cat`. Nyní si odeberte právo číst tento soubor pomocí příkazu `chmod u-r` *`filename`*. Parametr `u-r` se interpretuje jako „user minus read“. Když se nyní pokusíte soubor přečíst, obdržíte chybové hlášení `Permission denied!`. Chcete-li získat zpět právo soubor číst, zadejte příkaz `chmod u+r` *`filename`*.

Přístupová práva k adresářům používají stejné atributy (pro čtení, zapisování a spouštění), ale poněkud odlišným způsobem je interpretují. Právo číst znamená, že uživatel, skupina nebo ostatní mohou vypisovat seznam souborů v adresáři. Právo zapisovat znamená, že uživatel, skupina nebo ostatní mohou přidávat soubory do adresáře nebo rušit soubory. Právo spouštět znamená, že uživatel může zpřístupňovat soubory v adresáři i v jeho podadresářích. Pokud nemá uživatel žádná práva, nemůže ani použít příkaz `cd`.

Při používání příkazu `chmod` se specifikuje, kdo má k danému souboru přístupová práva (uživatel, skupina, ostatní nebo všichni). Dále se specifikují atributy, které definují, co se s daným souborem může dělat. Znaménko plus indikuje udělení daného práva, znaménko mínus popření. Pomocí znaménka rovná se (=) se specifikují přesná přístupová práva. Přípustná přístupová práva jsou read (čtení), write (zápis) a execute (spouštění).

Pokud se použije v příkazu chmod volba R, pak se změna přístupových práv týká všech souborů v adresáři a všech podadresářů (R je označením pro rekurzivní). Jestliže se použije volba f, pak se příkaz chmod pokusí změnit přístupová práva i v případě, že uživatel není vlastníkem daného souboru. Zadá-li se volba v, pak bude příkaz chmod podrobně vypisovat informace o všech krocích, které realizuje.

Systémová statistika

Příkazy uvedené v tomto oddílu jsou určeny k výpisu informací o systému nebo o jeho částech.

`du [-abs] [cesta1 cesta2 ... cestaN]`

Příkaz du je zkratkou pro „disk usage“ (využívání disku). Zobrazuje informaci o velikosti diskového prostoru, který je přidělen danému adresáři a všem jeho podadresářům. Samotný příkaz du zobrazuje seznam, jehož položky uvádějí u každého adresáře velikost diskového prostoru (který adresář obsazuje), kolik prostoru obsazuje aktuální adresář a jako poslední udává velikost diskového prostoru spotřebovaného aktuálním adresářem a všemi jeho podadresáři. Pokud příkazu du předáte nějaké parametry (jméno souboru nebo adresáře), pak vám vypíše uvedené informace o tomto souboru či adresáři.

Použije-li se volba a, pak se vypíší uvedené informace jak o souborech, tak i o adresářích. Po zadání volby b se informace o velikosti diskového prostoru nebudou uvádět v kilobajtech, ale přímo v bajtech. Jeden bajt je ekvivalentní jednomu znaku v textovém souboru.

Pokud se použije volba s, pak příkaz du vypíše zmíněné informace o adresářích uvedených jako parametry a nikoliv o jejich podadresářích.

`df`

Příkaz df je zkratkou pro „disk filling“. Sumarizuje množství použitého diskového prostoru. Pro každý souborový systém (připomeňme, že souborový systém zaujímá buď samostatný disk, nebo samostatný diskový oddíl) vypíše informaci o celkovém množství diskového prostoru, informaci o velikosti použité části, o volné kapacitě a nakonec o celkové kapacitě souborového systému.

Paradoxně se může stát, že se vám zobrazí kapacita větší než 100 %. Je to způsobeno tím, že operační systém Unix rezervuje pro každý souborový systém jistý prostor pro superuživatele. Proto je zajištěno, že i když uživatel zaplní celý disk, zůstává na disku něco málo volného místa, aby se mohly realizovat některé operace.

Pro většinu uživatelů nemá příkaz df žádné užitečné volby.

`uptime`

Příkaz uptime dělá přesně to, co byste podle jeho názvu očekávali. Vypisuje celkový čas, který uplynul od posledního zavedení operačního systému.

Dále příkaz uptime uvádí informaci o aktuálním čase a o tzv. „load average“. Termínem „load average“ se míní průměrný počet úloh čekajících na spuštění v průběhu daného časového intervalu. Příkaz uptime uvádí tyto hodnoty pro poslední minutu, posledních pět minut a posledních deset minut. Jestliže se hodnota „load average“ blíží k nule, pak to znamená, že je systém téměř nevytížen. Hodnota blízká k jedničce znamená, že je systém plně vytížen, ale není zahlcen. Vysoké hodnoty jsou výsledkem skutečnosti, že v systému běží několik programů současně.

Příkaz uptime patří k několika málo programům operačního systému Unix, které nemají parametry a žádné volby. (V nových verzích programu uptime, který je součástí balíku procps, je akceptována volba V, která je určena pro zobrazení verze balíku procps).

`who`

Příkaz `who` slouží k zobrazení seznamu momentálně přihlášených uživatelů systému. Pokud se k příkazu přidají parametry `am i`, pak se zobrazí informace o aktuálním uživateli.

`w [-f] [`*`uživatelské jméno`*`]`

Příkaz `w` zobrazuje informace o momentálních uživatelích operačního systému a informace o tom, co dělají. V podstatě tento příkaz kombinuje příkazy `uptime` a `who`. Záhlaví výstupu z příkazu `w` je přesně stejné jako v případě příkazu `uptime` a každý řádek obsahuje informace o uživateli, například kdy se přihlásil. Kolonka `JCPU` ukazuje celkový čas procesoru, který uživatel spotřeboval, zatímco kolonka `PCPU` ukazuje celkový čas procesoru spotřebovaný jejich momentálně běžící úlohou.

Jestliže u příkazu `w` uvedete volbu `f`, pak se nezobrazí vzdálený systém, ze kterého je daný uživatel přihlášen (ve výpisu bude chybět kolonka `FROM`).

Co obsahují soubory

V operačním systému Unix existují dva hlavní příkazy pro výpis obsahu souborů – `cat` a `more`. Již jsme o nich hovořili v kapitole 6.

`cat [-nA] [`*`soubor1 soubor2 ... souborN`*`]`

Příkaz `cat` nepatří mezi uživatelsky přátelské příkazy. Nečeká, až si přečtete obsah souboru a spíše se používá v souvislosti s rourami. Má však několik užitečných voleb. Například volba `n` zajistí, že všechny řádky vypisovaného souboru budou číslovány. Při zadání volby `A` se budou řídící znaky zobrazovat jako normální znaky a ne jako podivné sekvence paznaků. Opět připomínáme, že kompletní seznam voleb akceptovatelných příkazem `cat` naleznete v manuálových stránkách. Pokud se v příkazovém řádku neuvede ani jeden parametr, použije příkaz `cat` standardní vstup.

`more [-l] [+`*`linenumber`*`] [`*`file1 file2 ... fileN`*`]`

Příkaz `more` je mnohem užitečnější a budete jej často používat zejména k zobrazování souborů ve formátu ASCII. Jedinou zajímavou volbou je `l`, po jejímž uvedení příkazu `more` sdělíte, že nechcete interpretovat znak Ctrl-L jako znak „nová stránka“. `more` zahájí zobrazení od specifikovaného čísla řádku (*`linenumber`*).

Protože je more interaktivním příkazem, uvádíme v následujícím seznam příkazů, kterými lze jeho činnost řídit.

Mezerník	Zobrazí se následující stránka.
`d`	Text se posune o 11 řádků, což je přibližně polovina stránky.
`/`	Vyhledání regulárního výrazu. Zatímco regulární výraz může být opravdu komplikovaný, můžete zadat textový řetězec, který se má vyhledat. Zadáte-li například `/toad` Enter, vyhledá se v celém textu řetězec „`toad`“. Jestliže uvedete jen `/` Enter, pak se vyhledá další výskyt posledně zadaného řetězce.
`n`	Také tento příkaz je určen k vyhledání posledně zadaného regulárního výrazu.
`: n`	Pokud jste zadali prostřednictvím parametrů více než jeden soubor, zahájí se prohlížení následujícího souboru.
`: p`	Prohlížení se nastaví na předcházející soubor.
`q`	Program `more` se ukončí.

`head [-lines] [file1 file2 ... fileN]`

Příkaz `head` zobrazí prvních deset řádků každého z uvedených souborů nebo prvních deset řádků ze standardního vstupu, pokud žádný soubor není jako parametr v příkazovém řádku uveden. Jakákoliv numerická hodnota uvedená jako volba změní implicitní nastavení deset. Například `head -15 frog` zobrazí prvních patnáct řádků souboru `frog`.

`tail [-lines] [file1 file2 ... fileN]`

Podobně jako příkaz `head` zobrazuje příkaz `tail` jen část souboru. Normálně zobrazuje posledních deset řádků souboru nebo posledních deset řádků ze standardního vstupu. Také akceptuje volbu pro nastavení počtu zobrazených řádků.

`file [file1 file2 ... fileN]`

Příkaz `file` se pokouší identifikovat formát souborů uvedených v seznamu v příkazovém řádku. Protože ne všechny soubory mají příponu charakterizující formát souboru nebo neobsahují posloupnosti znaků, podle kterých by se dal formát identifikovat, pokouší se příkaz `file` provádět některé základní testy a tak odhadnout, co soubor obsahuje.

Buďte opatrní, často se může stát, že je formát souboru identifikován chybně.

Informační příkazy

Následující oddíl je věnovaný příkazům, které mění soubor, provádějí jisté operace se souborem nebo zobrazují některé statistické informace o souboru.

`grep [-nvwx] [-number] expression [file1 file2 ... fileN]`

Jeden z nejužitečnějších příkazů operačního systému Unix je příkaz `grep` (generalized regular expression parser). Jeho jméno se zdá být příliš nadsazené na to, že jde o program, jenž umí pouze vyhledávat regulární výrazy v textu. Následující příklad demonstruje nejjednodušší způsob používání příkazu grep:

```
/home/larry# cat animals
Animals are very interesting creatures. One of my favorite animals is
the tiger, a fearsome beast with large teeth.
I also like the lion---it's really neat!
/home/larry# grep iger animals
the tiger, a fearsome beast with large teeth.
/home/larry#
```

Z uvedeného příkladu vyplývá jedna nevýhoda. I když příkaz vypsal všechny řádky obsahující hledané slovo, neuvedl, kde se dané slovo v souboru vyskytuje (neuvedl číslo řádku).

Někdy to může stačit, v závislosti na tom, co potřebujete udělat. Když například hledáte chyby ve výstupu z programu, stačí zadat příkaz `a.out | grep error`, kde `a.out` je jméno vašeho programu.

Jestliže však chcete přesně vědět, kde se hledaný regulární výraz v souboru nachází, zadejte volbu `n`. Pak bude příkaz `grep` zobrazovat i pořadová čísla řádků. Volbu `v` použijte tehdy, když budete chtít zobrazit seznam všech řádků neobsahujících daný regulární výraz.

K dalším vlastnostem příkazu `grep` patří to, že implicitně vyhledává neúplná slova. V předcházejícím příkladu jsme zadali neúplné slovo `iger` a `grep` našel slovo `tiger`. Jestliže má program `grep` pracovat pouze s celými slovy, zadejte volbu `w`. Po zadání volby `x` bude `grep` pracovat s celými řádky.

Pokud neuvedete v příkazu `grep` žádné parametry, bude prohledávat standardní vstup.

`wc [-clw] [soubor1 soubor2 ... souborN]`

`wc` je zkratka pro „word count" (počet slov). Tento příkaz spočítá slova, znaky a řádky v souborech uvedených v seznamu. Pokud se v příkazovém řádku neuvedou jako parametry žádné soubory, pracuje příkaz `wc` se standardním vstupem.

Pro příkaz `wc` existují tři volby: `c` (character – znak), `l` (line – řádek) a `w` (word – slovo). Pomocí volby se specifikuje, co má program `wc` počítat. Pokud například zadáte příkaz `wc -cw`, pak spočítá znaky a slova, ale nikoliv řádky. Příkaz zadaný bez voleb spočítá vše – znaky, slova i řádky.

Jedna z aplikací příkazu `wc` spočívá v tom, že lze s jeho pomocí určit počet souborů v adresáři: `ls | wc -w`. Pokud chcete například vědět, kolik souborů končí s příponou `.c`, pak zadejte příkaz `ls *.c | wc -w`.

`spell [soubor1 soubor2 ... souborN]`

`spell` je velmi jednoduchý program operačního systému Unix pro kontrolu pravopisu textových souborů. Zpravidla kontroluje pravopis americké angličtiny.[3] Program `spell` je tedy filtr, který prochází soubor ve formátu ASCII a na výstupu vypisuje slova, která považuje za pravopisně nesprávná. `spell` pracuje se soubory uvedenými v příkazovém řádku jako parametry, jinak zpracovává standardní vstup.

Pravděpodobně máte k dispozici i dokonalejší program pro kontrolu pravopisu `ispell`. Program `ispell` navíc nabízí možnosti opravy chybně napsaného slova. Pokud se v příkazovém řádku specifikuje soubor, pak se program `ispell` ovládá pomocí nabídky, jinak pracuje jako klasický filtr.

Chcete-li se dozvědět o programu ispell více, podívejte se do manuálových stránek.

`cmp soubor1 [soubor2]`

Příkaz `cmp` porovnává obsahy dvou souborů. První musí být uveden jako parametr v příkazovém řádku, druhý je buď specifikován jako parametr, nebo se čte ze standardního vstupu. Příkaz `cmp` je velmi jednoduchý a jeho úkolem je ukázat, kde se dva soubory liší.

`diff soubor1 soubor2`

Jedním z nejkomplikovanějších standardních programů operačního systému Unix je příkaz `diff`. GNU verze programu `diff` má více než dvacet voleb! Jedná se o zdokonalenou verzi programu `cmp`, která ukazuje nejen kde se soubory liší, ale také v čem se liší.

Nebudeme probírat kompletní možnosti programu `diff`, protože to překračuje rámec této knihy. Místo toho se zaměříme na základní operace. Krátce řečeno, program `diff` akceptuje jako parametry dva soubory a zobrazí rozdíly mezi nimi na principu „řádek po řádku". Uveďme si příklad:

```
/home/larry# cat frog
Animals are very interesting creatures. One of my favorite
  animals is
the tiger, a fearsome beast with large teeth.
I also like the lion---it's really neat!
```

3 I když existuje několik verzí tohoto programu pro evropské jazyky, distribuce operačního systému Linux většinou obsahuje kontrolu pravopisu americké angličtiny.

```
/home/larry# cp frog toad
/home/larry# diff frog toad
/home/larry# cat dog
Animals are very nteresting creatures. One of my favorite
  animals is
the tiger, a fearsome beast with large teeth.
I also like the lion---it's really neat!
/home/larry# diff frog dog
1c1,2
< Animals are very interesting creatures. One of my favorite
  animals is
---
> Animals are very nteresting creatures. One of my favorite
  animals is
>
3c4
< I also like the lion---it's really neat!
---
> I also like the lion---it's really neat!
/home/larry#
```

Jak vidíte v našem příkladu, program `diff` neprodukuje žádný výstup, pokud jsou soubory identické. Pokud se porovnávaly dva různé soubory, pak řádek `1c1,2` říká, že byl porovnán první řádek levého souboru (`frog`) s prvním a druhým řádkem pravého souboru (`dog`) a že zde byly nalezeny rozdíly. Pak byly porovnány řádky 3 (v souboru `frog`) a 4 (v souboru `dog`) a i zde byl nalezen rozdíl. Může se zdát podivné, že se nejdříve porovnávají řádky s různým pořadovým číslem. Je to však z toho důvodu, aby program pracoval efektivněji.

`gzip [-v#] [`*`soubor1 soubor2 ... souborN`*`]`
`gunzip [-v] [`*`soubor1 souborX (xÎ{1,2,...,N})`*`]`
`zcat [`*`soubor1 soubor2 ... souborN`*`]`

Tyto tři programy se používají ke kompresi a dekompresi dat.

Příkaz `gzip` (GNU zip) je program, který čte původní soubory a produkuje soubory menší. Přitom soubory specifikované v příkazovém řádku ruší a nahrazuje je komprimovanými soubory. Komprimované soubory mají stejná jména, ale navíc mají příponu „`gz`“.

`tr` *`řetězec1 řetězec2`*

Příkaz `tr` (translate characters) pracuje pouze se standardním vstupem – neakceptuje žádné soubory jako parametry. Jeho dvěma parametry jsou dva řetězce. Příkaz nahradí všechny výskyty znaků řetězce *`řetězec1`* na vstupu odpovídajícím znakem z řetězce *`řetězec2`*. Kromě relativně jednoduchého tvaru tohoto příkazu, například `tr frog toad`, může příkaz `tr` akceptovat poměrně složité příkazy. Uvedeme si příklad, kdy se pomocí příkazu `tr` převádějí malá písmena na velká:

```
/home/larry# tr [:lower:] [:upper:]
this is WEIRD sentence.
THIS IS WEIRD SENTENCE.
/home/larry#
```

Program `tr` je dosti složitý a často se využívá v malých programech pro příkazové procesory.

KAPITOLA 8

Editace souborů v editoru Emacs

Co je to Emacs?

Abyste mohli dělat něco rozumného s počítačem, potřebujete mít možnost ukládat texty do souborů a měnit texty, které jsou v souborech uloženy. Takové úkony vám umožní realizovat textový editor. Emacs je jeden z nejpopulárnějších editorů na světě – zejména proto, že i začátečníkům umožňuje bez problémů pracovat s textovými soubory. Klasický editor dodávaný s operačním systémem Unix, `vi`, probereme v dodatku A.

Než se začnete učit pracovat s editorem Emacs, musíte si najít nějaký soubor obsahující obyčejný text a zkopírovat si jej do domovského adresáře.[1] Na začátek by bylo riskantní editovat originální soubor, mohl by obsahovat důležité informace. Editor Emacs se spouští jednoduše:

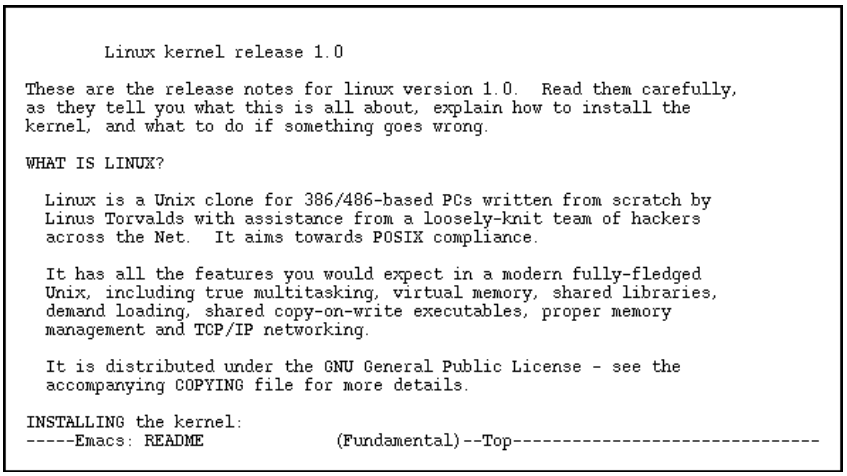
```
/home/larry# emacs README
```

Pokud jste se rozhodli zkopírovat soubor `/etc/rc`, `/etc/inittab` nebo jiný, pak samozřejmě jako parametr uvedete jméno tohoto souboru – například `emacs rc`.

Spuštění editoru Emacs může mít různý efekt. Záleží to na tom, v jakém prostředí jej spouštíte. V textovém terminálu zobrazujícím pouze textové znaky vytvoří nastartovaný editor Emacs okno přes celou obrazovku. Pokud jej spustíte v systému X Window, vytvoří si editor své vlastní okno. Budeme předpokládat, že jste spustili editor Emacs v textovém terminálu. Vše, co zde budeme uvádět, bude platit i pro editor Emacs spuštěný v okně systému X Window. V systému X Window nezapomínejte přesunout kurzor myši na okno obsahující editor, jinak nebudete moci nic psát.

Vaše obrazovka (nebo okno v systému X Window) by měla vypadat přibližně tak, jak je zobrazena na obrázku 8.1. Většina obrazovky obsahuje text vašeho souboru. Obzvláště důležité jsou poslední dva řádky, zejména chcete-li se naučit pracovat s tímto editorem. Řádek obsahující dlouhé posloupnosti pomlček je řádek specifikující mód editoru (modeline).

1 Například `cp /usr/src/linux/README ./README`



Obrázek 8.1 – Editor Emacs byl právě nastartován příkazem `emacs README`

Na uvedeném obrázku vidíte v předposledním řádku slovo „Top“. Může zde být také uvedeno slovo „All“ a mohou se zde také vyskytovat malé rozdíly, což záleží na verzi editoru. Někteří uživatelé mají v tomto řádku zobrazen aktuální čas. Pod uvedeným řádkem se nachází informační a **příkazový řádek**, někdy označovaný jako „minibuffer“, jindy jako „echo area“. Informační řádek slouží editoru ke komunikaci s uživatelem, k vypisování zpráv a někdy zde jeho prostřednictvím budete zadávat příkazy. Budou se zde objevovat takové pokyny, jako: „`For information about GNU project and its goals, type C-h C-p.`“ Zatím takové pokyny ignorujte, budeme se jimi zabývat později.

Než opravdu změníte text v nějakém souboru, musíte se naučit „pohybovat“ textem a pohybovat kurzorem. Kurzor by měl být umístěn na začátku souboru v levém horním rohu obrazovky. Chcete-li kurzor posunout o jeden znak dopředu, stiskněte kombinaci kláves C-F (stiskněte klávesu Ctrl, držte ji stisknutou a stiskněte klávesu F). Kurzor se posune o jeden znak dopředu. Pokud budete obě klávesy stále držet stisknuty, bude se pohyb kurzoru opakovat. Všimněte si, že když kurzor dospěje na konec řádku, automaticky se přesune na začátek následujícího řádku. Opačného směru pohybu kurzoru dosáhnete pomocí kláves C-B. Pokud chcete posunout kurzor o jeden řádek dolů, použijte klávesy C-N, pokud jej chcete posunout o jeden řádek nahoru, stiskněte klávesy C-P[2].

Používání klávesy Ctrl zajišťuje nejrychlejší způsob, jak při editování textu pohybovat kurzorem. Editor Emacs používá takové kombinace kláves, které při psaní udržují vaše ruce nad klávesnicí. Jestliže jste však zvyklí používat kurzorové klávesy, budou také fungovat.

Pokud pracujete v systému X Window, můžete k nastavení pozice textového kurzoru použít kurzor myši. Stačí kurzor myši nastavit na požadovanou pozici v textu a klepnout levým tlačítkem. Tento způsob je však velmi pomalý (jednou rukou musíte opustit klávesnici, uchopit myš, nastavit kurzor a pak se na klávesnici zase vrátit) a nedoporučujeme si na něj zvykat. Většina uživatelů, kteří pracují s editorem Emacs dlouhodobě, používá k pohybu kurzoru výhradně klávesnici.

[2] Zajisté jste si již všimli, že příkazy editoru Emacs jsou většinou realizovány pomocí kombinace dvou kláves.

Pomocí kláves C-P a C-B nastavte kurzor opět na začátek souboru, tedy horní levý roh obrazovky. Nyní podržte klávesy C-B stisknuty poněkud déle. Uslyšíte pípnutí a v informačním řádku uvidíte zprávu „`Beginning of buffer`“.

V tomto okamžiku se asi podivíte. Co je to buffer?

Když editor Emacs zpracovává soubor, nepracuje ve skutečnosti se souborem přímo. Zkopíruje si jeho obsah do speciální pracovní oblasti, která se nazývá **buffer** – zde můžete obsah souboru modifikovat. Až ukončíte editaci souboru, dáte příkaz, aby editor buffer uložil. Do té doby zůstane obsah původního souboru nezměněn a změny se provádějí pouze uvnitř pracovní oblasti editoru.

Nyní jsme tedy připraveni obsah bufferu modifikovat. Vše, co jsme dělali doposud, bylo „nedestruktivní“, což znamená, že jsme obsah bufferu neměnili. Předpokládejme, že chcete do editovaného souboru zapsat znak „X“. Jakmile stisknete klávesu X, změní se předposlední řádek obrazovky. Editor Emacs registruje změny v editovaném textu a jakmile nějaké nastanou, informuje o tom uživatele pomocí dvou hvězdiček před slovem Emacs. Pak bude uvedený řádek vypadat asi takto:

```
--**-Emacs: some_file.txt     (Fundamental)--Top--------------------
```

Hvězdičky zůstanou zobrazeny do té doby, než obsah bufferu uložíte. Obsah bufferu můžete v průběhu editace ukládat průběžně vícekrát – stačí stisknout klávesy C-X C-S (stiskněte klávesu Ctrl, držte ji stisknutou a pak stiskněte klávesu „X“ a „S“). Průběžné ukládání obsahu bufferu má význam především při vytváření nebo modifikaci velkých souborů.

Nyní si uvedeme seznam několika málo příkazů používaných v editoru Emacs spolu s těmi, které jsme již popsali. Je na vás, abyste si jejich používání procvičili. Než pokročíme dále, měli byste s těmito příkazy být důkladně seznámeni.

C-F	Posunutí kurzoru o jeden znak doprava.
C-B	Posunutí kurzoru o jeden znak doleva.
C-N	Posunutí kurzoru o jeden řádek dolů.
C-P	Posunutí kurzoru o jeden řádek nahoru.
C-A	Umístění kurzoru na začátek řádku.
C-E	Umístění kurzoru na konec řádku.
C-V	Zobrazení následující stránky.
C-L	Zobrazení stránky s aktuálním řádkem uprostřed.
C-D	Zrušení znaku, na kterém je umístěn kurzor.
C-K	Zrušení textu od pozice kurzoru do konce řádku.
C-X C-S	Uložení obsahu bufferu do odpovídajícího souboru.
Backspace	Zrušení znaku vlevo od kurzoru.

Používání editoru pod systémem X Window

Pokud chcete rychle editovat soubory pod systémem X Window, máte poněkud ulehčenou situaci. Editovaný text je zobrazen pod nabídkou funkcí editoru, proto je práce s editorem intuitivnější.

V textovém módu uvedená nabídka není dostupná.

Když poprvé spustíte editor Emacs pod systémem X Window, obsahuje nabídka čtyři hlavní položky: `Buffers`, `File`, `Edit` a `Help`. Potřebujete-li si některou z položek hlavní nabídky zpřístupnit, nastavte kurzor myši na tuto položku, stiskněte levé tlačítko myši (držte je stále stisknuté). Pak přesuňte kurzor na požadovanou funkci a tlačítko myši uvolněte. Jestliže žádnou funkci vybrat nechcete, přesuňte kurzor myši mimo nabídku a opět uvolněte tlačítko myši.

Nabídka `Buffers` obsahuje seznam různých souborů, které v této relaci s editorem Emacs editujete. Nabídka `File` obsahuje příkazy pro zavádění a ukládání souborů. Funkce této nabídky podrobněji popíšeme později. Nabídka `Edit` obsahuje některé nejdůležitější příkazy pro editování textového souboru a prostřednictvím nabídky `Help` si můžete interaktivně zobrazovat vybrané části dokumentace k editoru.

Jistě jste si všimli, že u každé funkce v nabídce je uvedena ekvivalentní kombinace kláves pro realizaci této funkce. Na jedné straně můžete funkce vyvolávat pomocí myši a menu, na druhé straně můžete k tomuto účelu používat ekvivalentní kombinace kláves, což je samozřejmě rychlejší.

Editování více souborů současně

V daném okamžiku je editor Emacs schopen pracovat s více soubory. Ve skutečnosti platí, že počet souborů, které mohou být současně uloženy v bufferech editoru, je omezen pouze velikostí paměti počítače. Po zadání příkazu Ⓒ-Ⓧ Ⓒ-Ⓕ se do bufferu editoru Emacs zavádí nový soubor. Když tento příkaz zadáte, v příkazovém řádku se objeví výzva k zadání jména souboru:

```
Find file: ~/
```

Syntaxe používaná k zadání jména souboru je stejná, jako v příkazovém řádku příkazového procesoru; lomítka se používají k oddělení adresářů a podadresářů, znak ~ je interpretován jako váš domovský adresář. I zde funguje možnost automatického **doplňování jména souboru** – zadáte-li dostatek znaků k tomu, aby mohlo být jméno souboru jednoznačně identifikováno, pak stačí stisknout klávesu Tab a zbytek jména souboru se doplní automaticky (nebo se zobrazí seznam jmen všech souborů, která zadané specifikaci vyhovují). Podobnou funkci jako klávesa Tab má klávesa mezerník. Necháme na vás, abyste zjistili, v čem se funkce vyvolané těmito klávesami liší. Máte-li zadáno jméno souboru, který chcete editovat, stiskněte klávesu Enter; editor zavede obsah souboru do své pracovní oblasti a zobrazí jej na obrazovce. V terminologii editoru Emacs se tento proces nazývá vyhledání souboru. Najděte si nějaký další soubor a popsaným způsobem jej zaveďte do editoru. Nyní máte v editoru nový buffer. Budeme předpokládat, že původní má název `some_file.txt` a nový má název `another_file.txt`. Zdá se, že se váš první buffer ztratil. Pravděpodobně se divíte, kam se poděl.

Neobávejte se, stále zůstává uvnitř editoru a k jeho opětovnému zobrazení stačí stisknout klávesovou kombinaci Ⓒ-Ⓧ Ⓑ. Pak se vás editor v příkazovém řádku zeptá, do kterého bufferu se má přepnout. Nabídne implicitní buffer, t.j. buffer, jehož obsah se zobrazí po stisknutí klávesy Enter (nemusíte jméno bufferu uvádět). Implicitní buffer je ten, který jste „opustili“ jako poslední. Jestliže tedy pracujete se dvěma soubory a často mezi nimi přepínáte, používejte klávesovou kombinaci Ⓒ-Ⓧ Ⓑ a nemusíte při každém přepnutí zadávat jméno souboru. Samozřejmě můžete jméno uvádět, ale bude vás to zdržovat.

I při přepínání mezi buffery lze použít funkci k doplňování jména – stejně jako v případě vyhledávání souboru. Po stisknutí klávesy Tab doplní editor automaticky jméno bufferu, je-li schopen jej z doposud zadaných znaků jednoznačně identifikovat. Kdykoliv jste v příkazovém řádku vyzváni k zadání jména, vyzkoušejte si, zda je editor schopen jméno automaticky doplnit. Automatické doplňování vám ušetří spoustu času a editor je schopen je realizovat pokaždé, když má vybrat nějakou položku z nějakého předdefinovaného seznamu.

Vše, co jste se naučili o příkazech pro editování v prvním bufferu platí i v druhém. Pokročme dále – v novém bufferu změňte nějaký text, ale neukládejte jej (pomocí kláves C-X C-S). Nyní předpokládejme, že nechcete provedené změny uložit a že chcete buffer zrušit. K tomu slouží příkaz C-X K. Nejdříve se vás editor zeptá, který buffer chcete zrušit. Stisknete-li klávesu Enter, zruší se implicitní buffer (což je asi nejčastější případ). Editor se vás znovu zeptá, zda chcete buffer opravdu zrušit, a když zadáte „yes“ a stisknete Enter, pak jej konečně zruší.

Nyní byste se měli důkladně procvičit v zavádění souborů, jejich modifikaci, ukládání, přepínání mezi buffery a rušení bufferů. Ujistěte se, že needitujete nějaké důležité soubory, které by mohly poškodit funkci vašeho systému.[3] Vytvořte si alespoň pět bufferů a vyzkoušejte si přepínání mezi nimi.

Ukončení práce s editorem

Když ukončíte práci s editorem Emacs, ujistěte se, že všechny buffery, které mají být uloženy jsou skutečně uloženy. Pak můžete editor ukončit pomocí kláves C-X C-C. Někdy se vás editor po zadání kláves C-X C-C v příkazovém řádku na něco zeptá. Nebuďte znepokojeni a odpovězte normálním způsobem. Jestliže se domníváte, že se budete do editoru chtít později vrátit, nepoužívejte klávesy C-X C-C, ale klávesu C-Z. Pak bude aktivita editoru pouze pozastavena. K opětovné aktivaci editoru můžete použít známý příkaz `fg`. Pozastavení editoru je mnohem efektivnější, než jej stále dokola ukončovat a znovu startovat, zejména když opakovaně editujete tytéž soubory.

V systému X Window má funkce C-Z jiný efekt – provede transformaci okna s editorem do ikony. Podrobně jsme tento mechanismus popsali v kapitole 5. Existují tedy dvě možnosti, jak transformovat okno obsahující editor do ikony – buď normálním způsobem pomocí správce oken, nebo pomocí klávesy C-Z. V systému X Window však k opětovné aktivaci editoru nelze použít příkaz fg – musíte použít správce oken.

Klíče Meta

Doposud jsme při práci s editorem Emacs používali kombinace kláves s klávesou Ctrl. Existují však další možné kombinace, a to s klávesou Meta. Těmto kombinacím se v terminologii editoru Emacs říká meta-klíče. Bohužel ne všechny klávesnice mají klávesu Meta umístěnu ve stejné poloze a některé ji nemají vůbec. V případě klávesnic u počítačů IBM PC je klávesa Meta totožná s klávesou Alt.

Klávesu Meta si můžete otestovat. Stiskněte tu klávesu, o které si myslíte, že by mohla být klávesou Meta, a zároveň stiskněte klávesu X. Pokud se v příkazovém řádku objeví malá nápověda (zpravidla M-X), pak jste kýženou klávesu nalezli. Stiskněte klávesu C-G a znovu se vrátíte do bufferu editoru Emacs.

3 Pokud nejste přihlášeni jako uživatel root, neměli byste být schopni vašemu systému ublížit, ale stejně buďte opatrní.

Jestliže se vám v příkazovém řádku nic neobjeví, stále existuje řešení. Místo klávesy Meta můžete použít klávesu Escape. Avšak v tomto případě ji nedržte stisknutou – stiskněte ji, uvolněte a zadejte další klávesu reprezentující danou funkci. Vyzkoušejte si naši oblíbenou kombinaci, tedy Escape a pak „X“. Nyní se vám v příkazovém řádku nápověda určitě objeví. Opět stiskněte klávesu C-G. Klávesa C-G je v editoru Emacs určena ke zrušení čehokoliv, co nehodláte realizovat. Editor zpravidla vyšle zvukový signál, aby vás upozornil, že danou funkci rušíte.[4]

Označení M-X je analogické označení C-X (kde x je klávesa reprezentující nějakou funkci). Pokud jste nalezli skutečný klíč Meta, pak jej používejte. Jinak budete muset používat popsanou sekvenci s klávesou Escape. Kdekoliv v následujícím textu uvidíte M-X, pak to znamená, že máte použít meta-klíč.

Práce s bloky textu

Editor Emacs, tak jako prakticky každý editor, umožňuje pracovat s bloky textu. Abyste tyto funkce mohli využívat, musíte mít prostředek pro definování **začátku bloku** a **konce bloku**. V editoru Emacs se definice bloku realizuje pomocí nastavení dvou pozic v bufferu, které se označují jako mark (značka) a point. Chcete-li v bufferu nastavit začátek bloku, nastavte kurzor na požadovanou pozici a stiskněte klávesu C-SPC (SPC je zkratka pro mezerník). V příkazovém řádku se objeví zpráva „Mark set“.[5] Nyní je začátek bloku nastaven. Editor nepoužívá žádné zvýrazňovací prostředky k tomu, aby byl začátek bloku viditelný.

A co konec bloku? Konec bloku je definován aktuální pozicí kurzoru. Proto se pro něj používá v terminologii editoru Emacs termín point. Point zrovna tak například znamená místo, od kterého se má vkládat text při kopírování nebo vkládání bloků. Nastavením začátku bloku a přesunutím kurzoru na kteroukoliv jinou pozici v textu je definován začátek a konec bloku. Tento blok se nazývá **region**. Region je tedy vždy část textu mezi značkou a aktuální pozicí kurzoru.

Pouhá definice regionu nezpřístupňuje blok ke kopírování. Nejdříve musíte blok „**zkopírovat**“ (copy), abyste jej pak mohli „**nalepit**“ (paste) někam jinam. Má-li se blok zkopírovat, stiskněte klávesu M-W. Nyní je blok uložen ve speciálním bufferu editoru Emacs. Jestliže nyní chcete blok nalepit někam jinam, nastavte si kurzor na požadovanou pozici a stiskněte C-Y.

Pokud chcete blok textu přesunout (a ne zkopírovat), pak místo příkazu M-W použijte příkaz C-W. Tímto způsobem se blok po přenesení do speciálního bufferu vymaže. Když zjistíte, že jste blok nechtěli vymazat, stiskněte klávesu C-Y a blok se obnoví. Místo, do kterého editor Emacs ukládá části textu, se nazývá **kill-ring**. U jiných editorů se tato oblast nazývá „clipboard“ nebo „paste buffer“.

Existuje ještě jeden způsob, jak vystřihovat a nalepovat text: kdykoliv stisknete klávesu C-K, dojde ke zrušení textu od pozice kurzoru do konce řádku a přitom se zrušený text také ukládá do oblasti kill-ring. Pokud takto zrušíte více než jeden řádek, uloží se všechny řádky do oblasti kill-ring a vy máte možnost je později nalepit najednou.

Někdy je tento způsob přesouvání a kopírování textu rychlejší, než používat systém s označováním začátku a konce bloku. Záleží jen na vás, kterému způsobu dáte přednost.

---

4 Příležitostně se může stát, že jedno stisknutí klávesy C-G nepřesvědčí editor Emacs, že chcete opravdu přerušit činnost, kterou právě děláte. Stačí však trvat na svém a pak se editor vrátí do předcházejícího módu.

5 Na některých počítačích příkaz C-SPC nebude fungovat. Místo toho použijte C-@.

Vyhledávání a náhrada řetězců

V editoru Emacs máte několik možností, jak vyhledávat text. Některé způsoby jsou komplikované a nemá smysl je zde popisovat. Nejjednodušší a nejčastěji používaná metoda je tzv. inkrementální vyhledávání označované jako „isearch“ (incremental search). Předpokládejme, že potřebujete vyhledat řetězec „gadfly“ v následujícím textu:

```
I was growing afraid that we would run out of gasoline, when my passenger ex-
claimed
''Gadzooks! There's gadfly in here!''.
```

Nastavte pozici kurzoru na začátek textu nebo na místo, o kterém víte, že předchází hledanému řetězci, a zadejte příkaz C-S. Tím nastavíte editor Emacs do módu vyhledávání. Nyní začněte zadávat řetězec, který chcete vyhledat. Jakmile však napíšete první znak, tedy „g“ „přeskočí“ kurzor na první výskyt písmene „g“ v textu. Jestliže máte v editoru text uvedený v našem příkladu, pak kurzor skočí na začátek slova „growing“. Nyní napište písmeno „a“ (druhé písmeno ve slově „gadfly”) a editor přesune kurzor na slovo „gasoline“, protože vyhledal první výskyt dvojice znaků „ga“. Když zadáte další písmeno „d“, skočí kurzor na slovo „gadzooks“ a nakonec po zadání písmene „f“ skočí na hledané slovo „gadfly“. Přitom jste nemuseli zadat kompletní hledaný řetězec.

Při postupném vyhledávání funguje editor Emacs tak, že po každém zadání dalšího znaku hledaného řetězce vyhledá první výskyt toho slova, jehož začátek je shodný s doposud zapsanými znaky. Jakmile zadáte tolik znaků, aby vyhledávání bylo jednoznačné, můžete funkci ukončit stisknutím klávesy Enter. Jestliže se domníváte, že hledaný řetězec je nad aktuální pozicí kurzoru, pak zadejte příkaz C-R, čímž inicializujete zpětné vyhledávání.

Pokud naleznete první výskyt zadaného řetězce, ale zajímá vás jeho další výskyt, pak znovu zadejte příkaz C-S. Editor Emacs vyhledá následující výskyt zadaného řetězce a tak můžete pokračovat dále. Když editor výskyt hledaného řetězce nenalezne, vypíše zprávu, že řetězec nenašel a začne znovu vyhledávat od začátku bufferu. Podobná pravidla platí pro příkaz C-R.

Nyní si popsané funkce vyzkoušejte. Najděte si soubor s anglickým textem a vyhledejte v něm výskyt slova „the“. Pak pomocí opakované funkce C-S najděte další výskyty. Všimněte si, že editor přitom vyhledá i jiné řetězce, například „them“, protože vyhovují zadané specifikaci. Jestliže chcete vyhledat pouze řetězec „the“, pak na konec hledaného řetězce budete muset přidat mezeru. Při editování hledaného řetězce můžete používat standardní editační klávesy Backspace a Delete. Chcete-li vyhledávání ukončit, vždy použijte klávesu Enter.

Editor Emacs také umožňuje nahradit výskyt jednoho řetězce jiným. Tento proces se v terminologii editoru Emacs označuje jako **query-replace**. Proces zahájíte tak, že stisknete M-X a napíšete `query-replace` a stisknete Enter.

I v případě zadávání příkazů funguje v editoru Emacs doplňování, a proto stačí, když například napíšete „query-re“ a stisknete klávesu Tab. Předpokládejme, že chcete řetězec „gadfly“ nahradit řetězcem „housefly“. V příkazovém řádku „`Query replace:`“ zadejte „gadfly“ a stiskněte Enter. Jak budete opět vyzváni k zadání řetězce, kterým se má původní řetězec nahradit – zadejte „housefly“. Editor Emacs bude nyní procházet text, zastavovat kurzor na každém výskytu řetězce „gadfly“ a bude se vás ptát, zda má nalezený řetězec nahradit. Pokud ano, stiskněte klávesu Y, pokud ne, stiskněte klávesu N. Jestliže je pro vás předcházející výklad příliš komplikovaný, popsané funkce si vyzkoušejte. Bude to mít pro vás větší význam, než kdybyste předcházející odstavce četli desetkrát.

Vnitřní funkce editoru Emacs

Všechny funkce editoru Emacs vyvolané stisknutím nějaké kombinace kláves mají nějaký název, kterému editor „rozumí". Například klávesa C-P znamená pro editor Emacs provedení vnitřní funkce `previous-line`. Všechny vnitřní funkce mohou být volány prostřednictvím svého jména, a to po stisknutí kláves Alt+X. Když například zapomenete, jaký klíč máte použít k nastavení kurzoru na předcházející řádek, stačí zadat: M-X `previous-line` a Enter. Vyzkoušejte si to a přesvědčte se, že C-P a M-X `previous-line` realizují tutéž funkci.

Autor editoru Emacs postupoval tak, že nejdříve definoval celou množinu vnitřních funkcí editoru a pak teprve navrhl klávesové zkratky pro realizaci nejčastěji používaných funkcí. Někdy je jednodušší použít explicitní volání funkce prostřednictvím M-X, než si pamatovat klávesovou zkratku svázanou s touto funkcí. Například funkce `query-replace` je u některých verzí editoru Emacs svázána s klávesou M-%. Kdo si ale má pamatovat takovou divnou kombinaci? Pokud budete náhradu řetězců provádět extrémně často, pak má samozřejmě smysl si klávesovou zkratku pamatovat.

Většina kláves, které stisknete, jsou písmena, číslice, případně další znaky, které se mají vkládat do textového bufferu. Každá z těchto kláves je **svázána** s funkcí, jež má jméno `self-insert-command`. Tato funkce nedělá nic jiného, než že dané písmeno nebo číslici vloží do bufferu. Kombinace kláves, například s klávesou Ctrl, jsou obecně svázány s jinými funkcemi – pohyb kurzoru a podobně. Například kombinace C-V je svázána s funkcí `scroll-up`, což znamená, že se editovaný text posune o jednu obrazovku dolů.

Jak ale postupovat, když do textu chcete vložit řídící znak? Konec konců, řídící znaky jsou také znaky ASCII, i když zřídkakdy používané. Může se stát, že budete chtít do textu takový znak vložit. Proto v editoru Emacs existuje prostředek, který zabrání editoru interpretovat kombinaci klávesy s klávesou Ctrl jako příkaz. Klávesa C-Q[6] je svázána se speciální funkcí, která má název `quoted-insert`. Jediné, co funkce `quoted-insert` dělá, je, že přečte následující klávesu a vloží ji do textového bufferu bez interpretace. Pokud chcete do textu vložit samotný znak C-Q, pak zadejte C-Q dvakrát.

V editoru Emacs existují některé funkce, jež nejsou svázány s žádnou kombinací kláves. Když například píšete dlouhý text, pak asi nebudete chtít ukončovat každý řádek klávesou Enter. Po editoru Emacs můžete chtít, aby to dělal za vás (po editoru Emacs můžete chtít cokoliv). Příkaz, který takovou funkci realizuje, má název `auto-fill-mode`. Není však implicitně svázán s žádnou kombinací kláves. Jestliže chcete tento příkaz inicializovat, zadejte `M-X auto-fill-mode`. Jak jsme si již řekli, „M-X" je klíč umožňující vyvolat vnitřní funkci editoru jménem. Takto byste mohli vyvolat i funkci `next-line` (následující řádek) nebo `previous-line` (předcházející řádek), ale to by bylo velmi neefektivní, protože uvedené funkce jsou svázány s klávesami C-N a C-P.

Mimochodem, když se po vyvolání funkce `auto-fill-mode` podíváte na předposlední řádek, uvidíte zde na pravé straně slovo Fill. Dokud se zde toto slovo vyskytuje, bude editor Emacs ukončovat řádky za vás. Když funkci „`M-X auto-fill-mode`" vyvoláte znovu, automatické ukončování řádků přestane fungovat – funkce je vytvořena jako přepínač.

Může se vám zdát, že zadávání funkcí prostřednictvím jejich dlouhých jmen není příliš pohodlné. Naštěstí i v tomto případě v editoru Emacs funguje doplňování jmen (stejně jako doplňování jmen souborů). Proto platí, že zřídkakdy budete muset vypisovat celé jméno funkce, písmeno po písmenu. Pokud si nejste jisti, zda bude editor schopen doplnit jméno, stiskněte klávesu Tab. V horším případě se vám objeví znak Tab, v lepším případě editor provede doplnění.

---

6 Pro kombinaci kláves C-Q se používá označení „klávesa", protože představuje jediný znak z tabulky ASCII.

Nápověda v editoru Emacs

Editor Emacs má velmi rozsáhlou nápovědu. Tak rozsáhlou, že se o ní zmíníme jen velmi stručně. Nejvyšší úroveň nápovědy se vyvolá stisknutím kombinace kláves Ⓒ-Ⓗ a nějakého písmene. Například Ⓒ-Ⓗ Ⓚ vyvolá nápovědu vztahující se ke kombinacím kláves (budete vyzváni k zadání kombinace kláves a pak se objeví text s vysvětlením, jakou funkci kombinace kláves realizuje). Ⓒ-Ⓗ Ⓣ vyvolá výukový program editoru Emacs. K důležitým kombinacím kláves patří Ⓒ-Ⓗ Ⓒ-Ⓗ, kdy se vám objeví „nápověda o nápovědě“. Zde se dozvíte vše o systému nápovědy. Když znáte jméno funkce editoru Emacs (například `save-buffer`), ale nemůžete si vzpomenout na kombinaci kláves k jejímu vyvolání, použijte Ⓒ-Ⓗ Ⓦ („`where - is`”) a zadejte jméno funkce. Zrovna tak máte možnost zjistit podrobné informace o dané funkci – zadejte Ⓒ-Ⓗ Ⓕ a pak jméno funkce.

Mějte na paměti, že editor Emacs je sice schopen doplňovat jména funkcí, avšak nemůžete na tuto vlastnost příliš spoléhat, pokud k nějaké funkci (jejíž název přesně neznáte) potřebujete nápovědu. Jestliže si myslíte, že můžete odhadnout slovo, kterým funkce začíná, zkuste je napsat a stiskněte Tab. Uvidíte, zda editor byl schopen funkci identifikovat. Pokud ne, zkuste něco jiného. Totéž platí pro jména souborů. I když si nemůžete vzpomenout, jak se jmenuje soubor, který jste editovali před mnoha měsíci, můžete název odhadnou a zkusit, co na to editor odpoví. Doplňování jmen je tedy nejen užitečné v tom, že šetří čas, ale pomáhá vám nalézt to, co jste již zapomněli, nebo to, co si přesně nepamatujete.

Existuje několik dalších znaků, které můžete zadat po příkazu Ⓒ-Ⓗ a tak získat nápovědu různým způsobem. Nejčastěji však budete používat Ⓒ-Ⓗ Ⓚ, Ⓒ-Ⓗ Ⓦ a Ⓒ-Ⓗ Ⓕ. Až budete blíže seznámeni s editorem Emacs, vyzkoušejte si například Ⓒ-Ⓗ Ⓐ. Pak se vás editor zeptá na řetězec a mezi všemi jmény funkcí nalezne to, které daný řetězec obsahuje. (Písmeno „a“ je zkratkou pro „apropos“ nebo „about“.)

Jiný zdroj informací o editoru Emacs nabízí systém pro čtení dokumentace v hypertextovém formátu **Info**. Ten můžete inicializovat přímo z editoru Emacs tak, že stisknete kombinaci kláves Ⓒ-Ⓗ Ⓘ. Pak se vám objeví základní stránka systému Info, ve které najdete další pokyny, jak postupovat při vyhledávání informací.

Pracovní módy editoru Emacs

Každý buffer editoru Emacs je spjat s tzv. módem.[7] Módy byly zavedeny z toho důvodu, že například při psaní zprávy pro elektronickou poštu má uživatel jiné požadavky, než při psaní programu v jazyku C. Kdyby měl editor splňovat všechny požadavky najednou, bylo by jeho používání velmi komplikované.

Proto se autor editoru Emacs[8] rozhodl řešit tuto situaci pomocí módů. Chování editoru se mění podle toho, s jakým módem je konkrétní buffer spjat. Jednotlivé módy se od sebe liší vazbou mezi kombinacemi kláves a funkcemi, ale jsou zde i jiné rozdíly.

Nejzákladnějším módem je mód `fundamental`, který nemá žádné speciální funkce. Uvedeme zde, co o základním módu říká samotný editor Emacs:

```
Fundamental mode:
Major mode not specialized for anything in particular.
Other major modes are defined by comparison with this one.
```

7 Aby nebyla situace tak jednoduchá, existují zde hlavní módy (Major Modes) a vedlejší módy (Minor Modes). Zatím však pro nás nejsou podstatné.

8 Autor editoru Emacs je Richard Stallman.

Právě uvedenou informaci jsem získal takto: zadal jsem příkaz C-X B (což znamená vyvolání funkce `switch-to-buffer`) a zadal jsem „foo“ jako jméno bufferu, do kterého se chci přepnout. Protože buffer s takovým jménem nebyl doposud vytvořen, editor jej vytvořil a přepnul se do něj. Implicitně v něm nastavil základní mód (`fundamental-mode`). Všechny názvy módů mají tvar `<modename>-mode` a pomocí funkce „M-X“ lze pro každý buffer specifikovat mód explicitně právě pomocí jeho názvu. Abych získal více informací o základním módu, zadal jsem příkaz C-H M. Nakonec se zobrazila výše uvedená informace o základním módu.

Od základního módu je odvozen mírně užitečnější mód `text-mode` (textový mód), který má dva speciální příkazy: M-S pro funkci `center-paragraph` a M-S pro funkci `center-line`. Příkaz M-S znamená, že máte stisknout klávesu Shift, držet ji stisknutou, a pak stisknout klávesu Shift a „S“.

Nyní si můžete vyzkoušet vytvořit nový buffer a definovat v něm textový mód. Pak stiskněte kombinaci kláves C-H M. Objeví se vám informace o textovém módu. Možná nebudete všemu rozumět, ale jistě zde najdete užitečné informace.

V dalších oddílech si probereme některé z nejpoužívanějších módů. Budete-li s nimi pracovat, nezapomeňte na kombinaci kláves C-H M, kdykoliv budete chtít znát o aktuálním módu nějaké podrobnosti.

Programovací módy

Mód pro jazyk C

Jestliže použijete editor Emacs pro psaní programů v jazyku C, můžete po něm chtít, aby prováděl automatické odsazování. Soubory s příponou „.c“ nebo „.h“ zavádí editor Emacs automaticky v módu „c-mode“. To znamená, že jsou k dispozici některé speciální funkce vhodné pro psaní programů v jazyku C. Klávesa Tab je v módu `c-mode` svázána s funkcí `c-indent-command`. To znamená, že se po stisknutí klávesy Tab nevloží do textu znak Tab, ale dojde k automatickému odsazení řádku podle kontextu ve vytvářeném programu. Z toho vyplývá, že editor Emacs má jisté znalosti o syntaxi programů napsaných v jazyku C. V žádném případě však nepočítejte s tím, že vás bude editor upozorňovat na chyby v programu!

Navíc editor předpokládá, že jsou předcházející řádky odsazeny správně. Pokud v předcházejícím řádku chybí závorka, středník, složená závorka či cokoliv jiného, pak editor provede odsazení chybně. To je pro vás signál, že jste na něco zapomněli a můžete předcházející řádek opravit.

Uvedenou vlastnost editoru Emacs můžete využít ke kontrole oddělovačů ve zdrojovém programu. Nemusíte celý program číst od začátku a pracně hledat chybu, ale stačí zahájit odsazování řádků od začátku souboru pomocí klávesy Tab. Když dojde k nesprávnému odsazení, zkontrolujte předcházející řádek. Jinými slovy, v tomto směru za vás editor Emacs může udělat spoustu práce.

Mód pro jazyk Scheme

Tento mód pro vás bude užitečný jen tehdy, když používáte programovací jazyk Scheme. Tento jazyk není tak běžně používaným jazykem jako jazyk C nebo Pascal, ale v poslední době jeho popularita vzrůstá, a proto se mu budeme také věnovat. Většina toho, co platí pro jazyk Scheme, platí i pro jazyk Lisp.

Aby to nebylo tak jednoduché, v editoru Emacs jsou definovány dva módy pro programovací jazyk Scheme a každý uživatel se může rozhodnout, který mu bude lépe vyhovovat. Zaměříme se na popis módu s názvem `cmuscheme` a později, v oddíle o konfiguraci editoru Emacs, si vysvětlíme rozdíly mezi oběma módy. Nebuďte při čtení následujících řádků zneklidněni, když se váš edi-

tor Emacs bude chovat trochu jinak, než zde bude popisováno. V editoru lze konfigurovat prakticky vše a různé distribuce operačního systému Linux obsahují různě konfigurované editory Emacs.

V editoru Emacs můžete inicializovat interaktivní proces Scheme pomocí příkazu `M-X run-scheme`. Takto vytvoříte buffer nazvaný „`*scheme*`“, ve kterém se nachází obvyklý příkazový řádek jazyka Scheme. V tomto řádku můžete zadávat výrazy v jazyku Scheme ukončené klávesou Enter, jazyk Scheme je bude vyhodnocovat a bude zobrazovat příslušné odpovědi. Tímto způsobem můžete, máte-li interaktivně komunikovat s procesem Scheme, zadat definice všech svých funkcí a aplikací v příkazové řádku. Jiná situace nastane v případě, že máte zdrojový kód zapsaný v nějakém samostatném souboru. Pak by mohlo být jednodušší editovat tento soubor a definice odeslat prostřednictvím bufferu Scheme.

Jestliže do editoru Emacs zavedete soubor s příponou „`.ss`“ nebo „`.scm`“, pak se automaticky nastartuje mód **Scheme mode**. Pokud se z nějakých důvodů tento mód nenastartuje, můžete jej aktivovat prostřednictvím příkazu M-X `scheme-mode`. Tento mód ovšem není totožný s bufferem, ve kterém běží proces Scheme. V módu `scheme-mode` však máte k dispozici některé příkazy pro komunikaci s uvedeným bufferem.

Jestliže ve zdrojovém kódu jazyka Scheme máte vytvořenu definici nějaké funkce, pak ji stisknutím kláves C-C C-E můžete odeslat do bufferu, ve kterém běží proces Scheme. Pokud stisknete klávesy C-C M-E, pak se po odeslání definice funkce dostanete přímo do uvedeného bufferu a zde můžete zadávat interaktivní příkazy. Klávesy C-C C-L jsou určeny k zavedení souboru s kódem v jazyku Scheme (fungují pro buffer s procesem Scheme i pro buffer se zdrojovým kódem). Stejně jako u ostatních programovacích jazyků je klávesa Tab určena k odsazování řádků.

Pokud pracujete v bufferu, ve kterém běží proces Scheme, můžete používat klávesy M-P a M-N k zobrazení předcházejících nebo následujících příkazů (tzv. **input history**).

Předpokládejme, že ladíte nějakou funkci, řekněme '`rotate`', pro kterou jste použili argumenty, například:

```
> (rotate '(a b c d e))
```

pak můžete tento příkaz znovu zavést do příkazového řádku pomocí M-P. Nemusíte znovu v příkazovém řádku Scheme zadávat dlouhou sekvenci znaků a ušetříte tak spoustu času.

Editor Emacs má speciální módy jen pro několik málo programovacích jazyků. Patří mezi ně jazyk C, C++, Lisp a Scheme. (V současné době je dostupný mód snad pro každý programovací jazyk – např. Java, Python nebo JavaScript.)

Mód pro elektronickou poštu

Prostřednictvím editoru Emacs můžete také vytvářet a odesílat zprávy pro elektronickou poštu. K tomu je určen speciální buffer, tzv. „mail buffer“, který se inicializuje prostřednictvím klávesy C-X M. Nejdříve vyplníte položky „`To:`“ a „`Subject:`“ a pak se pomocí kombinace kláves C-N přenesete přes oddělovací čáru do pole, ve kterém můžete napsat zprávu. Oddělovací čáru nikdy needitujte ani nerušte, protože pak by editor Emacs nebyl schopen elektronickou zprávu odeslat. Oddělovací čáru používá k odlišení záhlaví od textu zprávy.

V poli pro text zprávy (pod oddělovací čarou) můžete uvést cokoliv. Po dokončení zprávy ji odešlete pomocí kláves C-C C-C. Editor Emacs obsah bufferu odešle.

Jak zvýšit efektivitu práce s editorem Emacs

Zkušení uživatelé editoru Emacs jsou až fanatičtí, pokud jde o zvyšování efektivity práce s editorem. Ve skutečnosti někdy stráví více času zvyšováním efektivity, než kolik pak ušetří. I když nechci, abyste se stali takovými fanatiky, existuje několik opatření, pomocí kterých se vám bude s editorem Emacs pracovat snadněji. Někteří zkušení uživatelé pohlížejí na začátečníky jako na hlupáky, protože neznají všechny „triky“ s editorem. Já osobně tento druh elitářství odsuzuji, protože jsem také kdysi byl začátečníkem. Nyní se však opět věnujme editoru.

Pro pohyb kurzoru existují některé další klávesy. Víme, že pro pohyb kurzoru o jeden znak doprava je určena kombinace kláves C-F. Chcete-li „poskočit“ s kurzorem o celé slovo, zadejte M-F. Vyzkoušejte si, co udělá příkaz M-B. To ale není vše. Pokud zapisujete věty tak, aby za poslední tečkou byly vždy dvě mezery, můžete pomocí klávesy M-E přeskakovat celé věty. Dvě mezery jsou podmínkou, protože jinak by editor nebyl schopen konec věty identifikovat. Opět si vyzkoušejte, jak funguje kombinace kláves M-A.

Možná si to ani neuvědomujete a používáte opakovaně kombinace kláves C-F k posunu kurzoru na konec řádku. Připomínáme, že k tomuto účelu je určena kombinace kláves C-E a k nastavení kurzoru na začátek řádku je určena kombinace kláves C-A. Možná, že často používáte kombinaci kláves pro posun kurzoru na následující řádek, a přitom byste měli použít kombinaci kláves C-V, kdy se vám zobrazí následující stránka. Zrovna tak využívejte kombinaci kláves M-V.

Pokud jste někde u konce řádku a všimnete si, že jste někde na začátku udělali chybu, pak nepoužívejte klávesy Backspace nebo Delete, abyste se dostali k chybnému místu. Museli byste jinak dobře napsaný řádek psát znovu. Místo toho používejte kombinaci kláves M-B, C-B, případně C-F, opravte chybu a pak se pomocí C-E vraťte na konec řádku.

Pokud zadáváte jméno souboru, nezadávejte je nikdy celé. Zadejte jen nezbytný počet znaků, které soubor jednoznačně identifikují. Pak vám editor Emacs zbývající znaky po stisknutí klávesy Tab nebo mezerníku automaticky doplní. Šetřte sebe a ne procesor!

Když píšete nějaký dlouhý text, používejte funkci pro automatické ukončování řádků. Kombinace kláves M-Q vyvolá funkci `fill-paragraph` a lze jej použít ve všech textových módech. Také jej můžete použít k „zarovnání“ již napsaného odstavce. Stačí nastavit kurzor na nějakou pozici uvnitř odstavce a stisknout M-Q.

Někdy je užitečné použít kombinaci kláves C-X U, kdy se editor pokusí vrátit zpět provedené změny. Editor Emacs v tomto případě odhadne, kolik změn má vrátit, a jeho odhad je obvykle velmi inteligentní. Kombinace kláves C-X U můžete použít opakovaně až do okamžiku, kdy editor vrátí poslední změnu, kterou si „pamatuje“.

Konfigurace editoru Emacs

Editor Emacs je tak velký a tak komplexní program, že má i svůj vlastní programovací jazyk. Vlastnosti editoru můžete modifikovat pomocí vlastních programů. Programovací jazyk integrovaný v editoru Emacs se jmenuje Emacs Lisp a je dialektem jazyka Lisp. Pokud máte s jazykem Lisp nějaké zkušenosti, pak se vám bude zdát opravdu přátelským. Pokud nemáte, ničeho se neobávejte. V tomto oddíle se nebudeme programováním v editoru Emacs zabývat příliš do hloubky a pokud se s jazykem Emacs Lisp budete chtít seznámit podrobněji, pak si prostudujte stránky dostupné v systému Info.

Většina funkcí editoru Emacs je definována pomocí kódu napsaného v jazyku Emacs Lisp.[9] Většina z těchto souborů je distribuována spolu s editorem a kolektivně se nazývají „Emacs Lisp library". Umístění této knihovny závisí na tom, jakým způsobem je editor Emacs instalován ve vašem systému. Nejčastěji jej můžete najít v adresáři: `/usr/lib/emacs/lisp`, `/usr/lib/emacs/19.19/lisp` a podobně. Číslo `19.19` značí verzi editoru Emacs a může se lišit od verze ve vašem systému.

Informace o uložení knihovny je uložena v interní proměnné **load-path** editoru Emacs, proto ji nemusíte pracně hledat v souborovém systému. Jestliže chcete zjistit hodnotu této proměnné, musíte ji **vyhodnotit**. To znamená, že musíte aktivovat interpretr Emacs Lisp. V editoru Emacs existuje speciální mód pro vyhodnocování výrazů jazyka Lisp zvaný **lisp-interaction-mode**. S tímto módem je obvykle spjat buffer „`*scratch*`". Pokud se vám jej nepodaří nalézt, vytvořte nový buffer s jakýmkoliv jménem a zadejte příkaz M-X `lisp-interaction-mode`.

Nyní se nacházíte v pracovním prostoru pro interaktivní komunikaci s interpretrem Emacs Lisp. Zadejte příkaz:

```
load-path
```

a pak stiskněte C-J. V módu interaktivní komunikace je kombinace kláves C-J svázán s funkcí `eval-print-last-sexp`. Slovo „sexp" znamená „s-expression" (**s-výraz**), což znamená vyváženou skupinu závorek – to je opravdu řečeno velmi zjednodušeně, ale brzy budete tušit, k čemu jsou takové výrazy užitečné při práci z jazykem Emacs Lisp. V každém případě po vyhodnocení proměnné `load-path` obdržíte zprávu, která bude vypadat přibližně takto:

```
load-path Ctrl+j
("/usr/lib/emacs/site-lisp/vm.5.35" "/home/kfogel/elithp"
"/usr/lib/emacs/site-lisp" "/usr/lib/emacs/19.19/lisp")
```

Toto hlášení nebude vypadat v každém systému stejně, protože závisí na způsobu instalace editoru Emacs. Uvedený příklad pochází z mého počítače s procesorem 386, na němž běží operační systém Linux. Jak vyplývá z předcházejícího výpisu, proměnná `load-path` je seznam řetězců. Každý řetězec uvádí adresář, který by mohl obsahovat soubory náležící do systému Emacs. Když potřebuje editor Emacs zavést soubory obsahující kód Lisp, hledá tyto soubory v uvedených adresářích v uvedeném pořadí. Pokud je v proměnné `load-path` uveden adresář, který v souborovém systému neexistuje, editor jej ignoruje.

Ve fázi startování se editor Emacs pokouší nalézt soubor `.emacs` ve vašem domovském adresáři. Proto platí, že pokud chcete mít svoji vlastní konfiguraci editoru Emacs, měli byste používat soubor `.emacs`. Nejvýznamnější konfigurační nastavení se týkají vazeb mezi kombinaci kláves a funkcemi, proto se jim nyní budeme věnovat.

```
(global-set-key "\Ctrl+cl" 'goto-line)
```

Funkce `global-set-key` má dva argumenty: prvním z nich je kombinace kláves a druhým je funkce, která se má po stisknutí tohoto klíče realizovat. Slovo „`global`" znamená, že uvedená vazba mezi kombinacemi kláves a funkcí bude platit ve všech hlavních módech. Existuje jiná funkce, `local-set-key`, jež nastavuje vazbu mezi klíčem a funkcí pro jediný buffer. V uvedeném příkladu jsme nastavili vazbu mezi kombinaci kláves C-C a funkcí `goto-line`. Při definici kombinace kláves se musí použít řetězec, což znamená, že se kombinace kláves musí uzavřít do uvozovek. Speciální syntaxe „`\Ctrl+<char>`" znamená, že se má stisknout klávesa Ctrl, držet stisknutá, a pak se má stisknout klávesa `<char>`. Podobně platí, že „`\Alt+<char>`" indikuje kombinaci s klávesou Meta.

9 Někdy se neoficiálně nazývá „Elisp".

Konfigurace vazby mezi kombinaci kláves a funkcí vypadá jednoduše, ale kde získat informace o funkci „goto-line“? Nebo naopak, předpokládejme, že chci kombinaci kláves C-C L svázat s funkcí, která umožní přeskočit na řádek specifikovaného pořadového čísla, ale jak mám zjistit jméno takové funkce?

V tomto okamžiku využijete vynikajících vlastností systému nápovědy, který je integrován v editoru Emacs. Jakmile se jednou rozhodnete, jaký druh funkce chcete použít, můžete použít editor Emacs k „vystopování“ jejího jména. Jedna sice rychlá, ale ne příliš přímočará metoda spočívá v tom, že se využije schopnosti editoru Emacs doplňovat jména funkcí. Měli byste si pamatovat, že kombinace kláves C-H F vyvolá nápovědu popisující funkci (t.j. vyvolá funkci `describe-function`). Pak stiskněte Tab, aniž cokoliv zadáte. Tak „donutíte“ editor Emacs doplnit prázdný řetězec. Jinými slovy, editor vypíše jména všech funkcí, které jsou v něm interně definovány. Uvedená operace bude chvíli trvat, protože takových funkcí je v editoru definováno opravdu mnoho.

Nyní stiskněte kombinaci kláves C-G, čímž funkci `describe-function` přerušíte. Nyní v editoru existuje buffer nazvaný „`*Completions*`“, který obsahuje požadovaný seznam všech interních funkcí editoru Emacs. Přepněte se do tohoto bufferu a pomocí postupného vyhledávání najděte funkci, kterou chcete použít. Můžete například využít předpokladu, že funkce pro přechod na řádek specifikovaného čísla bude ve svém názvu obsahovat slovo „`line`“. Proto zadejte vyhledávání slova „`line`“ a najdete tak všechny eventuality.

Vhodnější metoda spočívá v tom, že použijete kombinaci kláves C-H A (t.j. vyvoláte funkci `command-apropos`), kdy se vám zobrazí všechny funkce obsahující specifikovaný řetězec. Výstup z funkce `command-apropos` se poněkud hůře třídí, než výstup z funkce `describe-function`. Vyzkoušejte si obě metody a vyberte tu, která vám bude lépe vyhovovat.

Samozřejmě se může stát, že budete hledat funkci, která v editoru Emacs neexistuje. V takovém případě si ji budete muset napsat sami. Nebudeme zde popisovat, jak se v jazyku Emacs Lisp programuje. Doporučujeme prostudovat si příklady v knihovně Emacs Lisp a přečíst si stránky systému Info popisující jazyk Emacs Lisp. Pokud znáte někoho, kdo programování v jazyku Emacs Lisp ovládá, jistě vám pomůže.

Definice vlastních funkcí editoru Emacs není nic těžkého. Abych vás trochu navnadil, za poslední rok jsem jich bez problémů vytvořil 131. Vyžaduje to trochu zkušeností, ale programování v jazyku Emacs Lisp zvládnete poměrně rychle.

Další konfigurační možnosti spočívají v tom, že se v souboru `.emacs` nastaví hodnoty jistých proměnných. Přidejte například do vašeho souboru `.emacs` následující řádek a pak znovu nastartujte editor Emacs:

```
(setq inhibit-startup-message t)
```

Editor Emacs kontroluje ve fázi startování proměnnsou `inhibit-startup-message` a podle její hodnoty se rozhodne, zda zobrazovat jisté informace (o verzi editoru, zárukách a podobně) nebo ne. Uvedený výraz jazyka Lisp používá příkaz `setq` k nastavení proměnné `inhibit-startup-message` na hodnotu „t“ což je speciální hodnota v jazyce Lisp pro „true“. Opakem je hodnota „nil“ což je speciální hodnota pro „false“. V mém konfiguračním souboru `.emacs` se nacházejí některá nastavení, jež možná shledáte užitečnými:

```
(setq case-fold-search nil) ; gives case insensitivity in searching
;; make C programs ident the way I like them to:
(setq c-indent-level 2)
```

První výraz nastavuje způsob vyhledávání na tzv. case-insensitive, což znamená, že se při vyhledávání nerozlišují malá a velká písmena (a to dokonce i tehdy, když hledaný řetězec obsahuje buď pouze malá, nebo pouze velká písmena). V druhém výrazu se nastavuje odsazování řádků při psaní programů v jazyku C na hodnotu 2. Je to poněkud méně, než je implicitní nastavení, ale to záleží na individuálním vkusu a na tom, jaká hodnota odsazení učiní váš program napsaný v jazyku C čitelnějším.

V jazyku Lisp se komentářový řádek označuje znakem „;“. Editor Emacs ignoruje vše, co se nachází za tímto znakem. Výjimku tvoří případ, kdy se znak „;“ nachází uvnitř řetězce. Například:

```
;; Tyto dva řádky jsou v jazyku Lisp ignorovány, ale
;; následující s-výraz bude plně vyhodnocen:
(setq some-literal-string "An awkward pause; for no purpose.")
```

Doporučuje se, abyste změny ve zdrojových souborech pro jazyk Lisp komentovali, protože jinak například po šesti měsících určitě zapomenete, jakou změnu jste dělali. Komentář na celý řádek uvádějte dvojicí znaků `;;`. Pak bude editor Emacs správně provádět odsazování řádků.

To, co jsme si řekli o vyhledávání interních funkcí editoru Emacs, platí i pro vyhledávání proměnných. Chcete-li vytvořit seznam všech proměnných, zadejte příkaz C-H-C (describe-variable), nebo použijte C-H C-A (apropos). Použijete-li druhou možnost, pak musíte počítat s tím, že vám příkaz vyhledá funkce a proměnné dohromady.

Soubory se zdrojovým kódem v jazyku Emacs Lisp mají implicitní příponu „`.el`“, například „`c-mode.el`“. Aby však mohl kód vytvořený v jazyku Emacs Lisp běžet rychleji, umožňuje editor provést jakousi **předkompilaci** (soubory označované jako „**byte-compiled**“) a pak mají tyto soubory implicitní příponu „`.elc`“. Výjimku, pokud jde o implicitní příponu, tvoří soubor `.emacs`, jenž příponu „`.el`“ nepotřebuje, neboť jej editor hledá automaticky ve fázi startování.

Chcete-li zavést interaktivně soubor s kódem v jazyku Lisp, použijte příkaz `M-X load-file`. Editor vás vyzve k zadání jména souboru a pak specifikovaný soubor zavede do své pracovní oblasti. Jestliže chcete zavést soubor „zevnitř“ jiného souboru napsaného v jazyku Lisp, postupujte takto:

```
(load "c-mode") ; tento příkaz zavede buď soubor c-mode.el, nebo c-mode.elc
```

Editor Emacs nejdříve k uvedenému jménu přidá příponu `.elc` a pokusí se jej nalézt v některém adresáři specifikovaném v proměnné `load-path`. Pokud jej nenajde, přidá příponu `.el` a hledání zopakuje. Jestliže není úspěšný ani v tomto případě, pak použije přímo řetězec předaný funkci `load`. Překlad můžete realizovat prostřednictvím příkazu `Alt+X byte-compile-file`. Pokud však zdrojový soubor často aktualizujete, pak to zpravidla nemá smysl. Soubor .emacs nikdy nepřekládejte, ani mu nepřidávejte příponu `.el`.

Bezprostředně po zavedení souboru `.emacs` vyhledá editor Emacs soubor `default.el` a zavede jej. Tento soubor je obvykle umístěn v adresáři uvedeném v proměnné `load-path` s názvem `site-lisp` nebo `local-elisp` či v nějakém podobném adresáři (podívejte se na proměnnou `load-path`). Uživatelé, kteří provádějí údržbu editoru Emacs používají soubor `default.el` k nastavení globálních konfigurací, které pak platí pro každého uživatele v systému. Soubor `default.el` by také neměl být kompilován, protože se v něm často provádějí změny.

Jestliže váš osobní soubor `.emacs` obsahuje nějakou chybu, pak se editor Emacs nebude pokoušet načíst soubor `default.el`, ale zastaví svou činnost a vypíše hlášení: „`Error in init file`“. Uvidíte-li takové hlášení, pak jste pravděpodobně udělali nějakou chybu v souboru `.emacs`.

V souboru `.emacs` se ještě vyskytuje jeden druh výrazů. Knihovna Emacs Lisp někdy nabízí více sad programů, které realizují tytéž funkce různým způsobem. To znamená, že musíte specifikovat tu sadu, která se má používat (implicitní sada nemusí být pro vás vždy tou nejlepší). Tyto sady se například uplatňují v oblasti interaktivního rozhraní pro jazyk Scheme. S editorem Emacs se standardně distribuují dvě sady funkcí interaktivního rozhraní pro jazyk Scheme: `xscheme` a `cmuscheme`.

```
prompt>Is /usr/lib/emacs/19.19/lisp/*scheme*
/usr/lib/emacs/19.19/lisp/cmuscheme.el
/usr/lib/emacs/19.19/lisp/cmuscheme.elc
/usr/lib/emacs/19.19/lisp/scheme.el
/usr/lib/emacs/19.19/lisp/scheme.elc
/usr/lib/emacs/19.19/lisp/xscheme.el
/usr/lib/emacs/19.19/lisp/xscheme.elc
```

Já osobně dávám přednost sadě `cmuscheme` před sadou `xscheme`, avšak editor Emacs implicitně používá sadu `xscheme`. Jak „donutit" editor Emacs, aby byl nakonfigurován v souladu s mým přáním? Do souboru `.emacs` jsem vložil následující řádky:

```
;; notice how the expression can be broken across two lines. Lisp
;; ignores whitespaces, generally:
(autoload 'run-scheme "cmuscheme"
"Run an inferior Scheme, the way I like it". t)
```

Funkce `autoload` akceptuje jako argument jméno funkce (začínající apostrofem ') a „sdělí" editoru Emacs, že je tato funkce definována v jistém souboru. Jméno tohoto souboru se předává jako druhý argument, tedy řetězec (buď s příponou „`.el`", nebo „`.elc`"). Soubor pak bude vyhledán v adresářích definovaných v proměnné `load-path`.

Zbývající argumenty jsou volitelné, ale jsou nezbytné v tomto případě: Třetí argument je dokumentačním řetězcem pro specifikovanou funkci. Pokud použijete funkci `describe-function`, pak se jako popis k vyhledané funkci objeví právě tento řetězec.

Čtvrtý argument „sdělí" editoru Emacs, že specifikovaná funkce může být volána interaktivně, t.j. pomocí kombinace kláves Alt+X. V tomto případě je to velmi důležité, protože jedině tak lze spustit proces Scheme v prostředí editoru Emacs pomocí příkazu `Alt+x run-scheme`.

Nyní, když je funkce `run-scheme` definována jako automaticky zaveditelná, co se stane, když se zadá příkaz `Alt+x run-scheme`? Editor Emacs vyhledá funkci `run-scheme`, zjistí, zda je automaticky zaveditelná a zavede specifikovaný soubor (v našem případě „`cmuscheme`"). Protože kompilovaný soubor `cmuscheme.elc` existuje, editor jej zavede. Tento soubor musí definovat funkci `run-scheme`, jinak dojde při jeho zavádění k chybě. Naštěstí tuto funkci skutečně definuje, proto jde vše hladce a já mám k dispozici své oblíbené rozhraní pro jazyk Scheme.[10]

Automatické zavedení funkce zajišťuje, že bude editor Emacs schopen funkci najít, až ji budete potřebovat. Navíc můžete nad automaticky zaváděnými funkcemi získat jistou kontrolu. Dále platí, že automaticky zaveditelné funkce šetří paměť spotřebovanou editorem Emacs, protože se tyto funkce zavádějí až v okamžiku, kdy jsou potřebné. Řada příkazů není ve fázi startování editoru ve skutečnosti definována. Místo toho jsou nastaveny jako automaticky zaveditelné funkce. Pokud takový příkaz nezadáte, nikdy se odpovídající funkce nezavede. Uvedená vlastnost automatického zavádění funkcí je pro editor Emacs (a hlavně pro uživatele) „životně" důležitou. Kdyby editor ve

10 Jistě jste si všimli, že o rozhraní cmuscheme jsme hovořili již dříve. Proto byste si měli zavedení sady cmuscheme vyzkoušet.

fázi startování zaváděl všechny funkce, trvala by tato fáze asi dvacet minut a celá dostupná paměť vašeho počítače by byla obsazena. O automatické zavádění implicitních funkcí se nemusíte starat, editor Emacs je realizuje sám.

Kde získat další informace

V této kapitole jsme zdaleka neuvedli vše, co byste měli znát o editoru Emacs. Je zde zhruba jedno procento informací. Budete-li editor intenzivně využívat, pak budete potřebovat znát spoustu dalších „triků“ šetřících čas. Nejjednodušší bude, když vyčkáte, až budete muset řešit nějaký konkrétní problém. Pak vyhledáte funkci, která váš problém vyřeší.

Snad nejdůležitější je, abyste uměli plně využívat systém nápovědy integrovaný v editoru Emacs. Řekněme například, že budete chtít vložit do editovaného textu obsah jiného souboru. Snadno uhodnete, že asi existuje funkce `insert-file` (vložení souboru). Máte-li svou domněnku potvrdit, použijte příkaz Ⓒ-Ⓗ Ⓕ. V příkazovém řádku zadejte jméno funkce, kterou hledáte a o které si chcete přečíst nějaké informace. Protože víte, že editor Emacs je schopen doplňovat jména funkcí, můžete jako „počáteční“ odhad uvést řetězec „insert“. Pak stačí stisknout klávesu Tab. Objeví se vám seznam všech funkcí obsahujících řetězec „insert“ a funkce „`insert-file`“ je jedna z nich.

Nyní si můžete přečíst spoustu informací o funkci `insert-file` a také ji můžete použít – stačí zadat příkaz `Alt+x insert-file`. Chcete-li také zjistit, zda je tato funkce svázána s nějakým klíčem, zadejte příkaz `Ctrl+h w insert-file` a stiskněte Enter. Čím lépe budete znát vlastnosti systému nápovědy v editoru Emacs, tím snáze naleznete potřebné informace. Máte-li navíc dostatek erudice zkoumat nové věci a ochotu učit se, ušetříte mnoho času.

Jestliže si chcete objednat kopii manuálu k editoru Emacs a/nebo manuál „*Emacs Lisp Programming*“, pošlete objednávku na adresu:

Free Software Foundation
675 Mass Ave
Cambridge, MA 02139
USA

Oba tyto manuály jsou distribuovány v elektronické podobě spolu s editorem Emacs a jsou čitelné prostřednictvím systému Info (použijte klávesu Ⓒ-Ⓗ Ⓘ, pomocí které inicializujete rozhraní mezi editorem Emacs a systémem Info). Na druhé straně, cena za uvedené manuály je opravdu mírná a navíc se získané peníze budou věnovat na vývoj kvalitního volně šířitelného programového vybavení. Také chceme poznamenat, že pomocí kombinace kláves Ⓒ-Ⓗ Ⓒ-Ⓒ si můžete zobrazit informace o licenčních podmínkách platných pro editor Emacs. Jsou určitě zajímavější, než si teď myslíte, a pomohou vám vyjasnit si pojem „volné programové vybavení“. Pokud si myslíte, že pojem „volné programové vybavení“ znamená, že daný program nic nestojí, pak si licenční podmínky rychle přečtěte.

KAPITOLA 9

Konfigurace operačního systému Unix

Konfigurace příkazového interpretu bash

Filosofie operačního systému Unix se od filosofie jiných operačních systémů podstatně liší v jedné věci. Autoři Unixu se nepokoušeli předvídat všechny potřeby všech uživatelů. Místo toho se pokusili navrhnout operační systém tak, aby si každý uživatel pracovní prostředí svého operačního systému snadno upravil sám podle svých potřeb. Konfigurace jednotlivých programů operačního systému Unix se definuje prostřednictvím tzv. **konfiguračních souborů**. Někdy jsou označovány jako „ini-files“ nebo „rc files“ nebo dokonce jako „dot files“ (jedná se zpravidla o soubory, jejichž jména začínají tečkou). Pokud si vzpomínáte, tak jména souborů začínající znakem „.“ nejsou normálně příkazem `ls` zobrazována.

Nejdůležitější konfigurační soubory jsou ty, které používá příkazový interpret. Implicitním příkazovým procesorem v operačním systému Linux je `bash` a právě jemu bude věnována tato kapitola. Než začneme popisovat, jak příkazový interpret `bash` konfigurovat, podívejme se na soubory, které `bash` vyhledává.

Inicializace příkazového interpretu bash

Existuje několik různých způsobů, jak příkazový procesor `bash` spustit. Tzv. **login-shell** se automaticky spouští poté, co se přihlásíte do systému.

Další způsob spočívá ve spuštění **interaktivního příkazového procesoru** (interactive shell). To je jakýkoliv příkazový procesor, který se prezentuje příkazovým řádkem. Příkazový interpret, jenž se spustí bezprostředně po přihlášení se do systému, je také interaktivní. Jiným příkladem interaktivního příkazového interpretu je program xterm spuštěný z prostředí X Window.

Existují také **neinteraktivní příkazové interprety.** Tyto procesory se používají k vykonávání příkazů uvedených v souboru, jako jsou dávkové soubory s příponou .BAT v operačním systému MS-DOS. Analogií v operačním systému Unix jsou tzv. **skripty**. Skript příkazového procesoru je něco jako miniprogram. Jsou sice výrazně pomalejší než kompilovaný program, ale snadno se vytvářejí a modifikují.

Příkazové procesory v operačním systému Unix používají v závislosti na typu následující inicializační soubory:

Typ příkazového procesoru	inicializační soubor
Interaktivní příkazový interpret spuštěný při přihlášení se do systému	`.bash_profile`
Interaktivní příkazový procesor	`.bashrc`
Neinteraktivní příkazový procesor	skript příkazového procesoru

Inicializační soubory

Protože většina uživatelů chce mít stejné uživatelské prostředí bez ohledu na to, jaký typ příkazového procesoru se spustí, začneme konfiguraci tím, že do konfiguračního souboru `.bash_profile` vložíme jednoduchý příkaz „source ~/.bashrc“. Příkaz source „sdělí“ příkazovému procesoru, že má jeho argument interpretovat jako skript. To znamená, že se při každém spuštění skriptu `.bash_profile` také spustí skript `.bashrc`.

Nyní budeme přidávat příkazy do našeho souboru `.bashrc`. Do souboru `.bash_profile` se zadávají pouze příkazy, které se mají spouštět při přihlášení se do systému.

Vytváření aliasů

Jakým způsobem lze měnit konfigurační nastavení? Hned uvedeme příklad, který si do svého konfiguračního souboru `.bashrc` zadává devadesát procent uživatelů operačního systému Unix:

```
alias ll="ls -l"
```

Příkaz definuje tzv. **alias** (druhé jméno) příkazu. V našem případě se jméno příkazu `ll` expanduje na příkaz příkazového procesoru `"ls -l"`. Za předpokladu, že příkazový procesor bash přečetl uvedenou definici ve vašem konfiguračním souboru `.bashrc`, pak má příkaz `ll` stejný efekt jako příkaz `ls -l`. Přitom ušetříte polovinu stisknutých kláves. Když zadáte v příkazovém řádku `ll` a stisknete Enter, příkazový procesor zadaný příkaz rozvine podle definice a pak realizuje příkaz `ls -l`. Ve skutečnosti v systému příkaz `ll` neexistuje, ale příkazový procesor `bash` jej automaticky transformuje na platný program.

Některé příklady aliasů jsou uvedeny na této straně dole. Můžete si je vložit do vašeho souboru `.bashrc`. Zvláště zajímavý je první z nich. Je-li definován alias `ls="ls -F"`, pak po každém zadání příkazu `ls` bude automaticky aplikovat volbu `-F`. Platí, že se alias nikdy nepokusí rekurzivně rozšířit sám sebe. Uvedený příklad demonstruje nejčastěji používaný způsob automatického přidávání voleb do příkazů.

Všimněte si znaku „#“. Ve skriptech příkazového procesoru se tento znak používá k označení komentáře a příkazový procesor zbytek řádku za znakem # ignoruje.

Dále jste si mohli všimnout několika dalších věcí. Především se v některých definicích nevyskytují uvozovky, například v definici `pu`. Platí totiž, že pokud se na pravé straně znaku rovná se (=) vyskytuje jediné slovo, pak se uvozovky nemusejí uvádět.

Nic by se nestalo, kdyby zde uvozovky byly uvedeny. Určitě však uvozovky používejte v případě, když budete definovat nové jméno pro příkaz s volbami a/nebo argumenty. Například:

```
alias rf="refrobnicate -verbose -prolix -wordy -o foo.out"
alias ls="ls -F"              # give characters at end of listing
alias ll="ls -l"              # special ls
alias la="ls -a"
alias ro="rm  *~; rm.*~"      # removes backup files created by Emacs
alias rd="rmdir"              # saves typing!
alias md="mkdir"
alias pu=pushd         # pushd, popd, and dirs weren't covered
alias po=popd          # manual---you might want to look them up
alias ds=dirs          # in the bash manpage
# these all are just keyboard shortcuts
alias to="telnet cs.oberlin.edu"
alias ta="telnet altair.mcs.anl.gov"
alias tg="telnet wombat.gnu.ai.mit.edu"
alias tko="talk kold@cs.oberlin.edu"
alias tjo="talk jimb@cs.oberlin.edu"
alias mroe="more"             # spelling correction!
alias moer="more"
alias email="emacs -f rmail"  # my mail reader
alias ed2="emacs -d floss:0 -fg \"grey95\" -bg \"grey50\""
                                    # one way of invoking emacs
```

Asi se vám zdá, že poslední alias má divně umístěné uvozovky:

```
alias ed2="emacs -d floss:0 -fg \"grey95\" -bg \"grey50\""
```

Jak asi tušíte, chtěl jsem umístit uvozovky do samotných voleb. Proto jsem musel před každou uvozovku vložit obrácené lomítko. Příkazový procesor pak takovou uvozovku nebude interpretovat obvyklým způsobem.

Všimněte si také dvou definic pro opravu chybně zadaného příkazu `more`. Pokud omylem zadám „`mroe`“ nebo „`moer`“, bude příkazový procesor „vědět“, že jsem chtěl zadat „`more`“. Aliasy neinterferují s argumenty předávanými programům. To znamená, že například příkaz

```
/home/larry# mroe hurd.txt
```

bude fungovat dobře.

Určitě platí, že správné používání definice aliasů představuje polovinu konfiguračních možností, které jsou u příkazových procesorů k dispozici. Při práci v operačním systému Unix si všímejte, které příkazy často zadáváte, a pak si pro ně pořiďte druhá jména, tedy zkratky. Zjistíte, že se vám pak bude pracovat mnohem radostněji.

Systémové proměnné

V souboru `.bashrc` se definují další důležitá konfigurační nastavení prostřednictvím systémových proměnných (environment variables). Co jsou to systémové proměnné? Pojďme na to z druhé strany: předpokládejme, že čtete dokumentaci k programu `fruggle` a že narazíte na následující věty:

> *Fruggle normally looks for its configuration file, .frugglerc, in the user's home directory. However, if the environment variable FRUGGLEPATH is set to a different filename, it will look there instead.*

Každý program běží v nějakém **prostředí** a to je definováno příkazovým interpretem, jenž program volá.[1] Lze si představit, že prostředí existuje uvnitř příkazového interpretu. Programátoři mají k dispozici speciální funkci pro získání hodnoty systémové proměnné a program `fruggle` tuto funkci využívá. To znamená, že ověří hodnotu v systémové proměnné `FRUGGLEPATH`. Pokud není tato systémová proměnná definována, pak program použije soubor `.frugglerc` ve vašem domovském adresáři. Pokud však je definována, program `fruggle` použije její hodnotu místo implicitního souboru `.frugglerc`.

Nyní si uveďme ukázku, jak změnit prostředí v příkazovém procesoru `bash`:

```
/home/larry# export PGPPATH=/home/larry/secrets/pgp
```

Pod příkazem `export` si můžete představit následující větu: „Exportuj tuto proměnnou do prostředí, ze kterého budu spouštět program, tak, aby proměnná byla z tohoto programu viditelná.“ Později uvidíte, že jsou i jiné důvody pro používání příkazu `export`.

Uvedenou systémovou proměnnou používá program `pgp` pro šifrování, jehož autorem je Phil Zimmerman. Program `pgp` implicitně používá váš domovský adresář jako adresář, ve kterém hledá jisté soubory (šifrovací klíče). Také tento adresář využívá k vytvoření dočasných pracovních souborů. Nastavením systémové proměnné `PGPPATH` jsem změnil pracovní adresář na `/home/larry/secrets/pgp`. Abych zjistil přesné jméno této systémové proměnné, musel jsem si přečíst manuál k programu `pgp`. Systémové proměnné se zpravidla uvádějí velkými písmeny a končí na „`PATH`“.

Je užitečné vědět, jak hodnotu systémové proměnné zjistit:

```
/home/larry# echo $PGPPATH
.pgp
/home/larry#
```

Všimněte si, že jsme před jméno systémové proměnné uvedli znak `$`. Jedině tak lze zjistit hodnotu systémové proměnné. Pokud byste znak dolaru neuvedli, dostali byste následující výpis:

```
/home/larry# echo PGPPATH
PGPPATH
/home/larry#
```

Znak dolaru se používá k vyhodnocení systémové proměnné, avšak pouze v kontextu s příkazovým procesorem – přesněji s příkazovým procesorem, jenž realizuje interpretaci příkazu. V jakých případech realizuje příkazový procesor interpretaci?

Proměnná	Obsahuje	Příklad
`HOME`	**Váš domovský adresář**	`/home/larry`
`TERM`	**Typ vašeho terminálu**	`xterm, vt100, console`
`SHELL`	**Cesta k vašemu příkazovému procesoru**	`/bin/bash`
`USER`	**Jméno vašeho účtu**	`larry`
`PATH`	**Seznam adresářů, ve kterých se automaticky vyhledávají programy ke spuštění**	`/bin:/usr/local/bin:/usr/bin/X11`

Tabulka 9.1 – Některé důležité systémové proměnné

1 Nyní vidíte, proč jsou příkazové procesory tak důležité. Představte si, že byste museli definovat celé prostředí, kdykoliv budete spouštět nějaký program!

Je to v těch případech, kdy zadáváte příkaz z příkazového řádku nebo kdy příkazový procesor čte příkazy z nějakého souboru, například ze souboru `.bashrc`.

Existuje další důležitý příkaz, pomocí kterého lze získat informace o prostředí. Tímto příkazem je `env`. Po zadání příkazu `env` se vám zobrazí seznam všech systémových proměnných. Jste-li uživateli systému X Window, pak bude tento seznam velmi dlouhý, proto používejte příkaz `env | more`.

Některé systémové proměnné jsou opravdu důležité, proto jsou uvedeny v tabulce 9.1. Tyto systémové proměnné se automaticky definují po přihlášení se do systému. Není proto nutné je nastavovat v souboru `.bashrc` nebo `.bash_login`.

Nyní se blíže podívejme na systémovou proměnnou `TERM`. Abychom jí mohli porozumět, podívejme se zpět do historie operačního systému Unix. Operační systém musí znát jisté údaje o vaší konzole, aby mohl realizovat takové funkce, jako je zápis znaků na obrazovku, pohyb kurzoru v textovém řádku a podobně. V počátcích rozvoje výpočetní techniky výrobci terminálů průběžně rozšiřovali jejich vlastnosti: nejdříve inverzní video, později znaky pro evropské jazyky, dokonce i první funkce pro kreslení (připomínáme, že jde o dobu, kdyse ještě nikomu ani nesnilo o grafických oknech a myších). Každá nová vlastnost však přinášela programátorům problémy: jak měli vědět, co terminál podporuje a co ne? Jak měli podporovat nové vlastnosti a přitom „neodepsat“ staré terminály?

V operačním systému Unix najdete odpověď na tyto otázky v souboru `/etc/termcap`. Soubor `/etc/termcap` obsahuje seznam všech terminálů, o kterých váš operační systém „ví,“ a dále informace, jakým způsobem se má řídit kurzor. Pokud systémový správce dostane nový terminál, pak pouze do souboru `/etc/termcap` přidá záznam vztahující se k novému terminálu a nemusí pracně konfigurovat celý operační systém Unix. Někdy je situace ještě jednodušší. Kdysi se stal terminál vt100 od firmy Digital Equipment Corporation jakýmsi pseudostandardem, který převážná většina moderních terminálů respektuje a umí emulovat.

V operačním systému Linux je někdy hodnota systémové proměnné `TERM` nastavena jako console, což znamená emulaci terminálu vt100 s některými speciálními funkcemi.

Další proměnná, `PATH`, je rovněž kritickou systémovou proměnnou z hlediska funkčnosti příkazového procesoru. Zde uvádím nastavení na mém počítači:

```
/home/larry# env | grep ^PATH
PATH=/home/larry/bin:/bin:/usr/bin:/usr/local/bin:/usr/bin/X11:/usr/TeX/bin
/home/larry#
```

Systémová proměnná `PATH` obsahuje seznam adresářů oddělených dvojtečkou, ve kterých systém automaticky vyhledává spustitelné programy. Když například zadám příkaz `ls` a stisknu klávesu Enter, bude příkazový procesor `bash` hledat program `ls` nejdříve v adresáři `/home/larry/bin`, který jsem vytvořil pro ukládání mých vlastních programů.

Program `ls` jsem však nenapsal já (ten byl pravděpodobně napsán ještě před tím, než jsem se narodil), proto zde příkazový procesor tento příkaz nenašel. Jako další prohledává příkazový procesor adresář /bin. A zde program `ls` našel. Protože soubor `ls` je spustitelný program, přestane příkazový procesor dále hledat a spustí jej. Může se stát, že v jiném adresáři bude také uložen spustitelný program `ls` (například v adresáři `/usr/bin`), ale příkazový procesor jej nespustí, pokud mu to výslovně nepřikážete:

```
/home/larry# /usr/bin/ls
```

Systémová proměnná PATH existuje hlavně proto, abychom nemuseli při každém spouštění nějakého programu zadávat celou cestu k tomuto programu. Zadáte-li tedy jakýkoliv příkaz, prohledá příkazový procesor všechny adresáře uvedené v systémové proměnné PATH. Když jej najde, spustí jej, a pokud ne, vypíše následující zprávu:

```
/home/larry# clubly
clubly: command not found
```

Všimněte si, že moje systémová proměnná PATH neobsahuje aktuální adresář, tedy „.“. Pokud by obsahovala, pak by vypadala takto:

```
/home/larry# echo $PATH
.:/home/larry/bin:/bin:/usr/bin:/usr/local/bin:/usr/bin/X11:/usr/TeX/bin
/home/larry#
```

Zda zadávat nebo nezadávat aktuální adresář do systémové proměnné PATH je předmětem diskuse v kuloárech okolo operačního systému Unix. Problém tkví v tom, že aktuální adresář v systémové proměnné PATH by mohl představovat „díru“ v bezpečnosti systému. Předpokládejme, že se přepnete do adresáře, ve kterém někdo nechal virový program „Trójský kůň“ a nazval jej ls. Vy nic zlého netušíte a zadáte příkaz ls, což je přirozené, chcete-li se seznámit s novým adresářem. Protože je aktuální adresář uveden v systémové proměnné PATH jako první, spustí příkazový procesor právě tuto verzi příkazu ls. I když jste nechtěli udělat nic špatného, rozpoutali jste ve vašem systému virovou nákazu. Proti takovým haváriím neexistuje dost účinná ochrana. Virový program může spustit osoba, která ani nemá privilegia uživatele root. Stačí, aby měla právo zapisovat do adresáře, ve kterém se virový program nachází. Dokonce to může být její domovský adresář.

Pokud jde o váš systém, je pravděpodobné, že jeden uživatel neklade druhému různé nástrahy ve formě „Trojských koňů“ a že kolektiv uživatelů je založen na přátelských a kolegiálních vztazích. Avšak ve velkých systémech s mnoha uživateli (jako jsou například universitní počítače), mohou být desítky programátorů, které byste nejraději ani nepotkali. Z toho vyplývá, že zařazení aktuálního adresáře do systémové proměnné PATH závisí na konkrétní situaci.[2]

Skutečný způsob, kterým je nastavena systémová proměnná PATH v mém počítači, je dostatečně ilustrativní. Zde je uvedeno nastavení v souboru .bashrc:

```
export PATH=${PATH}:.:${HOME}/bin:/bin:/usr/bin:/usr/local/bin:/usr
  /bin/X1,1:/usr/TeX/bin
```

Zde jsem využil skutečnosti, že proměnná HOME je nastavena před tím, než příkazový procesor bash přečte můj soubor .bashrc. Systémová proměnná PATH je tedy nastavena prostřednictvím systémové proměnné HOME. Složené závorky („{...}") představují další úroveň uvozovek. Oddělují to, co se vyhodnotí po znaku $, proto bude příkazový procesor moci jednoznačně identifikovat následující část příkazu (tedy „/bin“ v tomto případě). Zde uvádíme další příklad efektu, který vyvolají složené závorky:

```
/home/larry# echo ${HOME}foo
/home/larryfoo
/home/larry#
```

2 Pamatujte si, že program z aktuálního adresáře můžete vždy spustit explicitně, tedy například „./foo“.

Bez uvedení složených závorek se nevypíše nic, protože neexistuje proměnná prostředí HOMEfoo:

```
/home/larry# echo $HOMEfoo
/home/larry#
```

Nyní se podívejme na další zvláštní konstrukci v uvedeném příkazu. Jaký je význam řetězce „$PATH"? Tento řetězec zařadí do proměnné prostředí PATH hodnotu proměnné prostředí PATH dříve definovanou. Kde byla nastavena původní hodnota? Soubor /etc/profile slouží jako jistý typ globálního souboru .bash_profile, kde se nacházejí nastavení společná pro všechny uživatele systému. Jeden soubor s globálním nastavením má jistou výhodu. Má-li například systémový správce přidat do proměnné prostředí PATH důležitou cestu, stačí, když to udělá v souboru s globální platností a nemusí opravovat inicializační soubory všech uživatelů. Proto je proměnná prostředí PATH již nastavena a vy ji můžete použít.

Prostřednictvím jisté proměnné prostředí můžete také specifikovat, jak má vypadat váš příkazový řádek. Tato proměnná prostředí má označení jako PS1. Předpokládejme, že dáváte přednost tomu, aby se stále zobrazovala cesta do aktuálního adresáře. Pak nastavte proměnnou prostředí PS1 takto:

```
export PS1='$PWD#'
```

Jak asi tušíte, jsou zde ve skutečnosti použity dvě proměnné prostředí. První, která se nastavuje, je PS1 a druhou je PWD. Proměnná prostředí PWD může znamenat buď „Print Working Directory", nebo „PATH to Working Directory". Vyhodnocení proměnné prostředí však probíhá uvnitř jednoduchých uvozovek. Jednoduché uvozovky slouží k vyhodnocení vnitřního výrazu. Výsledkem tohoto vyhodnocení je proměnná prostředí PWD. Kdybyste použili výraz export PS1=$PWD, pak by se vám stále zobrazoval ten adresář, který byl aktuální v době vyhodnocení tohoto výrazu. Trochu jsme pohled na vyhodnocování proměnných prostředí zkomplikovali, ale to není tak důležité. Pouze si pamatujte, že budete-li chtít zobrazovat v příkazovém řádku aktuální adresář, budete muset použít jednoduché uvozovky.

Možná, že budete chtít podobný příkazový řádek jako v operačním systému MS-DOS. Pak zadejte export PS1='$PWD>'. Chcete-li mít v příkazovém řádku jméno vašeho systému, zadejte PS1="hostname"'>'.

V posledním příkladu jsme použili nový typ uvozovek, tzv. zpětné uvozovky. Tyto uvozovky ve skutečnosti nic nechrání. Výraz uzavřený ve zpětných uvozovkách se vyhodnotí jako příkaz a výstup se objeví na místě zpětných uvozovek.

Vyzkoušejte si příkazy echo `ls` nebo wc `ls`. Čím více budete seznámeni s příkazovým procesorem, tím užitečnější vám budou uvedené techniky.

V souboru .bashrc je mnoho dalších možností, jak konfigurovat váš příkazový procesor, ale zde není dostatek prostoru všechny prodiskutovat. Další podrobnosti si nastudujte v manuálových stránkách k příkazovému procesoru bash nebo se zeptejte zkušených uživatelů. Dále uvádíme kompletní soubor .bashrc, abyste si jej mohli nastudovat. Představuje typický standard, i když je proměnná prostředí PATH poněkud dlouhá.

```
# some random stuff:
ulimit -c unlimited
export history_control=ignoredups
export PS1='$PWD>'
umask 022
# application-specific PATHs:
```

```
export MANPATH=/usr/local/man:/usr/man
export INFOPATH=/usr/local/info
export PGPPATH=${HOME}/.pgp
# make the main PATH:
homepath=${HOME}:~/bin
stdpath=/bin:/usr/bin:/usr/local/bin:/usr/ucb:/etc:/usr/etc:/urs
  /games
pubpath=/usr/public/bin:/usr/gnusoft/bin:/usr/local/contribs/bin
softpath=/usr/bin/X11:/usr/local/bin/X11:/usr/TeX/bin
export PATH=.:${homepath}:${stdpath}:${pubpath}:${softpath}
# Technically, the curly braces were not necessary, because the
  colons
# were valid delimiters; nevertheless, the curly braces are a good
# habit to get into, and they can't hurt.
# aliases
alias ls="ls -CF"
alias fg1="fg %1"
alias fg2="fg %2"
alias tba="talk sussman@tern.mcs.anl.gov"
alias tko="talk kold@cs.oberlin.edu"
alias tji="talk jimb@totoro.bio.indiana.edu"
alias mroe="more"
alias moer="more"
alias ll="ls -l"
alias la="ls -a"
alias ro="rm *~; rm.*~"
alias rd="rmdir"
alias pu=pushd
alias po=popd
alias ds=dirs
alias to="telnet cs.oberlin.edu"
alias ta="telnet altair.mcs.anl.gov"
alias tg="telnet wobat.gnu.ai.mit.edu"
alias email="emacs -f rmail"
alias ed2="emacs -d floss:0 -fg \"grey95\" -bg \"grey50\""
function gco
{
gcc -o $1 $1.c -g
}
```

Inicializační soubory systému X Window

Většina lidí dává při své práci přednost grafickému uživatelskému prostředí, kterým je v opeačním systému Unix systém X Window. Jestliže jste seznámeni s operačním systémem Macintosh nebo Microsoft Windows, pak vám systém X Window bude připadat velmi známý. Méně známé vám však budou připadat konfigurační možnosti, které systém X Window nabízí.

V případě operačního systému Macintosh nebo Microsoft Windows se konfigurace realizuje přímo v grafickém uživatelském prostředí. Když chcete například změnit barvu pozadí, klepnete myší na novou barvu nějakého speciálního inicializačního grafického programu. V systému X Window se implicitní hodnoty řídí textovými soubory, které můžete přímo editovat – jinými slovy, chcete-li například zadat novou barvu pozadí, musíte její jméno uvést v jistém inicializačním souboru.

Nikdo nepopírá, že inicializační metody v systému X Window jsou poněkud těžkopádnější než u komerčních programů. Domnívám se, že tendence zachovávat textově orientované inicializační metody dokonce i v grafickém uživatelském prostředí tkví v tom, že například systém X Window vytvořila poměrně nesourodá skupina programátorů, kteří až příliš lpí na tradicích dodržovaných v operačních systémech typu Unix. Tyto tendence se mohou v příštích verzích systému X Window změnit (alespoň doufám, že se změní), ale nyní se budeme zabývat inicializací prostřednictvím textových souborů. Tak alespoň máte k dispozici velmi flexibilní a přesnou kontrolu nad konfigurací.

Nejdůležitější konfigurační soubory pro systém X Window jsou tyto:

`.xinitrc`	Skript, který systém X Window spouští ve fázi startování.
`.twmrc`	Soubor, který čte správce oken `twm`.
`.fvwmrc`	Soubor, který čte správce oken `fvwm`.

Všechny uvedené soubory by měly být uloženy ve vašem domovském adresáři.

Soubor `.xinitrc` je jednoduchý skript příkazového procesoru, jenž se automaticky spouští ve fázi startování systému X Window. Může dělat vše, co mohou dělat ostatní skripty, avšak nejvíce ze všeho má smysl jej použít k nastartování různých programů systému X Window a k nastavení parametrů. Posledním příkazem v souboru .xinitrc je obvykle příkaz pro spuštění správce oken, například `/usr/bin/X11/twm`.

Jaký druh konfiguračních nastavení má smysl v souboru `.xinitrc` uvádět? Snad nějaká volání programu `xsetroot`, čímž si můžete nastavit pozadí okna a vlastnosti kurzoru. Dále volání programu `xmodmap`, jenž předá serveru[3] informace o tom, jak má interpretovat signály z vaší klávesnice. Všechny ostatní programy, které se mají spustit pokaždé spolu se systémem X Window (například `xclock`).

Zde uvádím některé řádky z mého souboru `.xinitrc`. Váš konfigurační soubor bude jistě vypadat jinak, proto je považujte za pouhý příklad.

```
#!/bin/sh
# The first line tells the operating system which shell to use in
# interpreting this script. The script itself ought to be marked as
# executable; you can make it so with "chmod +x ~/.xinitrc".
# xmodmap is a program for telling the X server how to interpret your
# keyboard's signals. It is *definitely* worth learning out. You
# can do "man xmodmap", "xmodmap -help", "xmodmap -grammar", and more.
# I don't guarantee that the expressions below will mean anything on
# your system (I don't even guarantee that they mean anything on
# mine):
xmodmap -e 'clear Lock'
xmodmap -e 'keycode 176 = Control_R'
xmodmap -e 'add control = Control_R'
xmodmap -e 'clear Mod2'
xmodmap -e 'add Mod1 = Alt_L Alt_R'
# xset is a program for setting other parameters of the X server:
xset m 3 2 & # mouse parameters
xset s 600 5 & # screen saver prefs
xset s noblank # ditto
xset fp+ /home/larry/x/fonts # for cxterm
```

3 Server představuje hlavní proces systému X Window, tedy program, se kterým musejí všechny ostatní programy běžící pod systémem X Window komunikovat, pokud chtějí využívat grafické prostředí. Tyto ostatní programy se nazývají klienti a systém jako celek bývá označován termínem systém „klient-server“.

```
# To find out more, do "xset -help"
# Tell the X server to superimpose fish.cursor over fish.mask, and use
# the resulting pattern as my mouse cursor:
xsetroot -cursor /home/lab/larry/x/fish.cursor /home/lab/larry/x/fish.mask &
# a pleasing background pattern and color:
xsetroot -bitmap /home/lab/larry/x/pyramid.xbm -bg tan
# todo: xrdb here? What about .Xdefaults file?
# You should do "man xsetroot", or "xsetroot -help" for
# more information on the program above.
# A client program, the imposing circular color-clock by Jim Blandy:
/usr/local/bin/circles
# Maybe you'd like to have a clock on your screen at all time:
/usr/bin/X11/xclock -digital &
# Allow client program running at occs.cs.oberlin.edu to display
# themselves here, do the same thing for juju.mcs.anl.gov:
xhost occs.cs.oberlin.edu
xhost juju.mcs.anl.gov
# You can simply tell the X server to allow clients running on any
# other host (a host being a remote machine) to display here, but this
# is a security hole -- those clients can be run by someone else,
# and watch your keystrokes as you type your password or something!
# However, if you wanted to do it anyway, you could use a "+" to stand
# for all possible hostnames, instead of a specific hostname, like
# this:
# xhost +
# And finally, run the window namager:
/usr/bin/X11/twm
# Some people prefer other window manager. I use twm, but fvwm is
# often distributed with Linux too:
# /usr/bin/X11/fvwm
```

Všimněte si, že některé programy běží na pozadí. Jsou to ty programy, jejichž zadání je ukončeno znakem &. Jiné programy se na pozadí nespouštějí. Rozdíl spočívá v tom, že se startují současně se systémem X Window a běží na pozadí, dokud systém X Window neukončíte. Jiné se vykonají bezprostředně a ihned skončí – jedním z nich je `xsetroot`, který pouze nastaví základní okno a chování kurzoru a pak skončí.

Jakmile se nastartuje správce oken, načte svůj vlastní inicializační soubor. Tento inicializační soubor řídí takové věci, jako je nastavení nabídek, pozice oken, řízení ikon a další. Pokud používáte jako správce oken program `twm`, pak tímto inicializačním souborem je soubor `.twmrc`, který se nachází ve vašem domovském adresáři. Jestliže však používáte `fvwm`, pak je inicializačním souborem soubor `.fvwmrc`. V následujících oddílech se budeme zabývat pouze těmito dvěma, protože se běžně distribuují s operačním systémem Linux.

Konfigurace programu twm

Soubor `.twmrc` není skript příkazového procesoru – je napsán v jazyku speciálně vyvinutém pro program `twm`.[4] Při konfiguraci prostřednictvím souboru `.twmrc` si uživatelé nejraději hrají s nastavením oken (barvy a podobně) a nabídek. Zde uvádíme příklad konfiguračního souboru `.twmrc`:

4 Toto je jedna z odpuzujících vlastností inicializačních souborů: některé z nich mají svůj vlastní příkazový jazyk. To znamená, že uživatel musí být velmi zdatný a rychle se naučit další jazyk. Předpokládám, že takové vymyšlenosti mohly být zajímavé v ranných dobách operačního systému Unix, kdy programátoři stále toužili po něčem novém. Dnes je ale vyčerpávající učit se stále dokola nové a nové syntaxe jazyků, které konec konců slouží jedinému programu.

```
# Set colors for various parts of windows. This has a great
# impact no the "feel" of your environment.
Color
{
BorderColor "OrangeRed"
BorderTitleForeground "Black"
BorderTitleBackground "Black"
TitleForeground "black"
TitleBackground "gold"
MenuForeground "black"
MenuBackground "LightGrey"
MenuTitleForeground "LightGrey"
MenuTitleBackground "LightSlateGrey"
MenuShadowColor "black"
IconForeground "DimGray"
IconBackground "Gold"
IconBorderColor "OrangeRed"
IconManagerForeground "black"
IconManagerBacground "honeydew"
}
# I hope you don't have a monochrome system, but if you do...
Monochrome
{
BorderColor "black"
BorderTitleForeground "black"
BorderTitleBackground "white"
TitleForeground "black"
TitleBackground "white"
}
# I created beifang.bmp with the program "bitmap". Here I tell twm to
# use it as the default highlight pattern on windows' title bars:
Pixmaps
{
TitleHighlight "/home/larry/x/beifang.bmp"
}
# Don't worry about this stuff, it's only for power users :-)
BorderWidth 2
TitleFont "-adobe-new century schoolbook-bold-r-normal--14-140-75-75
  -p-87-iso8859-1"
MenuFont "6x13"
IconFont "lucidasans-italic-14"
ResizeFont "fixed"
Zoom 50
RandomPlacement
# These programs will not get a window titlebar by default:
NoTitle
{
"stamp"
"xload"
"xclock"
"xlogo"
"xbiff"
"xeyes"
```

```
"oclock"
"xoid"
}
# "AutoRaise" means that a window is brought to the front whenever the
# mouse pointer enters it. I find this annoying, so I have turned
# off. As you can see, I inherited my .twmrc from people who also
# did not like autoraise.
AutoRaise
{
"nothing" # I don't like auto-raise # Me either # nor I
}
# Here is where the mouse button functions are defined. Notice the
# pattern: a mouse button pressed on the root window, with no modifier
# key being pressed, always brings up a menu. Other locations usually
# result in window manipulation of some kind, and modifier keys are
# used in conjuction with mouse buttons to get at the more
# sophisticated window manipulations.
#
# You don't have to follow this pattern in your own .twmrc -- it's
# entirely up to you how you arrange your environment.
# Button = KEYS : CONTEXT : FUNCTION
# ----------------------------------
Button1 = : root : f.menu "main"
Button1 = : title : f.raise
Button1 = : frame : f.raise
Button1 = : icon : f.iconify
Button1 = m : window : f.iconify
Button2 = : root : f.menu "stuff"
Button2 = : icon : f.move
Button2 = m : window : f.move
Button2 = : title : f.move
Button2 = : frame : f.move
Button2 = s : frame : f.zoom
Button2 = s : window : f.zoom
Button3 = : root : f.menu "z"
Button3 = : title : f.lower
Button3 = : frame : f.lower
Button3 = : icon : f.raiselower
# You can write your own functions; this one gets used in the menu
# "windowops" near the end of this file:
Function "raise-n-focus"
{
f.raise
f.focus
}
# Okay, below are the actual menus reffered to in the mouse button
# section. Note that many of these menu entries themselves call
# sub-menus. You can have as many levels of menus as you want, but be
# aware that recursive menus don't work. I tried it.
menu "main"
{
"Vanilla" f.title
"Emacs" f.menu "emacs"
```

```
"Logins" f.menu "logins"
"Xlock" f.menu "xlock"
"Misc" f.menu "misc"
}
# This allows me to invoke emacs on several different machines. See
# the section on .rhosts files for more information about how this
# works:
{
"Emacs" f.title
"here" !"/usr/bin/emacs &"
"" f.nop
"phylo" !"rsh phylo \"emacs -d floss:0\" &"
"geta" !"rsh geta \"emacs -d floss:0\" &"
"darwin" !"rsh darwin \"emacs -d floss:0\" &"
"ninja" !"rsh ninja \"emacs -d floss:0\" &"
"indy" !"rsh indy \"emacs -d floss:0\" &"
"oberlin" !"rsh cs.oberlin.edu \"emacs -d floss.life.uiuc.edu:0\" &"
"gnu" !"rsh gate-1.gnu.ai.mit.edu \"emacs -d floss.life.uiuc.edu:0\" &"
}
# This allows me to invoke xterms on several different machines. See
# the section on .rhosts files for more information about how this
# work:
menu "logins"
{
"Logins" f.title
"here" !"/usr/bin/X11/xterm -ls -T `hostname` -n `hostname` &"
"phylo" !"rsh phylo \"xterm -ls -display floss:0 -T phylo\" &"
"geta" !"rsh geta \"xterm -ls -display floss:0 -T geta\" &"
"darwin" !"rsh darwin \"xterm -ls -display floss:0 -T darwin\" &"
"ninja" !"rsh ninja \"xterm -ls -display floss:0 -T ninja\" &"
"indy" !"rsh indy \"xterm -ls -display floss:0 -T indy\" &"
}
# The xlock screensaver, called with various options (each of which
# gives a different pretty picture):
menu "xlock"
{
"Help" !"xlock -mode hop &"
"Qix" !"xlock -mode qix &"
"Flame" !"xlock -mode flame &"
"Worm" !"xlock -mode worm &"
"Swarm" !"xlock -mode swarm &"
"Hop NL" !"xlock -mode hop -nolock &"
"Qix NL" !"xlock -mode qix -nolock &"
"Flame NL" !"xlock -mode flame -nolock &"
"Worm NL" !"xlock -mode worm -nolock &"
"Swarm NL" !"xlock -mode swarm -nolock &"
}
# Miscellaneous programs I run occassionally:
menu "misc"
{
"Xload" !"/usr/bin/X11/xload &"
"XV" !"/usr/bin/X11/xv &"
"Bitmap" !"/usr/bin/X11/bitmap &"
```

```
"Tetris" !"/usr/bin/X11/xtetris &"
"Hextris" !"/usr/bin/X11/xhextris &"
"XRoach" !"/usr/bin/X11/xroach &"
"Analog Clock" !"/usr/bin/X11/xclock -analog &"
"Digital Clock" !"/usr/bin/X11/xclock -digital &"
}
# This is the one I bound to the middle mouse button:
menu "stuff"
{
"Chores" f.title
"Sync" !"/bin/sync"
"Who" !"who | xmessage -file - -columns 80 -lines 24 &"
"Xhost +" !"/usr/bin/X11/xhost + &"
"Rootclear" !"/home/larry/bin/rootclear &"
}
# X functions that are sometimes convenient
menu "x"
{
"X Stuff" f.title
"Xhost +" !"xhost + &"
"Refresh" f.refresh
"Source .twmrc" f.twmrc
"(De)Iconify" f.iconify
"Move Window" f.move
"Resize Window" f.resize
"Destroy Window" f.destroy
"Window Ops" f.menu "windowops"
"" f.nop
"Kill twm" f.quit
}
# This is submenu from above:
menu "windowops"
{
"Window Ops" f.title
"Show Icon Mgr" f.showiconmgr
"Hide Icon Mgr" f.hideiconmgr
"Refresh" f.refresh
"Refresh Window" f.winrefresh
"twm version" f.version
"Focus on Root" f.unfocus
"Source .twmrc" f.twmrc
"Cut File" f.cutfile
"(De)Iconify" f.iconify
"DeIconify" f.deiconify
"Move Window" f.move
"ForceMove Window" f.forcemove
"Resize Window" f.resize
"Raise Window" f.raise
"Lower Window" f.lower
"Raise or Lower" f.raiselower
"Focus on Window" f.focus
"Raise-n-Focus" f.function "raise-n-focus"
"Destroy Window" f.destroy
"Kill twm" f.quit
}
```

Věřte mi, že to není zdaleka nejobsáhlejší soubor `.twmrc`, jaký jsem kdy viděl. Je vysoce pravděpodobné, že nějaký příklad souboru .twmrc bude obsažen ve vaší distribuci systému X Window. Prohledejte adresář `/usr/lib/X11/twm/` nebo `/usr/X11/lib/X11/twm`. Zde byste měli uvedený soubor nalézt.

Často dělají uživatelé chybu a zapomínají uvádět znak `&` na konec příkazů. Pokud systém X Window „zatuhne“ právě když spustíte nějaký příkaz, pak pravděpodobně tkví příčina v tom, že jste zapomněli uvést znak `&`. V takovém případě ukončete systém X Window kombinací kláves Ctrl-Alt-Backspace, opravte soubor `.twmrc` a spusťte systém X Window znovu.

Konfigurace programu fvwm

Pokud používáte jako správce oken program `fvwm`, prohledejte tento adresář:

```
/usr/lib/X11/fvwm/ nebo /usr/X11/lib/X11/fvwm.
```

Zde najdete nějaké příklady konfiguračních souborů.

Poznámka: O programu `fvwm` nic nevím. Asi bych byl schopen něco vyčíst z příkladů konfiguračních souborů, ale zůstal bych pouze čtenářem a těžko bych dokázal něco vysvětlovat. Zájemcům o program fvwm a jeho konfiguraci doporučuji přečíst si příslušné manuálové stránky a prostudovat příklady konfiguračních souborů nacházejících se ve výše zmíněných adresářích.

Ostatní inicializační soubory

Za zmínku stojí následující inicializační soubory:

`.emacs`	Inicializační soubor editoru Emacs. Editor jej čte ve fázi startování.
`.netrc`	Soubor obsahující implicitní jména a hesla pro `ftp`.
`.rhosts`	Zpřístupňuje váš účet vzdáleným systémům.
`.forward`	Inicializační soubor pro automatické přesměrování elektronické pošty.

Inicializační soubor pro editor Emacs

Pokud používáte editor Emacs jako primární editor, pak má pro vás soubor `.emacs` mimořádný význam. Podrobně jsme jej popsali v kapitole 8.

Implicitní nastavení pro FTP

V souboru `.netrc` můžete mít uložena některá implicitní nastavení pro `ftp`. Následující řádky obsahují příklad takového souboru:

```
machine floss.life.uiuc.edu login larry password fishSticks
machine darwin.life.uiuc.edu login larry password fishSticks
machine geta.life.uiuc.edu login larry password fishSticks
machine phylo.life.uiuc.edu login larry password fishSticks
machine ninja.life.uiuc.edu login larry password fishSticks
machine indy.life.uiuc.edu login larry password fishSticks


machine clone.mcs.anl.gov login fogel password doorm@
machine osprey.mcs.anl.gov login fogel password doorm@
machine tern.mcs.anl.gov login fogel password doorm@
machine altair.mcs.anl.gov login fogel password doorm@
machine dalek.mcs.anl.gov login fogel password doorm@
```

```
machine juju.mcs.anl.gov login fogel password doorm@
machine sunsite.unc.edu login anonymous password larry@cs.oberlin.edu
```

Každý řádek souboru `.netrc` specifikuje jméno počítače, přihlašovací jméno, které se má použít jako implicitní pro tento počítač, a heslo. Uvedená nastavení vám ušetří velké množství času, protože jinak byste při přihlašování se k jednotlivým serverům `ftp` museli pokaždé zadávat dlouhé identifikační řetězce. Pokud se budete přihlašovat k serveru ftp, jehož jméno je uvedeno v souboru `.netrc`, pokusí se program `ftp` aplikovat uživatelské jméno a heslo z tohoto souboru.

Při spouštění programu `ftp` můžete pomocí parametru -n specifikovat, že nechcete použít implicitní nastavení ze souboru .netrc. Příkaz pak bude mít tvar „`ftp -n`“.

Musíte se ujistit, že soubor `.netrc` jste schopni číst pouze vy. K nastavení příslušných přístupových práv použijte program `chmod`. Kdyby soubor `.netrc` mohli číst jiní uživatelé, pak by mohli odhalit vaše hesla platná pro servery ftp uvedené v tomto souboru.

Takový případ by představoval značnou „bezpečnostní díru“. Naštěstí program `ftp` a ostatní programy, které čtou soubor `.netrc`, odmítnou pokračovat ve své činnosti, pokud zjistí, že přístupová práva k tomuto souboru nejsou v pořádku.

Další informace týkající se souboru .netrc si najděte v manuálových stránkách prostřednictvím příkazu „`man .netrc`“ nebo „`man ftp`“.

Povolení snadného vzdáleného přístupu k vašemu účtu*

Jestliže se ve vašem domovském adresáři nachází soubor `.rhosts`, pak lze ze vzdálených počítačů spouštět aplikace na vašem počítači. Uveďme si příklad. Předpokládejme, že jste přihlášeni na počítači `cs.oberlin.edu`. Na počítači `floss.life.uiuc.edu` je správně konfigurován soubor `.rhosts`.

Pak ze svého počítače můžete na počítači `floss.life.uiuc.edu` spustit aplikaci, jejíž výstup bude přesměrován na vaši obrazovku, a dokonce se před tím nebudete muset přihlašovat a zadávat heslo.

Soubor `.rhosts` může vypadat takto:

```
frobnozz.cs.knowledge.edu jsmith
aphrodite.classics.havhaahd.edu wphilps
frobbo.hoola.com trixie
```

Formát souboru je velmi jednoduchý: jméno počítače následované jménem uživatele. Předpokládejme nyní, že uvedený příklad je mým skutečným souborem `.rhosts` na vzdáleném počítači `floss.life.uiuc.edu`. To by znamenalo, že bych mohl spustit program na počítači floss a výstup by mohl být přesměrován na kterýkoliv počítač uvedený v tomto souboru, pokud bych byl přihlášen jako odpovídající uživatel.

Přesný mechanismus, prostřednictvím kterého uživatel uskutečňuje spouštění vzdáleného programu, je realizován programem `rsh`, jehož název je zkratkou pro „remote shell“, tedy „vzdálený příkazový procesor“. Ten nastartuje příkazový procesor na vzdáleném počítači a spustí specifikovaný program. Uveďme si příklad:

```
frobbo$ whoami
trixie
frobbo$ rsh floss.life.uiuc.edu "ls ~"
```

---

* Poznámka korektora: Používání programu `rsh` a `rlogin` je v současné době nahrazeno podobným programem `ssh`, který je bezpečnější.

```
foo.txt mbox url.ps snax.txt
frobbo$ rsh floss.life.uiuc.edu "more ~/snax.txt"
[nyní se zobrazí stránky souboru snax.txt]
```

Uživatel trixie na počítači `floss.life.uiuc.edu`, který má soubor `.rhosts` uvedený v předcházejícím příkladě, umožňuje uživateli trixie na počítači `frobbo.hoola.com` spouštět programy z počítače floss.

Soubor `.rhosts` bude pracovat správně, i když nemáte na všech počítačích stejné uživatelské jméno. Chcete-li „sdělit" vzdálenému počítači, jaké jméno uživatele chcete použít k přihlášení se, použijte při zadání příkazu `rsh` volbu „`-l`". Pokud takový uživatel na vzdáleném počítači existuje a pokud zde existuje soubor `.rhosts` obsahující jméno vašeho (t.j. lokálního) počítače a jméno uživatele, pak se příkaz `rsh` provede úspěšně.

```
frobbo$ whoami
trixie
frobbo$ rsh -l larry floss.life.uiuc.edu "ls ~"
[zde se objeví seznam souborů mého adresáře na počítači floss]
```

Uvedený příklad bude fungovat, pokud má uživatel `larry` na počítači `floss.life.uiuc.edu` takový soubor `.rhosts`, jenž umožňuje uživateli `trixie` z počítače `frobbo.hoola.com` spouštět programy. Není relevantní, zda se jedná nebo nejedná o tutéž osobu. Důležitá jsou pouze jména uživatelů, jména počítačů a záznamy v souboru `.rhosts`.

V souboru `.rhosts` mohou být uvedeny i jiné kombinace – například jméno uživatele následující za jménem vzdáleného počítače se může vynechat a tak umožnit kterémukoliv uživateli na vzdáleném počítači spouštět programy na vašem počítači. To je však spojeno s jistými riziky. Nepovolaná osoba by mohla například zrušit vaše soubory. Pokud se rozhodnete pro takto organizovaný soubor `.rhosts`, pak byste se měli ujistit, že právo číst soubor `.rhosts` máte pouze vy.

Přesměrování elektronické pošty

Kromě jiných souborů můžete také mít soubor `.forward`, který nelze přímo nazvat „inicializačním" souborem. Pokud tento soubor obsahuje adresu elektronické pošty, pak veškeré zprávy elektronické pošty budou odesílány na tuto adresu. Soubor je užitečný v případě, kdy máte účet na několika různých systémech, ale elektronickou poštu chcete číst pouze na jednom.

Na vašem počítači se mohou nacházet i jiné inicializační soubory. Jejich počet se liší podle operačního systému, jenž používáte, a také podle programů, které máte nainstalovány. Jeden ze způsobů, jak získat více informací o inicializačních souborech, spočívá v tom, že si prostudujete soubory ve vašem domovském adresáři začínající tečkou. Není sice garantováno, že všechny takové soubory jsou inicializační, ale převážná většina z nich určitě bude.

Kde si můžete prohlédnout některé příklady

Jestliže na svém počítači máte instalován operační systém Linux a jestliže máte přístup do sítě Internet, pak se můžete přihlásit prostřednictvím služby telnet do systému `floss.life.uiuc.edu`. Přihlašte se jako „guest" a heslo uveďte jako „explorer". Připravili jsme pro vás spoustu příkladů, z nichž většina je uložena v adresáři `/home/kfogel`. Tyto příklady si můžete zkopírovat a prostudovat. Buďte ale opatrní. Počítač floss nepředstavuje zcela zabezpečený systém a když se budete dostatečně snažit, podaří se vám získat přístupová práva uživatele root. Dali jsme přednost důvěře před neustálou ostražitostí a doufáme, že toho nikdo nezneužije.*

* Poznámka korektora: Autor knihy to zřejmě myslel jako vtip.

KAPITOLA 10

Komunikace s ostatními systémy

Moderní operační systémy typu Unix jsou dobře přizpůsobeny ke komunikaci s ostatními počítači, což znamená, že jsou vybaveny prostředky pro práci v síti. Dva různé počítače s operačním systémem Unix si mohou vyměňovat informace mnoha způsoby. Tato kapitola je věnována metodám, pomocí kterých budete moci komunikovat prostřednictvím sítě s ostatními počítači.

Budeme se zabývat elektronickou poštou, zájmovými skupinami a některými základními programy pro komunikaci.

Elektronická pošta

Mezi nejpopulárnější vlastnosti operačního systému Unix patří možnost odesílat a přijímat zprávy elektronické pošty. Máte-li k dispozici elektronickou poštu, nepotřebujete papír, inkoust a pero, obálky, známky a nesrovnatelně pomalejší poštovní službu.

Odesílání elektronické pošty

Potřebujete-li odeslat zprávu elektronickou poštou, stačí zadat příkaz `mail username` a pak zapsat vaši zprávu.

Předpokládejme například, že chcete odeslat zprávu elektronickou poštou uživateli jménem sam:

```
/home/larry# mail sam
Subject: The user documentation
Just testin out the mail system.
EOT
/home/larry#
```

Program `mail` je velmi jednoduchý. Podobně jako program `cat` čte text ze standardního vstupu řádek po řádku, dokud nenarazí na znak „konec textu“ jenž je reprezentován klávesou Ctrl-D. To znamená, že po dokončení textu zprávy stisknete Enter a pak Ctrl-D.

Příkaz `mail` představuje nejrychlejší způsob, jak odeslat zprávu elektronické pošty, a je užitečný zejména ve spojení s prostředky pro přesměrování vstupu/výstupu či rourami. Jestliže chcete například odeslat soubor `report1` uživateli jménem „Sam“, můžete použít příkaz `mail sam < report1` nebo dokonce „`sort report1 | mail sam`“.

Používání příkazu `mail` je však spojeno s velkou nevýhodou. Jestliže uděláte v nějakém řádku chybu, pak ji již nemůžete opravit. Proto vám doporučuji (pokud nepoužijete možnost s přesměrováním vstupu/výstupu) k odesílání elektronické pošty používat editor Emacs. Postup jsme popsali v oddílu Pracovní módy editoru Emacs.

Čtení zpráv elektronické pošty

`mail [`*`user`*`]`

Program `mail` poskytuje poněkud těžkopádnou možnost číst zprávy elektronické pošty. Zadáte-li příkaz `mail` bez jakýchkoliv parametrů, objeví se vám následující hlášení:

```
/home/larry# mail
No mail for larry
/home/larry#
```

Předpokládejme, že chcete odeslat zprávu elektronickou poštou sami sobě, abyste si mohli vyzkoušet způsoby jejího čtení:

```
/home/larry# mail larry
Subject: Frogs!
and toads!
EOT
/home/larry# echo "snakes" | mail larry
/home/larry# mail
Mail version 5.5. 6/1/90 Type ? for help.
"/usr/spool/mail/larry" 2 messages new
>N 1 larry Tue Aug 30 18:11 10/211 "Frogs!"
N 2 larry Tue Aug 30 18:12 10/211
&
```

Příkazový řádek uvnitř programu pro čtení elektronické pošty je uvozen znakem ampersand („&"). Umožňuje specifikovat spoustu jednoduchých příkazů a po zadání znaku ? a Enter zobrazuje stručnou nápovědu.

Základními příkazy pro práci s programem mail jsou:

`t` *`message-list`*	Na obrazovce se zobrazí zprávy uvedené v seznamu *`message-list`*.
`d` *`message-list`*	Zprávy uvedené v seznamu *`message-list`* se zruší.
`s` *`message-list file`*	Zprávy uvedené v seznamu *`message-list`* se uloží do souboru *`file`*.
`r` *`message-list`*	Odpověď na zprávy – program `mail` zahájí proces sestavování nových zpráv, které budou odeslány každému, kdo vám odeslal zprávu uvedenou v seznamu *`message-list`*.
`q`	Program `mail` se ukončí a uloží každou zprávu, která nebyla zrušena příkazem `d` do souboru mbox ve vašem domovském adresáři.

Jak vypadá seznam *`message-list`*? Skládá se ze seznamu čísel oddělených mezerami, nebo může být vyjádřen ve formě intervalu, tedy například 2-4 (což znamená „2 3 4"). Také můžete uvést uživatelské jméno odesílatele. Například `t sam` zobrazí všechny zprávy odeslané uživatelem sam. Jestliže se seznam *`message-list`* neuvede, zobrazí se vždy poslední zpráva.

Se čtením zpráv elektronické pošty je spojeno několik problémů. Především platí, že pokud má zpráva více řádků, než se vejde na obrazovku, program mail se nezastaví! Takou zprávu budete muset uložit do souboru a pak si ji prohlédnout prostřednictvím příkazu `more`. Dále program mail nemá dobré prostředky pro čtení starých zpráv, tedy zpráv uložených do souborů.

Editor Emacs má prostředek pro čtení zpráv elektronické pošty, jenž se jmenuje `rmail`. V této knize se však programem rmail nebudeme zabývat. V operačním systému Linux jsou navíc k dispozici další programy pro čtení elektronické pošty, jako je `elm` nebo `pine`.*

Jak vyhledat uživatele sítě

Příkaz finger

Příkaz `finger` vám umožní získat informace o ostatních uživatelích vašeho systému nebo o uživatelích sítě Internet. Jméno příkazu nepochybně vzniklo jako zkratka s reklamním podtextem ve firmě AT&T.

`finger [-slpm] [`*`user`*`][`*`@machine`*`]`

Volitelné parametry v příkazu `finger` mohou být mírně matoucí. Pomocí příkazu `finger` můžete získat informace o lokálním uživateli (například „sam“), o jiném počítači (například „@lionsden“), informace o uživateli vzdáleného počítače (například „sam@liondsen“) nebo informace o lokálním počítači (neuvede se žádný parametr).

Příkaz finger má další zajímavou vlastnost. Pokud se pokusíte získat informace o uživateli, jehož jméno přesně neznáte, pokusí se příkaz `finger` najít jméno sám (zkouší různé kombinace založené na původně zadaném jménu). To znamená, že když například zadám příkaz `finger Greenfield`, obdržím zprávu, že účet `sam` existuje pro jméno `Sam Greenfield`.

```
/home/larry# finger sam
Login: sam Name: Sam Greenfield
Directory: /home/sam Shell: /bin/tcsh
Last login Sun Dec 25 14:47 (EST) on tty2
No Plan.
/home/larry# finger greenfie@gauss.rutgers.edu
[gauss.rudgers.edu]
Login name: greenfie In real life: Greenfie
Directory: /gauss/u1/greefie Shell: /bin/tcsh
On since Dec 25 15:19:41 on ttyp0 from tiptop-slip-6439
13 minutes Idle Time
No unread mail
Project: You must be joking!
No Plan.
/home/larry# finger
Login Name Tty Idle Login Time Office Office Phone
larry Larry Greenfield 1 3:51 Dec 25 12:50
larry Larry Greenfield p0 Dec 25 12:51
/home/larry#
```

* Poznámka korektora: Samozřejmě můžete používat i jiné poštovní klienty, např. i Netscape Communicator.

Použijete-li u příkazu `finger` volbu `-s`, pak se vám vždy zobrazí stručný výpis (stejný, který obdržíte, když program `finger` použijete k získání informací o počítači) a pokud použijete volbu `-l`, zobrazí se vám vždy kompletní výpis (i tehdy, když program `finger` použijete k získání informací o počítači). Po zadání volby `-p` se nezobrazí informace ze souborů `.forward`, `.plan` a `.project`. Zadáte-li volbu `-m` a žádáte-li pouze informace o uživateli, pak se vám zobrazí pouze přihlašovací jméno.

Soubory .plan a .project

Nyní bychom měli vysvětlit, jaký je význam souborů `.plan` a `.project`. Jedná se o soubory, které jsou uloženy v domovském adresáři uživatele a jejichž obsah se zobrazí pokaždé, když je na daného uživatele aplikován program `finger`. Soubory `.plan` a `.project` si můžete vytvořit sami – jediné omezení spočívá v tom, že ze souboru .project se zobrazuje pouze první řádek.

Dále platí, že každý, kdo chce aplikovat program `finger` musí mít možnost procházet vaším domovským adresářem (`chmod a+x ~/`) a každý musí být schopen číst soubory `.plan` a `.project` (`chmod a+r ~/.plan ~/.project`).

Používání systémů vzdálenými počítači

`telnet` *`vzdálený systém`*

Hlavní prostředek pro využívání vzdálených systémů založených na operačním systému Unix představuje program Telnet. (V současné době se spíše používá program `ssh`, který na rozdíl od příkazu `telnet` umožňuje šifrovanou komunikaci se vzdáleným systémem.) Používání programu telnet je velmi jednoduché:

```
/home/larry# telnet lionsden
Trying 128.2.36.41...
Connected to lionsden
Escape character is '^]'.
lionsden login:
```

Jak vidíte z následujícího příkladu, po zadání příkazu `telnet` budete vyzváni k přihlášení se ke vzdálenému systému. Uživatelské jméno lze zadat jakékoliv (ovšem heslo musíte znát správné) a pak je vám vzdálený systém k dispozici téměř stejně jako váš lokální systém.

Normální způsob ukončení programu `Telnet` spočívá v odhlášení se, ale je také možnost zadat znak Escape, kterým je zpravidla Ctrl-]. Tak obdržíte nový příkazový řádek uvozený řetězcem `telnet>`. Nyní stačí napsat `quit` a pak stisknout klávesu Enter – spojení se přeruší a program `Telnet` se ukončí. Pokud si to rozmyslíte a nechcete relaci ukončit, stiskněte pouze klávesu Enter.

Pokud jste uživateli systému X Window, pak si pro komunikaci se vzdáleným počítačem otevřete nové terminálové okno – například prostřednictvím příkazu „`xterm -title "lionsden" -e telnet lionsden &`“. Uvedený příkaz otevře nové terminálové okno, ve kterém automaticky poběží program `Telnet`. Jestliže takový příkaz budete používat častěji, pak doporučujeme, abyste si pro něj vytvořili alias.

Přenášení souborů

`ftp` *`vzdálený systém`*

Normální způsob přenášení souborů zprostředkovává v operačním systému Unix program `ftp`, což je zkratka pro „**file transfer protocol**“. Po zadání příkazu `ftp` budete vyzváni, abyste se přihlásili ke vzdálenému systému téměř stejným způsobem, jako v případě programu `telnet`. Pak se vám zobrazí speciální příkazový řádek.

Nyní máte k dispozici několik příkazů běžných v operačním systému Unix, jež můžete aplikovat v příkazovém řádku programu `ftp`. Například příkaz `cd` i zde slouží k přepínání se mezi adresáři a příkaz `ls` slouží k zobrazení seznamu souborů uložených v aktuálním adresáři.

Navíc máte k dispozici dva důležité příkazy: `get` a `put`. Příkaz `get` je určen k přenosu souborů ze vzdáleného počítače do vašeho lokálního počítače a příkaz `put` slouží k opačnému úkolu. Oba příkazy jako implicitní používají váš domovský adresář a lokální adresář na vzdáleném počítači (který můžete měnit prostřednictvím příkazu `cd`).

S používáním programu `ftp` je spojen jeden problém. Spočívá v rozlišení mezi znakovými a binárními soubory. Protokol pro přenos dat programem `ftp` je velmi starý a implicitně předpokládá, že přenášené soubory jsou textové. Aplikuje-li se tento implicitní přenos na binární soubory, pak budou s největší pravděpodobností po přenosu poškozeny. Před přenosem binárního souboru proto používejte příkaz `binary`.

Program `ftp` ukončíte příkazem `bye`.

Putování po stránkách WWW

WWW, neboli World Wide Web (doslova „celosvětová pavučina“) zřejmě představuje nejpopulárnější způsob využívání Internetu. Skládá se ze stránek, z nichž každá je spojena s tzv. lokátorem URL (**uniform resource locator**). Lokátory URL jsou komické řetězce následujícího tvaru: `http://www.rutgers.edu/`. Stránky jsou vytvořeny v jazyku HTML (**hypertext markup language**).

Jazyk HTML umožňuje autorům stránek WWW začlenit do dokumentů odkazy na kterékoliv jiné stránky WWW (nebo obrázky) umístěné kdekoliv jinde v celosvětovém systému WWW. Když uživatel čte nějaký dokument, pak se pouhým klepnutím na odkaz (prezentovaný klíčovým slovem nebo tlačítkem) přenese na jinou stránku WWW, která může být uložena v počítači na druhém konci světa.

`netscape [`*`url`*`]`

Nejpopulárnějším programem pro prohlížení stránek WWW je v operačním systému Linux program Netscape od firmy Netscape Corporation. Program Netscape může běžet pouze pod systémem X Window. *Pozn.: V poslední době jsou populární i další prohlížeče jako Mozilla, Kongueror, Galeon a další.

Program Netscape je velmi jednoduchý, pokud jde o ovládání. Je založen na knihovně Motif a připomíná program napsaný pro Microsoft Windows (samozřejmě, že pro Microsoft Windows také existuje verze programu Netscape). Odkazy na další stránky WWW jsou v programu Netscape zobrazeny modře. Stačí na odkaz klepnout levým tlačítkem myši a vzápětí se vám zobrazí nová stránka WWW.

Operační systém Linux podporuje i jiné programy pro prohlížení stránek WWW, například program lynx, což je textově orientovaný program, proto není schopen zobrazovat obrázky a jiné grafické prvky dokumentů WWW. Je ovšem schopen pracovat pod systémem X Window.

`lynx [`*`url`*`]`

Naučit se pracovat s programem `lynx` je poněkud obtížnější, než naučit se pracovat s programem `Netscape`. Při práci si budete muset zvyknout na používání kurzorových kláves. Klávesy „šipka nahoru“ a „šipka dolů“ jsou určeny k přepínání mezi odkazy na dané stránce WWW, klávesa „šipka doprava“ zobrazí novou stránku (na kterou odkazuje zvýrazněný odkaz) a klávesa „šipka doleva“ je určena k zobrazení předcházející stránky. K ukončení programu `lynx` použijte klávesu Q. Program `lynx` má mnohem více příkazů, jejichž kompletní popis najdete v manuálových stránkách.

KAPITOLA 11

Zábavné příkazy

Většina lidí, která má co do činění s operačním systémem Unix, asi nebude souhlasit s titulem této kapitoly. Někteří se dokonce rozčílí, protože si u každého příkazu musejí pamatovat až desítky parametrů a najednou se jim někdo snaží namluvit, že by používání takových příkazů mohlo být zábavné. V nadpisu této kapitoly se spíš odráží až sarkastický humor autorů operačního systému Unix. Dále uvedené příkazy totiž nemají ekvivalentní příkazy v operačním systému MS-DOS. Přitom jsou velmi výkonné, a když je zatvrzelí příznivci operačního systému MS-DOS chtějí používat, musejí si je koupit (speciální verze těchto příkazů pro MS-DOS byly dodatečně vytvořeny). Protože operační systém Unix odjakživa soupeří s operačními systémy od firmy Microsoft, jsou touto skutečností příznivci operačního systému Unix pobaveni.

V této kapitole se budeme zabývat příkazy `find` (příkaz pro vyhledávání skupin souborů v adresářové struktuře), `tar` (příkaz pro archivaci souborů nebo celých adresářů), `dd` (příkaz pro kopírování) a `sort` (příkaz pro třídění souborů). Předem je nutno upozornit na skutečnost, že tyto příkazy nejsou standardizovány. Zaměříme se na popis verzí náležejících do projektu GNU, protože právě tyto verze jsou implementovány v operačním systému Linux a právě tyto verze pravděpodobně budou (tak jako ostatní programy vytvořené v rámci projektu GNU) v blízké budoucnosti představovat standard. Používáte-li jiný operační systém typu Unix, pak nezapomeňte konfrontovat příslušné manuálové stránky.

Příkaz find

Všeobecné poznámky

Mezi doposud probranými příkazy jsou takové, které umožňují rekurzivní procházení stromovou strukturou adresářů. Příkladem jsou příkazy `ls -R` nebo `rm -R`. Příkaz `find` je také rekurzivní. Kdykoliv byste měli něco dělat s jistým typem souborů v adresáři a ve všech jeho podadresářích, vzpomeňte si na příkaz `find`. V jistém smyslu platí, že vyhledávání souborů příkazem `find` představuje spíše vedlejší efekt.

Základní struktura příkazu `find` je následující:

`find` *`cesta`* `[...]` *`výraz`* `[...]`

Uvedená syntaxe platí pouze pro verzi GNU. Ostatní verze neumožňují specifikovat více než jednu cestu (path), což z praktického hlediska není tak důležité. Hrubé vysvětlení funkce příkazu je následující: prostřednictvím prvního parametru zadáte, odkud má prohledávání začít (parametr Cesta; u verze GNU nemusíte tento parametr vůbec uvádět a prohledávání pak začne od aktuálního adresáře), a druhý parametr (Výraz) určuje typ vyhledávání.

Standardní chování příkazu `find` je poněkud lstivé a stojí za to tomuto faktu věnovat trochu pozornosti. Předpokládejme, že vaším domovským adresářem je adresář garbage a že obsahuje soubor foobar. Řekněme, že náhodou napíšete příkaz `find . -name foobar`. Tento příkaz by měl podle všech předpokladů vyhledat soubory nazvané `foobar` – avšak nic se nestane. Potíž je v tom, že program `find` je tzv. tichý (silent) příkaz. Pouze vrátí 0 (ať už nějaký soubor najde nebo ne), nebo vrátí záporné číslo, pokud nastal nějaký problém. Uvedený případ nenastane, pokud používáte operační systém Linux, ale je dobré jej mít na paměti.

Výrazy

Výraz může být rozdělen do čtyř různých skupin klíčových slov: *volby*, *testy*, *akce* a *operátory*. Každé klíčové slovo může vracet hodnotu true nebo false a přitom může produkovat nějaký vedlejší efekt. Rozdíl mezi skupinami klíčových slov popisuje následující seznam:

volby celkově ovlivňují operaci vyhledávání a netýkají se zpracování souboru. Příkladem volby je `-follow`, která „sdělí" příkazu `find`, aby procházel symbolické odkazy. Volby vždy vracejí hodnotu true.

testy jsou skutečnými testy. Například test `-empty` ověřuje, zda je soubor prázdný. Testy mohou vracet hodnotu true nebo false.

akce produkují jako vedlejší efekt jméno programu. Také mohou vracet hodnotu true nebo false.

operátory ve skutečnosti nevracejí žádnou hodnotu (dle konvence vracejí hodnotu true) a používají se ke skládání výrazů. Příkladem je operátor `-or`, jenž jako výsledek produkuje logickou operaci OR nad operandy, kterými jsou dva výrazy. Jsou-li vedle sebe uvedeny dva výrazy bez operátoru, pak se implicitně aplikuje operátor `-and`.

Poznamenejme, že program `find` spoléhá na syntaktickou analýzu příkazu, kterou realizuje příkazový procesor. To znamená, že všechna klíčová slova musejí být oddělena nezobrazitelnými znaky a zejména platí, že spousta znaků musí být uvozena znaky Escape – jinak by je příkazový procesor nemohl správně interpretovat. Jako znaky Escape mohou být použita obrácená lomítka, jednoduché nebo dvojité uvozovky. V později uvedených příkladech se jednoznaková klíčová slova uvozují obráceným lomítkem, protože je to nejjednodušší.

Volby

V následujícím seznamu uvádíme přehled všech voleb, které je schopen příkaz `find` akceptovat (připomínáme, že se jedná o verzi GNU). Nezapomeňte, že volby vždy vracejí hodnotu true.

- `-daystart`

  Volba `-daystart` měří dobu, která uplyne od poslední půlnoci. Počítačový nadšenec asi nemůže pochopit, proč má nějaký program takovou volbu, ale programátor, který pracuje od osmi do pěti, ji ocení.

- `-depth`

  Tato volba zajistí, že příkaz `find` bude zpracovávat obsah každého podadresáře před zpracováním adresáře samotného. Abych řekl pravdu, neumím si představit užitečné uplatnění této volby, kromě případu, kdy se emuluje příkaz `rm -F` (podadresáře nemůžete zrušit, dokud obsahují nějaké soubory).

- `-noleaf`

  Volba `-noleaf` vypíná optimalizaci, která indikuje, že adresář obsahuje o dva podadresáře méně, než by měl obsahovat. Kdyby byl svět dokonalý, pak by bylo možné odkazovat se na všechny adresáře z prostředí každého jejich podadresáře (pomocí volby `..`), pomocí odkazu `.` uvnitř samotného adresáře a prostřednictvím jeho skutečného jména z jeho nadřazeného adresáře.

  To znamená, že na každý adresář musejí existovat alespoň dva odkazy (jeden z adresáře samotného a jeden z adresáře nadřazeného) a případně další odkazy ze všech podadresářů. V praxi se však může stát, že symbolické odkazy a distribuované souborové systémy mohou toto pravidlo porušit.[1]

- `-maxdepth` *`levels`*, `-mindepth` *`levels`*

  Parametry `levels` jsou nezáporná čísla, jež udávají maximální a minimální úroveň podadresářů, které má příkaz `find` prohledávat. Uveďme příklady: `-maxdepth 0` indikuje, že příkaz `find` má být realizován pouze pro argumenty uvedené v příkazovém řádku a že se žádné podadresáře prohledávat nemají. `-mindepth 1` zakazuje zpracování příkazu pro argumenty uvedené v příkazovém řádku, zatímco ostatní soubory v podadresářích budou zpracovány.

- `-version`

  Po zadání parametru `-version` se pouze vypíší informace o verzi programu find.

- `-xdev`

  Parametr `-xdev` má poněkud zavádějící označení. Pro program `find` předává informaci o tom, že se dané zařízení (tedy souborový systém) nemá prohledávat. Volba `-xdev` je velmi užitečná v případě, kdy se má vyhledávat něco v hlavním souborovém systému. Hlavní souborový systém většinou obsazuje malou diskovou oblast. Kdyby se použil příkaz `find /` bez parametru `-xdev`, pak by se prohledávala celá adresářová struktura!

Testy

První dva testy jsou velmi jednoduché: `-false` vždy vrací hodnotu false a `-true` vždy vrací hodnotu true. K dalším testům, které nevyžadují specifikaci hodnoty, patří test `-empty` (vrací hodnotu true, pokud je daný soubor prázdný) a dále dvojice `-nouser / -nogroup`, kdy test vrací hodnotu true, právě když se v souboru `/etc/passwd` nebo `/etc/group` nevyskytuje záznam identifikující vlastníka souboru jako uživatele nebo skupinu. Jedná se o postižení situace, kdy je v systému zrušen účet uživatele, ale soubory jím vlastněné jsou stále uloženy někde v souborovém systému. Podle Murphyho zákonů jsou takové soubory neobyčejně velké.

Samozřejmě je možné vyhledávat soubory vlastněné jistým uživatelem nebo jistou skupinou. Odpovídající testy jsou `-uid nn` a `-gid nn`. Naneštěstí nelze přímo zadat uživatelské jméno, ale musí se vždy uvést jeho identifikační číslo `nn`.

Je povoleno použít zápis identifikačního čísla ve tvaru `+nn`, což znamená „ostře větší než“ nebo `-nn`, což znamená „ostře menší než“. Ve spojení s identifikačními čísly uživatelů se tato možnost jeví jako hloupá, ale v souvislosti s ostatními testy může být užitečná.

Další užitečnou volbou je volba `-type c`, která vrací hodnotu true, pokud je soubor typu `c`. Mnemotechnické zkratky jsou stejné, jako v případě příkazu `ls`: **b** pro binární soubor, **c** pro znakový soubor, **d** pro adresáře, **p** pro pojmenované roury, **l** pro symbolické odkazy a **s** pro sokety (sockets). Obyčejné soubory jsou specifikovány prostřednictvím zkratky **f**. K testu `-type` se vztahuje

1 Distribuované souborové systémy umožňují, aby se soubory lokalizované někde jinde jevily jako lokální.

test `-xtype`, který je podobný, ale chová se jinak v případě symbolických odkazů – pokud není zadána volba `-follow`, ověřuje se místo vlastního symbolického odkazu soubor, na který odkaz odkazuje.

Testy `-inum nn` a `-links nn` ověřují, zda je číslo i-uzlu příslušné k danému souboru nn nebo zda má soubor nn symbolických odkazů. Test `-size nn` má hodnotu true, jestliže je pro soubor alokováno nn bloků po 512 bajtech. Dnes již však neplatí, že se velikost souboru vždy měří (například po zadání příkazu `ls -s`) v blocích po 512 bajtech – Linux například používá bloky po 1024 bajtech. Proto je možné číslo `nn` doplnit znakem `b` (pak se měří velikost v bajtech) nebo k (pak se měří velikost v kilobajtech).

Bity specifikující přístupová práva se testují pomocí testu `-perm` *`mode`*. Pokud argumentu *`mode`* nepředchází žádné znaménko, pak bity specifikující přístupová práva musejí přesně souhlasit. Pokud předchází znaménko `-`, pak musejí být nastavena příslušná přístupová práva, ale o ostatních se nic nepředpokládá. Pokud předchází znak +, pak je podmínka splněna, právě když je kterýkoliv z bitů nastaven. Důležité je, že hodnota mode musí být uvedena buď symbolicky, nebo jako oktalové číslo, tak jako v případě příkazu `chmod`.

Následující skupina testů se vztahuje k času, kdy byl soubor naposledy použit. Takové testy jsou zejména užitečné tehdy, když dojde k zaplnění diskového prostoru a uživatel potřebuje vyhledat staré soubory, které lze zrušit. Staré a zapomenuté soubory se těžko hledají a příkaz `find` vám dává naději, že se vám je podaří nalézt. Test `-atime nn` vrací hodnotu true, pokud byl daný soubor zpřístupněn před `nn` dny, `-ctime nn` vrací hodnotu true, když byly atributy přístupových práv daného souboru změněny před `nn` dny (například příkazem `chmod`). Test `-mtime nn` vrací hodnotu true, když byl soubor naposledy modifikován před `nn` dny. Užitečným bude zřejmě test `-newer file`, který vrací hodnotu true, když byl uvažovaný soubor modifikován později než soubor uvedený jako parametr `file`. GNU-verze příkazu `find` také akceptuje testy `-anewer` a `-cnewer`, které se chovají podobně jako test `-newer`, a dále testy `-amin`, `-cmin` a `-mmin`, které pracují s časem uvedeným v minutách.

V neposlední řadě se musíme zmínit o testu `-name pattern`. Tento test vrací hodnotu true, pokud jméno souboru vyhovuje šabloně uvedené prostřednictvím parametru `pattern`. Pro šablonu platí prakticky stejná pravidla jako pro používání pseudoznaků ve jménech souborů, například při používání příkazu `ls`. Jistě si pamatujete, že příkazový procesor je schopen zvláštním způsobem interpretovat tzv. pseudoznaky (hvězdičku a otazník). Avšak test `-name foo*` nevrátí tu hodnotu, kterou byste očekávali. Místo toho budete muset použít test `-name foo` nebo `-name "foo*"`. Dobře si tuto situaci zapamatujte, většina uživatelů používá nesprávný test. Je zde ještě jeden rozdíl oproti příkazu `ls` – tečky na začátku nejsou interpretovány. Pokud se potřebujete vyrovnat s tímto problémem, použijte test `-path pattern`, který se o tečky a zpětná lomítka při porovnávání cest nestará.

Akce

Jak jsme si již řekli, argumenty charakterizující akce slouží k inicializaci nějaké operace. Přesto mezi tyto parametry patří například akce `-prune`, která nic neaktivuje, pouze realizuje krok dolů ve stromové struktuře adresářů. Většinou je tato akce spjata s parametrem `-fstype`, prostřednictvím kterého se realizuje výběr mezi různými souborovými systémy. Ostatní akce mohou být rozděleny do dvou širokých kategorií.

- Akce, které něco tisknou. Je zřejmé, že implicitní akcí bude akce `-print`. Ta v průběhu činnosti programu find tiskne jména souborů, které se právě zpracovávají (ovšem za předpokladu, že ostatní podmínky uvedené v příkazovém řádku vracejí hodnotu true). Akce `-print` má několik variant. První z nich, `-fprint file`, používá soubor uvedený jako pa-

rametr `file` místo standardního výstupu. Další, `-ls`, zobrazuje jméno aktuálního souboru jako příkaz `ls -dils`. Akce `-printf format` se chová podobně jako funkce `printf()` v jazyku C (v této variantě lze specifikovat formát výstupu). Totéž realizuje akce `-fprintf file`, ale do souboru uvedeného prostřednictvím parametru `file`. Tato akce rovněž vždy vrací hodnotu true.

- Akce, které aktivují nějaký příkaz. Jejich syntaxe je poněkud zvláštní, a proto se na ně podívejme podrobněji.

  ```
  -exec command \;
  ```

  Akce `-exec` spustí příkaz zadaný prostřednictvím parametru `command` a vrací hodnotu true, pokud má příkaz status ukončení 0. Za příkazem je uvedena dvojice znaků „`\;`“, protože jinak by příkaz `find` nebyl schopen rozeznat, kde příkaz `command` končí (trik s uvedením akce `-exec` na konec příkazu `find` není aplikovatelný). Proto se příkaz `command` ukončuje znakem „`;`“, což je zároveň znak k oddělování příkazů v jazyku příkazového procesoru. Bez znaku „`\`“ by středník byl příkazovým procesorem na základě syntaktické analýzy odstraněn a do příkazu `find` by se již nedostal. Proto se zde znak „`\`“ musí uvést. Další věc, kterou si musíte zapamatovat, spočívá ve způsobu specifikace jména aktuálního souboru uvnitř příkazu `command`. Jméno souboru musí být uvedeno pomocí dvojice složených závorek `{}`. Některé starší verze programu `find` vyžadují, aby bylo jméno odděleno netisknutelnými znaky (white spaces), například mezerami. To však není příliš šikovné, proto je ve verzi GNU zavedena konstrukce se složenými závorkami umožňující generovat řetězce. Asi vás napadá otázka, zda musejí být složené závorky uzavřeny do uvozovek. Podle mých zkušeností nemusejí – ani v případě, že použijete příkazový procesor `bash`, ani v případě, že použijete příkazový procesor `tcsh`. Proto zůstanou v příkazu `find` řetězce uvedené ve složených závorkách příkazovým procesorem nedotčeny.

  ```
  -ok command \;
  ```

  Akce `-ok` se chová podobně jako akce `-exec` s tím rozdílem, že pro každý vybraný soubor je uživatel vyzván k potvrzení příkazu (tj. zda se má provést nebo ne). Pokud odpověď začíná písmenem „y“ nebo „Y“, příkaz se provede a akce vrátí hodnotu true. Jinak se akce neprovede a vrácená hodnota je false.

Operátory

V příkazu `find` lze použít spoustu operátorů. Následující seznam je uvádí v pořadí s klesající prioritou.

```
\( expr \)
```

Kulaté závorky mění prioritu operandu. Závorkám musí předcházet znak „`\`“, protože v příkazovém procesoru mají speciální význam.

```
! expr
-not expr
```

Operátory `!` a `-not` mění pravdivostní hodnotu výrazu `expr`. Je-li hodnota `expr` true, změní se na false a naopak. Znak „`!`“ nemusí být oddělen obráceným lomítkem nebo uvozovkami, protože je následován netisknutelným znakem (mezerou).

```
expr1 expr2
expr1 -a expr2
expr1 -and expr2
```

Všechny tři varianty odpovídají logické operaci AND, přičemž se nejčastěji používá první varianta. Jestliže je první výraz `expr1` vyhodnocen jako false, pak se již druhý výraz nevyhodnocuje.

```
expr1 -o expr2
expr1 -or expr2
```

Obě varianty odpovídají logické operaci OR. Je-li první výraz `expr1` vyhodnocen jako true, pak se již druhý výraz `expr2` nevyhodnocuje.

```
expr1 , expr2
```

Uvedený operátor má poněkud speciální význam. Oba výrazy `expr1` i `expr2` se vyhodnotí (samozřejmě se všemi vedlejšími účinky) a hodnota konečného výrazu je nastavena na hodnotu výrazu `expr2`.

Příklady

Jak vyplývá z předcházejících seznamů, příkaz `find` má spoustu parametrů. Existuje však několik často používaných konstrukcí s příkazem `find`, které stojí za to si zapamatovat. Některé z nich si uvedeme jako příklady.

```
% find .-name foo\* -print
```

První příklad vyhledá všechny soubory, jejichž jména začínají řetězcem `foo`. Pokud se mají hledat soubory, jejichž jména mají řetězec `foo` někde uvnitř, pak je vhodné místo `foo` použít „`*foo*`“.

```
% find /usr/include/ -xtype f -exec grep foobar \
       /dev/null {}\;
```

V druhém příkladu se rekurzivně provádí příkaz `grep`, a to od adresáře `/usr/include`. V tomto případě se zajímáme o obyčejné soubory a symbolické odkazy, které na obyčejné soubory odkazují. Proto jsme použili test `-xtype`. V mnoha případech je jednodušší tento test nepoužít, zejména tehdy, kdy jsme si jisti, že žádný binární soubor neobsahuje požadovaný řetězec. A proč jsme použili příkaz `/dev/null`? Jedná se o trik, který příkaz `grep` „přinutí“ vypsat jméno souboru, pro který nebylo splněno výběrové kritérium. Příkaz `grep` je aplikován na každý soubor jinak, a proto „nepovažuje“ za nezbytné vypisovat jméno souboru. Ale nyní jsou zde dva soubory – aktuální soubor a soubor `/dev/null`. Jiná možnost spočívá v použití roury a příkazu `xargs`. Když jsem ji zkusil, zničil jsem si celý souborový systém.

```
%find / -atime +1 -fstype ext2 -name core \
       -exec rm {}\;
```

Tento příklad představuje klasickou úlohu pro crontab. Zruší ze souborového systému `ext2` všechny soubory s názvem `core`, které nebyly posledních 24 hodin zpřístupněny. Je možné, že někdo tyto soubory používá v souvislosti s programem `gdb` k ladění, ale je málo pravděpodobné, že je bude používat po 24 hodinách.

```
% find /home -xdev -size +500k -ls > piggies
```

Další příklad umožňuje zjistit, kdo vlastní soubory větší než 500 kilobajtů a kdo tedy zatěžuje souborový systém. Poznamenejme, že pokud se zajímáme pouze o jeden souborový systém, pak argument `-xdev` není nutný.

Slovo na závěr

Mějte na paměti, že příkaz `find` spotřebuje velmi mnoho času, pokud má provádět operace s každým souborem v celém souborovém systému. Proto je nutné počet operací vhodně optimalizovat, zejména když se na vašem systému pravidelně realizují úlohy pro údržbu prostřednictvím crontab. Jako poučný příklad vezměme následující: Předpokládejme, že chceme zrušit soubory končící na `.BAK`, přitom chceme změnit přístupová práva všech adresářů na 771 a přístupová práva souborů končících na `.sh` změnit na 755. Přitom chceme ještě připojit souborový systém NFS na telefonní linku, a v tomto systému nechceme žádné soubory kontrolovat. Proč bychom měli psát tři různé příkazy? Nejefektivněji uvedenou úlohu splníme takto:

```
% find .\( -fstype nfs -prune \) -o \
       \( -type d -a -exec chmod 771 {}\; \) -o \
       \( -name "*.BAK" -a -exec /bin/rm {}\; \) -o \
       \( -name "*.sh" -a -exec chmod 755 {}\; \)
```

Asi se vám bude takový příkaz zdát nesrozumitelný, ale když si jej pozorně prohlédnete, zjistíte, že je logický. Pamatujte si, že příkazy tohoto typu neprovádějí nic jiného, než že vyhodnocují výrazy, jejichž hodnota je buď false, nebo true. Samozřejmě mají vedlejší účinky – ty se realizují tehdy, když příkaz `find` musí vyhodnotit část obsahující výraz s akcí `-exec`, což nastane právě tehdy, když je levá strana výrazu vyhodnocena jako pravdivá (true). Když je například právě zpracovávaným souborem adresář, pak se bude realizovat první akce `-exec` a přístupová práva adresáře budou změněna na 771. Ostatní části příkazu budou vynechány. Budete-li takové příkazy aplikovat prakticky, přestanou se vám zdát nesrozumitelné a budete je používat zcela přirozeně.

Archivační program tar

Úvod

Tar je obecně použitelný program pro archivaci souborů, který je schopen sloučit (spakovat) velké množství souborů do jediného archivního souboru, přičemž zachovává veškeré informace o souborech, jako jsou uživatelská práva. Jméno `tar` je zkratkou „tape archive“, protože tento nástroj byl původně používán pro archivaci na magnetické pásky. Jak však uvidíme, používání příkazu `tar` není zdaleka omezeno pouze na záložní soubory pro magnetické pásky.

Hlavní funkce

Formát příkazu `tar` je následující:

`tar` *`functionoptions files...`*

kde *`function`* je jednoznaková indikace operace, která se má provést; *`options`* je seznam (jednoznakových) voleb pro tuto operaci a *`files`* je seznam souborů, jež se mají spakovat nebo rozpakovat. (Všimněte si, že argument function není oddělen od options mezerou.) Parametr `function` může být:

- `c` pro vytvoření nového archivního souboru
- `x` pro extrakci souborů z archivního souboru
- `t` pro výpis obsahu archivního souboru
- `r` pro přidání souborů na konec archivního souboru
- `u` pro aktualizaci souborů, které jsou novější než soubory v archivním souboru
- `d` pro porovnání souborů v archivním souboru se soubory v souborovém systému

Nejčastěji budete používat parametr `c`, `x` a `t`; ostatní budete používat jen zřídka.

Volby

Nejčastěji používané volby `options` jsou tyto:

- `v` pro tisk podrobné informace v průběhu pakování a rozpakování
- `k` pro zachování existujících souborů při rozpakování, tj. žádný existující soubor nebude přepsán souborem z archivního souboru
- `f` *`filename`* pro specifikaci jména archivního souboru

Další volby popíšeme v následujícím oddílu později.

Příklady

Ačkoliv se syntaxe příkazu `tar` zdá být na první pohled složitá, je jeho praktické použití velmi jednoduché. Předpokládejme, že máme adresář `mt` obsahující následující soubory:

```
rutabaga% ls -l mt
-rw-r--r-- 1 root root 24 Sep 21 1993 Makefile
-rw-r--r-- 1 root root 847 Sep 21 1993 README
-rw-r--r-- 1 root root 9220 Nov 16 19:03 mt
-rwxr-xr-x 1 root root 2775 Aug 7 1993 mt.1
-rw-r--r-- 1 root root 6421 Aug 7 1993 mt.c
-rw-r--r-- 1 root root 3948 Nov 16 19:02 mt.o
-rw-r--r-- 1 root root 11204 Sep 5 1993 st_info.txt
```

Nyní chceme spakovat pomocí příkazu `tar` obsah tohoto adresáře do jediného archivního souboru. Použijeme tedy příkaz:

```
tar cf mt.tar mt
```

Prvním argumentem v příkazu `tar` je operace (zde `c` pro vytvoření) následovaná volbou `options`, kde jsme použili `f mt.tar` a specifikovali tak jméno archivního souboru `mt.tar`. Jako poslední je uveden seznam souborů – pokud se místo seznamu uvede název adresáře, `tar` spakuje všechny soubory v tomto adresáři.

Poznamenejme, že prvním argumentem musí být písmeno, označující operaci, následované seznamem voleb „options“. Proto není důvod dávat před první argument pomlčku, jak to vyžaduje většina systémů Unix. Příkaz `tar` v systému Linux však tuto pomlčku připouští, proto by mohl mít posledně uvedený příklad tvar:

```
tar -cf mt.tar mt
```

V některých verzích musí být typ operace (jako je `c`, `t`, nebo `x`) na prvním místě, v jiných verzích na pořadí písmen nezáleží. Jistý přehled o průběhu pakování (nebo rozpakování) získáte prostřednictvím volby `v` – pak se bude každý soubor ukládaný do archivního souboru vypisovat na obrazovce. Například:

```
rutabaga% tar cvf mt.tar mt
mt/
mt/st_info.txt
mt/README
mt/mt.1
mt/Makefile
mt/mt.c
mt/mt.o
mt/mt
```

Použijete-li volbu `v` vícekrát, zobrazovaná informace bude podrobnější:

```
rutabaga% tar cvvf mt.tar mt
drwxr-xr-x root/root 0 Nov 16 19:03 1994 mt/
-rw-r--r-- root/root 11204 Sep 5 13:10 1993 mt/st_info.txt
-rw-r--r-- root/root 847 Sep 21 16:37 1993 mt/README
-rwxr-xr-x root/root 2775 Aug 7 05:50 1993 mt/mt.1
-rw-r--r-- root/root 24 Sep 21 16:03 1993 mt/Makefile
-rw-r--r-- root/root 6421 Aug 7 09:50 1993 mt/mt.c
-rw-r--r-- root/root 3948 Nov 16 19:02 1994 mt/mt.o
-rw-r--r-- root/root 9220 Nov 16 19:03 1994 mt/mt
```

Tyto podrobné informace jsou důležité zejména tehdy, když chcete zkontrolovat, zda `tar` realizuje pakování dle vašich představ. U některých verzí programu `tar` musí být volba `f` uvedena jako poslední. Je to z toho důvodu, že se za volbou `f` předpokládá jméno souboru. Pokud neuvedete volbu `f`, předpokládá `tar` (z historických důvodů), že má použít zařízení `/dev/rmt0`, což je první magnetopásková jednotka.

Nyní můžeme soubor `mt.tar` předat někomu jinému a ten si jej může rozpakovat na svém počítači. K rozpakování by měl použít následující příkaz:

```
tar xvf mt.tar
```

Tento příkaz vytvoří adresář `mt` a umístí do něj všechny soubory, které jsme dříve uvedli – s týmiž přístupovými právy jako v původním systému. Nové soubory budou vlastněny uživatelem, který provedl příkaz `tar xvf` – pokud ovšem nepoužijete tento příkaz jako root (pak je původní vlastník zachován). Parametr `x` zajistí rozpakování souborů a volba `v` opět zobrazí soubory, které se ukládají na disk:

```
courgette% tar xvf mmt.tar
mt/
mt/st_info.txt
mt/README
mt/mt.1
mt/Makefile
mt/mt.c
mt/mt.o
mt/mt
```

Všimněte si, že tar zachoval cestu každého souboru relativní k poloze v původní struktuře adresářů. To znamená, že když jsme vytvářeli archivní soubor pomocí příkazu `tar cf mt.tar mt`, jediný soubor, který jsme specifikovali místo seznamu souborů, byl adresář `mt` obsahující soubory. Proto tar uložil do archivního souboru samotný adresář a soubory, které se v něm nacházely. Při rozpakování se napřed vytvořil adresář `mt` a pak do něj byly uloženy soubory – tedy přesně opačný proces, než jaký probíhal při vytváření archivního souboru. tar implicitně rozpakuje všechny soubory relativně k pracovnímu adresáři, v němž jej spustíte. Pokud se například pokusíte spakovat obsah adresáře `/bin` příkazem:

```
tar cvf bin.tar /bin
```

dostanete varovné hlášení:

```
tar: Removing leading / from absolute path names in the archive.
```

To znamená, že soubory jsou ukládány v archivním souboru uvnitř adresáře `/bin`. Když tento soubor rozpakujete, adresář `/bin` bude vytvořen jako podadresář vašeho pracovního adresáře, v němž spouštíte `tar` – ne jako absolutní adresář `/bin`. Toto je velmi důležitý mechanismus, který byl vymyšlen za účelem ochrany před fatálními chybami při rozpakování pomocí příkazu `tar`. Jinak byste, podle předchozího příkladu, mohli přepsat soubory v adresáři `/bin`, a tak zničit systém.

Pokud byste však opravdu chtěli rozpakovat takový archivní soubor do adresáře `/bin`, museli byste mít jako pracovní adresář nastaven `/`. Výše uvedený mechanismus můžete potlačit pomocí volby `p`, ale nedoporučuje se to. Jiný způsob, jak vytvořit soubor `mt.tar`, spočívá v tom, že se přepnete do adresáře `mt` pomocí příkazu `cd` a zadáte příkaz:

```
tar cvf mt.tar *
```

Potom ovšem nebude podadresář `mt` ukládán do archivního souboru a při rozpakování budou soubory ukládány přímo do vašeho pracovního adresáře. Proto doporučujeme vždy pakovat soubory tak, aby archivní soubor obsahoval podadresář, jak jsme ukázali pomocí příkazu `tar cvf mt.tar mt`. Pak se vždy před rozpakováním vytvoří adresář pro uložení souborů a nemůže se stát, že přepíšete soubory ve svém pracovním adresáři. Navíc zbavíte osobu, která provádí rozpakování, starostí s vytvářením adresáře pro ukládání souborů a ušetříte jí dost času. Samozřejmě existuje mnoho situací, při nichž nebude vhodné doporučený postup dodržovat, ale všeobecně se takový postup považuje za jakousi etiketu při práci s programem `tar`.

Při vytváření archivních souborů můžete uvést seznam souborů nebo adresářů, které se mají do archivního souboru uložit. V prvním příkladu jsme použili příkaz `tar` pro jediný adresář a ukázali jsme použití hvězdičky, kterou příkazový procesor rozšíří na seznam všech souborů v pracovním adresáři. Před rozpakováním archivního souboru pomocí příkazu `tar` je vhodné se podívat, co je jeho obsahem. Tak se například dozvíte, zda budete muset vytvořit adresář pro uložení souborů vlastními silami, nebo bude vytvořen automaticky. K prohlížení obsahu archivního souboru použijte příkaz:

```
tar tvf tarfile
```

Ten zobrazí obsah archivního souboru `tarfile`. Poznamenejme, že pokud použijete funkci t, stačí uvést jednu volbu `v` a obdržíte podrobnou informaci o obsahu archivního souboru:

```
courgette% tar tvf mt.tar mt
drwxr-xr-x root/root 0 Nov 16 19:03 1994 mt/
```

```
-rw-r--r-- root/root 11204 Sep 5 13:10 1993 mt/st_info.txt
-rw-r--r-- root/root 847 Sep 21 16:37 1993 mt/README
-rwxr-xr-x root/root 2775 Aug 7 05:50 1993 mt/mt.1
-rw-r--r-- root/root 24 Sep 21 16:03 1993 mt/Makefile
-rw-r--r-- root/root 6421 Aug 7 09:50 1993 mt/mt.c
-rw-r--r-- root/root 3948 Nov 16 19:02 1994 mt/mt.o
-rw-r--r-- root/root 9220 Nov 16 19:03 1994 mt/mt
```

Žádné rozpakování v tomto případě neproběhne, ale pouze se zobrazí informace o obsahu archivního souboru. Zde vidíme, že jména souborů jsou doplněna jménem adresáře `mt`, a proto při rozpakování bude tento adresář nejdříve vytvořen a pak teprve do něj budou ukládány soubory. Příkaz `tar` můžete rovněž použít k extrakci jednotlivých souborů z archivního souboru. Potom použijte příkaz ve tvaru:

```
tar xvf tarfile files
```

kde `files` je seznam souborů, které se mají rozpakovat. Jak jsme si již ukázali, pokud neuvedeme žádný seznam, `tar` rozpakuje všechny soubory z archivního souboru.

Jestliže specifikujete soubory, které se mají rozpakovat, musíte uvést jejich plná jména včetně cesty tak, jak jsou uvedena v archivním souboru. Chceme-li například rozpakovat soubor `mt.c` z výše uvedeného archivního souboru `mt.tar`, použijeme následující příkaz:

```
tar xvf mt.tar mt/mt.c
```

který nejdříve vytvoří podadresář `mt` a do něj pak umístí soubor `mt.c`. Program `tar` má mnohem více možností, než které jsme zde uvedli. Ty, o nichž jsme se zde zmínili, budete zřejmě používat nejčastěji. Verze GNU implementovaná operačnímu systému Linux má řadu přípon a představuje ideální nástroj pro pořizování záložních kopií. Více informací naleznete v manuálové stránce k příkazu `tar`.

Jak používat program tar spolu s programem gzip

Program tar neprovádí při ukládání dat do archivního souboru žádnou komprimaci. Jestliže vytvoříte soubor `tar` ze tří souborů o velikosti 200 KB, pak výsledný soubor bude mít velikost 600 KB. Proto patří k běžné praxi komprimovat soubory `tar` pomocí programu `gzip` (nebo starším programem `compress`). Komprimovaný soubor tar můžete vytvořit pomocí následujících příkazů:

```
tar cvf tarfile files...
gzip -9 tarfile
```

Provedení takových příkazů je však těžkopádné a vyžaduje, abyste měli na disku dostatek místa pro nekomprimovaný soubor `tar`, než spustíte `gzip`.

Mnohem efektivnější způsob spočívá v tom, že se využije možnost programu tar zapisovat archivní soubor do standardního výstupu. Pokud použijete místo jména archivního souboru pomlčku (`-`), pak bude `tar` číst data ze standardního vstupu nebo zapisovat do standardního výstupu. K vytvoření komprimovaného souboru tar tedy můžeme použít příkaz:

```
tar cvf - files... | gzip -9 > tarfile.tar.gz
```

Zde vytváří `tar` archivní soubor ze souborů uvedených v seznamu `files`, zapisuje jej do standardního výstupu; `gzip` čte data ze standardního vstupu, komprimuje je a zapisuje do svého standardního výstupu. Nakonec jsme přesměrovali komprimovaný soubor `tar` do `tarfile.tar.gz`. Podobně bychom mohli k rozpakování takového souboru použít příkaz:

```
gunzip -9c tarfile.tar.gz | tar xvf -
```

Zde `gunzip` provede dekomprimaci uvedeného archivního souboru a zapisuje výsledek do standardního výstupu sloužícího jako standardní vstup pro program `tar`, jenž soubory rozpakuje a ukládá. Není práce v systému Unix zábavná? Samozřejmě, že jsou oba výše uvedené příkazy poněkud těžkopádné. Naštěstí GNU-verze programu tar má možnost uvést volbu `z`, která zajistí automatické vytváření nebo rozpakování komprimovaných archivních souborů `tar`. (Diskusi o použití volby `z` jsme úmyslně zařadili až na toto místo, abyste si uvědomili její výhody.) K vytvoření a rozpakování archivních souborů máte tedy možnost používat následující příkazy:

```
tar cvzf tarfile.tar.gz files...
```

a

```
tar xvzf tarfile.tar.gz
```

Poznamenejme, že byste měli soubory vytvořené tímto způsobem označovat pomocí přípony `.tar.gz`, aby byl zřejmý jejich formát. Volba funguje i ve spojení s dalšími parametry, jako je například `t`. Možnost používat volbu `z` podporuje pouze GNU-verze programu `tar`. Jestliže používáte `tar` na jiných systémech Unix, musíte pro realizaci stejného úkolu zadávat delší příkazy, jak jsme uvedli dříve. Téměř všechny verze operačního systému Linux používají verzi GNU.

Nyní využijeme svých znalostí a napíšeme krátké skripty pro příkazový procesor, které vám mohou sloužit jako návod pro vytváření a rozpakování souborů `tar`. V příkazovém procesoru bash doplňte následující řádky do souboru `.bashrc`:

```
tarc () {tar czvf $1.tar.gz $1 }
tarx () {tar xzvf $1 }
tart () {tar tzvf $1 }
```

Budete-li nyní chtít vytvořit komprimovaný archivní soubor obsahující soubory z jednoho adresáře, stačí zadat příkaz:

```
tarc directory
```

Výsledný archivní soubor bude mít název `directory.tar.gz`. (Přesvědčte se, že jste neuvedli před jménem adresáře lomítka (`/`) – jinak bude archivní soubor vytvořen jako `.tar.gz` uvnitř daného adresáře.) K zobrazení obsahu archivního souboru stačí zadat příkaz

```
tart file.tar.gz
```

a k jeho rozpakování příkaz

```
tarx file.tar.gz
```

Triky při používání programu tar

Protože tar ukládá do archivního souboru informace o vlastnictví souborů, přístupových právech, adresářové struktuře i symbolických odkazech, velmi dobře se hodí ke kopírování nebo přesouvání celých adresářových stromů z jednoho místa na druhé v rámci jednoho systému (a dokonce i mezi systémy, jak uvidíte). Použijete-li syntaxi s pomlčkou (`-`), budete zapisovat výsledný soubor do standardního výstupu, který může být čten jako standardní vstup a rozpakován kdekoliv.

Předpokládejme například, že máme adresář obsahující dva podadresáře: `from-stuff` a `to-stuff`. Adresář `from-stuff` obsahuje celý strom dalších podadresářů s mnoha soubory, symbolickými odkazy atd. a bylo by velmi obtížné zkopírovat jej pomocí příkazu `cp`. Ke zkopírování celého adresáře `from-stuff` do adresáře `to-stuff` však můžeme použít následující příkazy:

```
cd from-stuff
tar cf - .| (cd ../to-stuff; tar xvf -)
```

Velice jednoduché a elegantní! Začínáme v adresáři `from-stuff`, kde vytvoříme soubor tar obsahující celý adresář, a zapíšeme jej do standardního výstupu. Tento výstup pak čte podřízený příkazový procesor (příkazy uvedené v závorkách), který se nejdříve přepne do adresáře `../to-stuff` a pak spustí příkaz `tar xvf`, jenž čte ze standardního vstupu. Přitom se žádný archivní soubor `tar` neukládá na disk – data jsou přímo posílána z jednoho procesu tar do druhého. Druhý proces tar používá volbu `v` pro zobrazení informací o každém souboru, který se ukládá, a vy můžete kontrolovat, zda kopírování obsahu adresářů probíhá správně.

Pomocí tohoto „triku“ můžete ve skutečnosti přenášet celé adresáře mezi jednotlivými systémy – stačí vložit vhodný příkaz `rsh` mezi příkazy uzavřené v závorkách. Pak vzdálený příkazový procesor spustí `tar`, který bude číst archivní soubor jako standardní vstup. (Poznamenejme, že verze GNU příkazu `tar` má možnost automaticky číst nebo zapisovat soubory `tar` z/do jiných počítačů v síti; informace najdete v příslušné manuálové stránce.)

Program dd

Existuje jakási legenda, podle které v ranných dobách vývoje operačního systému Unix potřebovali programátoři program pro binární kopírování dat mezi jednotlivými zařízeními. Protože velmi spěchali, „vypůjčili“ si syntaxi příkazu používanou v operačním systému na počítačích IBAlt+360 a později vyvinuli rozhraní konzistentní s ostatními příkazy operačního systému Unix. Nevím, zda je tato legenda pravdivá, ale pěkně se vypráví.

Volby

Přes svůj legendou opředený původ není program `dd` úplně jiný než ostatní příkazy operačního systému Unix. Ve skutečnosti představuje filtr, který implicitně čte data ze standardního vstupu a zapisuje je na standardní výstup. Pokud zadáte na terminálu pouze příkaz `dd`, pak bude program `dd` tiše očekávat vstup z klávesnice.

Syntaxe příkazu `dd` je následující:

```
dd [if=file] [of=file] [ibs=bytes] [obs=bytes]
   [bs=bytes] [cbs=bytes] [skip=blocks] [seek=blocks]
   [count=blocks] [conv={ascii, ebcdic, ibm, block,
   unblock, lcase, ucase, swab, noerror, notrunc, sync}]
```

Všechny volby mají tvar *option=value*. Před a za znakem „=“ se nesmí objevit mezera, což je nepříjemné, protože v takové situaci neprovede příkazový procesor doplnění jména souboru, pokud se uvedou pseudoznaky. Verze příkazového procesoru bash v operačním systému Linux je však dostatečně inteligentní, proto nemusíte mít obavy. Všechny výše uvedené číselné hodnoty (`bytes` a `blocks`) mohou být následovány multiplikativní konstantou. Pro tyto konstanty lze použít následující zkratky: **b** pro bloky (multiplikativní konstanta 512), **k** pro kilobajty (multiplikativní konstanta 1024), **w** pro slova (multiplikativní konstanta 2) a prostřednictvím **xm** se číselná hodnota násobí konstantou **m**.

Význam voleb příkazu `dd` je popsán v následujícím seznamu.

- `if=`*`filein`* a `of=`*`fileout`* jsou volby specifikující jméno vstupního a výstupního souboru. Výstupní soubor je zkrácen na velikost danou volbou `seek`. Pokud není klíčové slovo `seek` uvedeno, pak je hodnota `seek` rovna nule (soubor je před provedením operace zrušen). Volba `notrunc` (viz dále) však může toto implicitní chování příkazu `dd` změnit.
- `ibs=`*`nn`* a `obs=`*`nn`* jsou volby specifikující počet bajtů, které se mají najednou přečíst a najednou zapsat. Domnívám se, že implicitní hodnoty jsou jeden blok (t.j. 512 bajtů), ale nejsem si tím tak docela jist. Uvedené parametry jsou velmi důležité v případě, kdy se jako vstup nebo výstup používají speciální zařízení – například při čtení ze sítě je hodnota `ibs` nastavena na 10 kilobajtů, zatímco disketa 3,5 palce má přirozenou délku bloku 18 kilobajtů. Nevhodné nastavení těchto hodnot může nejen značně prodloužit realizaci programu, ale také může vést k neodstranitelným chybám. Proto buďte opatrní.
- `bs=`*`nn`* je volba, která předefinuje nastavení `ibs` a `obs` – obě hodnoty se nastaví na stejnou konstantu *nn*.
- Volba `cbs=`*`nn`* nastavuje vyrovnávací paměti pro konverzi na *nn* bajtů. Vyrovnávací paměť se používá při konverzi z formátu ASCII na EBCDIC nebo při konverzi mezi textovými a binárními soubory a podobně. Například soubory vytvořené pod operačním systémem VMS mají zpravidla velikost bloku 512 bajtů, proto byste měli například při čtení dat zapsaných na magnetickou pásku v operačním systému VMS nastavit hodnotu cbs na 1 bajt.
- `skip=`*`nbl`* a `seek=`*`nbl`* jsou volby, které „sdělí“ programu `dd`, kolik bloků má „přeskočit“ při čtení vstupu a při zápisu na výstup. Samozřejmě platí, že druhá volba má smysl jedině tehdy, když je specifikována konverze notrunc. Velikost bloků je dána volbami `ibs` a `obs`. Uvědomte si, že pokud nezadáte hodnotu ibs a zadáte hodnotu `skip=1b`, pak ve skutečnosti dojde k „přeskočení“ 512x512 bajtů, což je 256 kilobajtů.
- Volbou `count=`*`nbl`* se nastavuje, kolik bloků se má ze vstupu kopírovat. Velikost bloku je přitom dána hodnotou `ibs`. Uvedená volba spolu s předcházející volbou je užitečná v případě, kdy chcete obnovit co nejvíce bajtů z porušeného souboru – „přeskočíte“ nečitelnou část souboru a zbytek zkopírujete do nového souboru.
- Volba `conv=`*`conversion`*`,[`*`conversion`*`,...]` je určena ke konverzi podle specifikace. Možné konverze jsou následující: `ascii` – konverze formátu EBCDIC na formát ASCII; `ebcdic` nebo `ibm` – konverze z formátu ASCII na EBCDIC (jednoznačná konverze z formátu EBCDIC na formát ASCII neexistuje; první představuje standardní konverzi a druhá funguje lépe, pokud se má soubor tisknout na tiskárně IBM); `block` – vyplňuje mezerami záznamy ukončené znakem newline na délku uvedenou prostřednictvím volby `cbs`; `unblock` – opačná konverze ke konverzi `block`; `lcase` – konverze z velkých písmen na malá; `ucase` – konverze z malých písmen na velká; `swab` – konverze, při které se každý pár vstupních bajtů zamění (chcete-li například použít na počítači s procesorem Intel soubor obsahující celá čísla, který byl vytvořen na počítači s procesorem 680x0, budete takovou

konverzi potřebovat); `noerror` – program `dd` bude pokračovat v činnosti i poté, co nastane nějaká chyba; `sync` – každý vstupní blok se doplní znaky NUL tak, aby měl délku definovanou prostřednictvím volby `ibs`.

Příklady

Dříve či později se setkáte s případem, kdy budete chtít pod operačním systémem Linux vytvořit svoji první disketu. Jak zapsat data na disketu bez souborového systému MS-DOS? Řešení je jednoduché:

```
% dd if=disk.img of=/dev/fd0 obs=18k count=80
```

V uvedeném příkladu jsem se rozhodl nepoužít volbu `ibs`, protože nevím jaká je optimální volba pro velikost bloku na pevném disku. Nemůže však uškodit, když se místo volby `obs` použije volba `bs` – pak je proces kopírování dokonce rychlejší. Všimněte si nastavení počtu sektorů pro zápis (18 kilobajtů je velikost sektoru, proto je hodnota `count` nastavena na `80`). Pro disketovou jednotku se v operačním systému Linux používá jméno zařízení `/dev/fd0`.

Další užitečné použití příkazu `dd` je v oblasti zálohování, zejména při zapojení počítače do sítě. Předpokládejme, že máte počítač alfa a že na počítači beta je magnetopásková jednotka `/dev/rst0` obsahující soubor, který chcete zkopírovat. Dále předpokládejme, že máte stejná přístupová práva na obou počítačích a že na pevném disku počítače beta není dostatek místa pro kopii uvažovaného souboru. Pak můžete použít příkaz:

```
% rsh beta 'dd if=/dev/rst0 ibs=8k obs=20k' | tar xvBf -
```

Celou operaci takto realizujete „na jeden průchod“. Zajisté jste si všimli, že se v příkazu využila schopnost příkazového procesoru číst data z magnetopáskové jednotky. Velikosti vstupních a výstupních bloků jsou nastaveny na implicitní hodnoty, tedy 8 kilobajtů pro čtení a 20 kilobajtů pro zápis do sítě ethernet.

Poznámka na závěr: zapomněl jsem říci, že označení příkazu `dd` je zkratkou pro výraz „data duplicator“.

KAPITOLA 12

Chyby, skryté závady a další nepříjemnosti

Jak předcházet chybám

Řada lidí na nejrůznějších fórech dává najevo zklamání z operačního systému Unix. Jejich zklamání zpravidla vyplývá z toho, že jej neumějí efektivně využívat – obvykle se jedná o uživatele dříve zvyklé na operační systémy s pohodlnou obsluhou, jako je Microsoft Windows, Macintosh Operating System a podobně. Naopak lidé, kteří zvládli filosofii operačního systému Unix a jsou schopni v něm efektivně pracovat, na něj nedají dopustit. Cení si zejména toho, že jen málokterý příkaz vyžaduje v průběhu své realizace nějakou komunikaci s uživatelem, a proto v operačním systému vše běží hladce, rychle a bez problémů.

Právě skutečnost, že například příkazy `rm` a `mv` nikdy nevyžadují potvrzení, zda mají skutečně zrušit specifikované soubory, vede často k problémům. Proto si nyní projděme malý seznam, v němž uvádíme zásady, jak se takovým a podobným problémům vyhnout.

- Pořizujte si záložní kopie. Tato výzva platí zejména pro uživatele, kteří používají systém sami. Každý systémový administrátor by měl pravidelně pořizovat záložní kopie – dostatečný interval je asi jeden týden. Podrobnosti najdete v *„Příručce správce operačního systému Linux“*.
- Každý uživatel by si měl pořizovat své vlastní záložní kopie. Pokud používáte více jak jeden systém, pak si uchovávejte aktualizované kopie všech svých souborů na každém systému. Jestliže máte přístup k disketové jednotce, pak si na ni ukládejte kopie důležitých souborů. Přinejhorším si kopie důležitých souborů uchovávejte alespoň v oddělených adresářích.
- Důkladně si promyslete, zda jste správně zadali „destruktivní“ příkazy, jako je `mv`, `rm` a `cp`. Zrovna tak si dejte pozor na přesměrování (>) souborů – i to může být nebezpečné a vyžaduje zvláštní pozornost. Nevinně vyhlížející opomenutí může způsobit katastrofu:

  ```
  /home/larry/report# cp report-1992 report-1993 backups
  ```

  Stačí vynechat poslední parametr a neštěstí je hotovo (místo zálohy máte zrušen jeden soubor):

  ```
  /home/larry/report# cp report-1992 report-1993
  ```

- Autor rovněž doporučuje na základě vlastních zkušeností neprovádět zálohování ani další úlohy údržby pozdě v noci, kdy jste unaveni a snadno uděláte chybu. Jestli o půl druhé v noci zjistíte, že máte příliš plný disk, pak jej nechte plným a nezačínejte hned rušit soubory – takovou práci si nechte až na ráno, kdy budete vyspaní a odpočatí.
- Používejte takovou výzvu příkazového řádku, která vás bude informovat o aktuálním adresáři. Pokud takovou výzvu nepoužíváte, rovněž hrozí nebezpečí. Následující příklad nám zaslal člen diskusní skupiny `comp.unix.admin`[1], který si nevšiml, že není v adresáři `/tmp`:

```
mousehouse> pwd
/etc
mousehouse> ls /tmp
passwd
mousehouse> rm passwd
```

  Série právě uvedených příkazů by vás mohla učinit velmi nešťastnými při pohledu na to, jak si rušíte soubor `passwd`, bez kterého se nemůže nikdo do operačního systému Unix přihlásit.

Chyba není ve vás

Naneštěstí pro programátory na celém světě platí, že ne všechny problémy pocházejí z chyb uživatelů. Operační systémy Unix a Linux jsou velmi komplikované a všechny známé verze mají chyby či skryté závady. Některé z těchto závad lze jen velmi těžko odstranit, protože se projevují jen za velmi speciálních okolností.

V souvislosti s chybami programového vybavení se používá termín „bug“ (doslova štěnice). Asi nejvýstižnější překlad bude „skrytá chyba“, protože se jedná o chybu, o které autoři programového vybavení neví. Chyba tohoto druhu se odhalí až při testování daného programu, někdy až po velmi dlouhé době.

Co dělat, když objevíte skrytou chybu

Když vám počítač dá chybnou odpověď (nejdříve důkladně prověřte, zda je odpověď opravdu chybná) nebo nějaký program zhavaruje, pak lze uvažovat o skryté chybě.

Skrytou chybu může obsahovat program, který stále běží, přesto že by měl skončit – i v tomto případě byste však měli nejdříve zkontrolovat, zda neprovádí příliš složité a zdlouhavé operace. Než použijete nový příkaz, důkladně si prostudujte příslušnou dokumentaci (nejlépe manuálové stránky).

Některá hlášení vám budou připadat jako skryté chyby, i když ve skutečnosti skrytými chybami nebudou. Prostudujte si oddíl Hlášení jádra systému a příslušnou dokumentaci, než učiníte nějaké závěry o skrytých chybách.

Například taková hlášení, jako „`disk full`“ nebo „`lp0 in fire`“ neznamenají, že je program vadný, ale že je něco v nepořádku s vaším technickým vybavením – nedostatek diskového prostoru nebo špatně připojená tiskárna.

1 Jedná se o mezinárodní diskusní skupinu, která řeší otázky kolem správy operačního systému Unix.

Pokud nemůžete najít něco v dokumentaci, pak je to chybou dokumentace a měli byste autora programového vybavení kontaktovat a o nedostatku informovat. Zrovna tak je chybou dokumentace, když popisuje jiné vlastnosti programového vybavení, než jaké ve skutečnosti jsou.[2] Je-li něco nekompletního nebo nejasného v dokumentaci, pak se jedná o chybu dokumentace.

Naopak, jestliže nejste schopni porazit šachový program `gnuchess`, pak je to tím, že algoritmus tohoto programu je opravdu dobrý a neznamená to nutně, že je ve vašem mozku skrytá chyba.

Jak ohlásit chybu

Jakmile jste si jisti, že jste odhalili skrytou chybu, je nutné zajistit, aby se hlášení o ní dostalo na správné místo. Pokuste se zjistit, co chybu způsobilo a pokuste se rekonstruovat všechny okolnosti, za jakých chyba nastala. Jestli se vám chybu nepodaří znovu „vyvolat" pročtěte si zprávy z diskusní skupiny `comp.os.linux.help` nebo `comp.unix.misc`. Také si důkladně pročtěte manuálové stránky k danému programu.

Při odesílání zpráv o skryté chybě se dává přednost elektronické poště. Pokud nemáte možnost odesílat zprávy prostřednictvím elektronické pošty, kontaktujte osobu, od které máte operační systém Linux nebo zkuste kontaktovat někoho, kdo přístup k elektronické poště má. Také se můžete obrátit na komerčního dodavatele operačního systému Linux. Ten má určitě zájem na tom, aby se skryté chyby rychle odstraňovaly. Mějte na paměti, že nikdo není povinen odstraňovat chyby, dokud s ním nemáte podepsanou příslušnou smlouvu.

Když odesíláte hlášení o chybě, pak uveďte všechny informace a okolnosti, za kterých se chyba projevila. Dodržujte dále uvedené zásady:

- Popište, co si myslíte, že má program dělat a co ve skutečnosti dělá. Například: „Program vrací hodnotu 5, když mu byl zadán výraz 2+2". Nebo „Program ohlásil `segment violation -- core dumped`". Je velmi důležité popsat, co se stalo, aby mohl autor chybu odstranit.
- Uveďte všechna nastavení proměnných prostředí.
- Uveďte verzi jádra operačního systému (najdete ji v souboru `/proc/version`) a verzi systémových knihoven (podívejte se do adresáře `/lib`, nebo pošlete výpis obsahu tohoto adresáře).
- Popište, jak jste program spustili, a popište, co jste dělali, když k chybě došlo.
- Uveďte všechny okolnosti, za kterých k chybě došlo. Řekněme například, že příkaz `w` některému uživateli nezobrazí informace o aktuálním procesu. Nestačí napsat: „Příkaz `w` nefunguje pro jistého uživatele". Chyba může nastat proto, že jméno uživatele má osm znaků nebo proto, že se přihlásil prostřednictvím sítě. Správná informace bude tedy znít takto: „Příkaz `w` nezobrazuje informace o aktuálním procesu uživateli greenfie, když se přihlásí prostřednictvím sítě.".
- Buďte zdvořilí. Většina lidí pracuje obětavě na volném programovém vybavení z vlastní iniciativy, vlastní dobré vůle a především proto, aby vám předložili program, který potřebujete. Nebuďte na lidi, kteří pracují od rána do noci, hrubí – právě hrubé osočování dokázalo odradit nejednoho nadějného programátora od práce na operačním systému Linux.

2 To bude asi také platit o tomto manuálu.

PŘÍLOHA A

Technické informace

Editor `vi` (vyslovuje se [ví áj]) je jediným editorem, který se nachází v každé instalaci operačního systému Unix. Původně byl vytvořen na univerzitě California v Berkeley, jeho různé verze se nacházejí ve všech distribucích operačního systému Unix a je také součástí distribuce operačního systému Linux. Editor `vi` se poměrně těžko učí, ale má mnoho velmi výkonných funkcí. Obecně se doporučuje začátečníkům editor Emacs, protože se mnohem snáze používá. Avšak uživatelé, kteří používají více počítačových platforem, raději pracují s editorem `vi`.

Abychom pochopili, proč klávesa K znamená posun kurzoru o jeden řádek nahoru a proč editor pracuje ve třech odlišných módech, musíme se podívat do historie. Pokud máte pokušení naučit se pracovat s editorem `vi`, pak můžete považovat tento dodatek za učebnici, která vás provede veškerými základy. Také zde předkládáme přehled příkazů, který můžete považovat za referenční příručku.

Dokonce i v případě, že editor `vi` nebude editorem pro vaši běžnou práci, není čas věnovaný jeho základům úplně zbytečný. Je téměř jisté, že v operačním systému typu Unix, který používáte, je editor `vi` instalován. Někdy je nezbytné použít editor `vi` při instalaci jiných programových produktů, například editoru Emacs. Spousta nástrojů operačního systému Unix, aplikací a her používá jistou podmnožinu příkazů editoru `vi`.

Stručná historie editoru vi

První textové editory byly orientovány na zpracování textových souborů řádek po řádku a používaly se na neinteligentních terminálech. Typickým editorem, který takto funguje, je editor **Ed**. Editor Ed je velmi výkonný a využívá velmi málo zdrojů počítače. V porovnání s editorem Ed nabízí editor `vi` uživateli vizuální alternativu s mnohem širší množinou příkazů.

Editor `vi` se startuje stejně jako řádkový editor `ex`. Platí, že editor `ex` je vlastně editor `vi` spuštěný ve speciálním módu. Vizuální složka editoru `ex` může být inicializována z příkazového řádku pomocí příkazu editoru `vi` nebo přímo z editoru `ex`.

Editor `ex/vi` byl vyvinut na univerzitě California v Berkeley a jeho autorem je William Joy. Původně byl dodáván jako nepodporovaný obslužný program. Oficiálně byl zařazen až v operačním systému System V Unix od firmy AT&T. Postupně se stal velmi populárním a dodnes konkuruje moderním celoobrazovkovým editorům.

V důsledku své popularity se editor `vi` objevil v mnoha verzích a dnes existují verze pro většinu operačních systémů (i jiných, než Unix). Cílem této kapitoly není popsat všechny dostupné příkazy editoru `vi` a jejich modifikace. Právě v důsledku toho, že vzniklo mnoho verzí editoru `vi`, není množina jeho příkazů standardizována. Řada klonů editoru `vi` dokonce nepodporuje některé původní příkazy.

Jestliže máte nějaké zkušenosti s editorem `ed`, pak se editor `vi` naučíte mnohem snáze. I když editor vi nehodláte používat, budou se vám základní znalosti hodit zejména v nouzových situacích.

Stručný výklad příkazů editoru Ed

Cílem tohoto oddílu je naučit vás pracovat s editorem `ed`. Byl navržen tak, aby se jej každý snadno naučil. Než jej budete moci používat, budete muset chvíli trénovat. Při probírání jednotlivých příkazů si hned vyzkoušejte uvedené příklady – tak se naučíte s editorem pracovat velmi rychle.

Vytvoření souboru

Editor `ed` je schopen editovat v daném okamžiku pouze jeden soubor. Vyzkoušejte si následující příklad a vytvořte si svůj první soubor prostřednictvím editoru `ed`.

```
/home/larry# ed

a

This is my first text file using Ed.
This is really fun.
.
w firstone.txt
q
/home/larry#
```

Nyní si můžete obsah souboru ověřit pomocí příkazu `cat` nebo `more`:

```
/home/larry# cat firstone.txt
```

Předcházející příklad ilustruje spoustu důležitých aspektů. Po spuštění editoru se objeví prázdný řádek. Příkaz `a` je určen k přidání textu do souboru. Pokud chcete ukončit zadávání textu, zapište do prvního sloupce na novém řádku tečku. Chcete-li text uložit, pak zadejte příkaz `w` a jméno souboru. Samotný příkaz `q` činnost editoru ukončí.

Nejdůležitější je poznatek, že editor pracuje ve dvou módech – v **textovém módu** a v **příkazovém módu**. Svoji činnost editor zahajuje v příkazovém módu, kdy lze na prázdných řádcích zadávat příkazy. Ty jsou definovány prostřednictvím jisté množiny znaků.

Jak editovat existující soubor

Pokud chcete do existujícího souboru přidat řádek, postupujte dle následujícího příkladu:

```
/home/larry# ed firstone.txt

a

This is a new line of text.


q
```

Když zkontrolujete soubor `firstone.txt` pomocí příkazu `cat`, zjistíte, že nový řádek byl vložen mezi původní první a druhý řádek. Jak editor `ed` pozná, kam má vložit nový textový řádek?

Po načtení souboru si editor `ed` uchovává informaci o aktuálním řádku. Příkaz a vkládá nový text za aktuální řádek. V editoru `ed` také můžete vkládat nový řádek před aktuální řádek a to prostřednictvím příkazu `i`.

Nyní je patrné, že editor `ed` pracuje s textem řádek po řádku. Všechny příkazy mohou být aplikovány na specifikovaný řádek.

Dále uveďme příklad, jak přidat textový řádek na konec souboru:

```
/home/larry# ed firstone.txt
$a
The last line of text.
.
w
q
```

Modifikátor příkazu `$` „sdělí“ editoru `ed`, že má přidat řádek za poslední řádek stávajícího textu. Pokud byste chtěli přidat textový řádek za první řádek, použijte modifikátor `1`. Nyní máte k dispozici příkazy, pomocí kterých lze vkládat nový text před nebo za řádek specifikovaného čísla.

Jak se dozvíme, který řádek je aktuální? Stačí zadat příkaz `p` a zobrazí se obsah aktuálního řádku. Pokud chcete změnit aktuální řádek na hodnotu 2, pak postupujte dle následujícího příkladu:

```
/home/larry# ed firstone.txt
2p
q
```

Podrobnosti o číslování řádků

Ukázali jsme si, jak prostřednictvím příkazu `p` zobrazit obsah aktuálního řádku. Také víme, že pro příkazy existují modifikátory ve formě pořadových čísel řádků. Chceme-li vypsat obsah druhého řádku, stačí zadat příkaz:

2p

Existují také jiné speciální modifikátory, prostřednictvím kterých lze měnit nastavení aktuálního řádku. Znak dolaru `$` se používá pro poslední řádek textu. Chcete-li vypsat obsah posledního řádku, zadejte:

```
$p
```

Pro aktuální číslo řádku se používá speciální modifikátor tečka. Obsah aktuálního řádku prostřednictvím tohoto modifikátoru lze vypsat takto:

```
.p
```

Možná se vám takový příkaz zdá zbytečný. Když však často měníte obsah aktuálního řádku, pak zjistíte, jak je příkaz užitečný.

Pokud chcete zobrazit text v rozsahu řádků od 1 do 2, pak musíte specifikovat rozsah:

```
1,2p
```

První číslo odkazuje na počáteční řádek, jenž se má zobrazit, a druhé číslo odkazuje na poslední řádek, jenž se má zobrazit. Aktuální řádek se po provedení tohoto příkazu nastaví na druhou hodnotu.

Další příklad ukazuje, jak zobrazit obsah souboru od prvního řádku po aktuální řádek:

```
1,.p
```

Také můžete zobrazit obsah souboru od aktuálního řádku po řádek poslední:

```
.,$p
```

Nyní budete jistě umět zadat příkaz, který zobrazí obsah celého souboru.

Dále si ukážeme, jak zrušit první dva řádky souboru:

```
1,2d
```

Příkaz `d` ruší text řádek po řádku. Chcete-li zrušit celý text, stačí zadat:

```
1,$d
```

Pokud provedete v editovaném textu velké množství změn a nechcete obsah souboru uložit, pak je nejlepší editor ukončit bez zápisu.

Většina uživatelů nepoužívá editor `ed` jako svůj hlavní editor. Moderní editory jsou celoobrazovkové a nabízejí mnohem pružnější množinu příkazů. Editor `ed` představuje dobrý úvod k editoru `vi` a pomůže vám pochopit, odkud příkazy editoru `vi` pocházejí.

Stručný výklad příkazů editoru vi

Cílem tohoto oddílu je naučit vás pracovat s editorem `vi`. Předpokládáme, že nemáte s editorem `vi` žádné zkušenosti a probereme si deset nejzákladnějších příkazů. Tyto příkazy vám budou stačit k realizaci nejnutnějších kroků při editování souboru a další příkazy se pak snadno naučíte podle svých potřeb. Při probírání jednotlivých příkazů si hned vyzkoušejte uvedené příklady – tak se naučíte s editorem pracovat velmi rychle.

Jak spustit editor vi

Chcete-li spustit editor `vi`, zadejte jméno programu a jako parametr uveďte jméno editovaného textového souboru. Objeví se vám obrazovka, na jejíž levé straně bude zobrazen sloupec znaků tilda (~). Editor `vi` se nyní nachází v příkazovém módu – cokoliv napíšete, bude považováno za příkaz a nikoliv za vstupní text. Jestliže chcete zadávat text, musíte nejdříve zadat příkazy. Pro vkládání textu existují tyto dva základní příkazy:

- `i` vložení textu vlevo od kurzoru
- `a` přidání textu vpravo od kurzoru.

Protože se v tomto okamžiku nacházíte na začátku prázdného souboru, nezáleží na tom, který z uvedených příkazů použijete. Zadejte tedy jeden z nich a zapište následující text. (Jedná se o báseň, jejímž autorem je Augustus DeMorgan. Je uvedena v manuálu *„The Unix Programming Environment“,* který napsali pan B.W. Kernighan a R. Pike.):

```
Great fleas have little fleas<Enter>
  upon their backs to bite 'em,<Enter>
And little fleas have lesser fleas<Enter>
  and so and infinitum.<Enter>
And great fleas themselves, in turn,<Enter>
  have greater fleas to go on;<Enter>
While these again have greater still,<Enter>
  and greater still, and so on.<Enter>
<Esc>
```

Všimněte si, že k ukončení vkládaného textu se používá klávesa Esc. Pak editor opět přejde do příkazového módu.

Příkazy pro pohyb kurzoru

- `h` posunutí kurzoru o jeden znak doleva
- `j` posunutí kurzoru o jeden řádek dolů
- `k` posunutí kurzoru o jeden řádek nahoru
- `l` posunutí kurzoru o jeden znak doprava

Uvedené příkazy mohou být opakovány tak, že danou klávesu budete držet stisknutou. Nyní si vyzkoušejte pohyb kurzoru všemi směry. Pokud se pokusíte posunout kurzor na neexistující pozici (například když stisknete klávesu K a přitom je kurzor na prvním řádku souboru), pak obrazovka blikne nebo se ozve zvukový signál. Ničeho se neobávejte, váš soubor nebude v takovém případě nijak poškozen.

Jak rušit text

- `x` zrušení znaku na pozici kurzoru
- `dd` zrušení řádku

Posuňte kurzor na druhý řádek a nastavte jeho pozici pod apostrof ve slově `'em`. Stiskněte klávesu X a apostrof zmizí. Nyní stiskněte klávesu I (přepnete tak editor `vi` do módu, ve kterém se vkládá text) a zapište „`th`“. Nakonec stiskněte klávesu Esc.

Uložení souboru

- `:w` uložení souboru (na disk)
- `:q` ukončení editoru `vi`

Nejdříve se přesvědčte, že jste v příkazovém módu – stiskněte klávesu Esc. Nyní zadejte `:w`. V tomto okamžiku se vámi vytvořený text uloží do diskového souboru.

Příkaz pro ukončeni editoru `vi` je reprezentován příkazem `:q`. Jestliže chcete zkombinovat uložení souboru a ukončení editoru, pak použijte příkaz `:wq`. Pro příkaz `:wq` existuje ekvivalentní zkratka `ZZ`. Zkratka `ZZ` se bude hodit zejména programátorům. Podívejme se na typickou posloupnost akcí, které provádí programátor při ladění programu: spuštění programu; zjištění problému; editování souboru obsahujícího zdrojový kód programu; provedení změn v souboru a jeho uložení na disk; přeložení programu; spuštění programu; ...a tak stále dokola. Pak bude programátor příkaz `ZZ` zřejmě používat velmi často. Ve skutečnosti není příkaz `ZZ` přesným ekvivalentem příkazu `:wq`. Příkaz `ZZ` totiž neprovede uložení souboru, pokud v textu neprovedete žádné změny, zatímco příkaz `:wq` soubor před ukončením editoru soubor vždy uloží.

Jestliže jste udělali nějaké změny, ale pak nechcete soubor ukládat, použijte příkaz `:q!` (nezapomeňte předtím stisknout klávesu Esc). Pokud byste zapomněli znak „!“ editor `vi` vám nepovolí ukončit práci bez uložení souboru.

Co bude následovat

Deset příkazů, které jsme v předcházejících odstavcích probrali, stačí k tomu, abyste byli schopni pracovat s editorem vi. Tyto příkazy však představují pouze základ práce s editorem. Existují další příkazy, například pro kopírování textu z jednoho místa na druhé, pro přesouvání textu z jednoho souboru do druhého, pro konfiguraci editoru a podobně. V editoru vi lze aplikovat asi 150 příkazů.

Pokročilejší techniky práce s editorem vi

Velká výhoda editoru `vi` spočívá v tom, že jej můžete používat, i když znáte jen několik málo základních příkazů. Většina uživatelů je nadšena, že stačí znát jen pár příkazů, ale jakmile s editorem pracují déle, potřebují k realizaci některých úloh další příkazy.

V následujícím oddílu se předpokládá, že se uživatel seznámil se základními příkazy uvedenými v předcházejícím oddílu. Nyní si probereme další příkazy – od kopírování textů až po definici maker.

Také uvedeme oddíl o konfiguraci editoru `vi`, kde se dozvíte, jak nastavit některé vlastnosti editoru, aby byla vaše práce co nejefektivnější. Následující odstavce jsou zaměřeny spíše na popis příkazů a nejsou již tolik zaměřeny na jejich procvičování. Proto doporučujeme, abyste si probírané příkazy průběžně procvičovali sami.

Nebudeme uvádět zcela vyčerpávající seznamy příkazů editoru `vi`, ale zaměříme se na příkazy nejčastěji používané zejména z praktického hlediska. I když si nakonec zvolíte pro svou běžnou práci jiný editor, budou se vám nabyté znalosti vždy hodit.

Příkazy pro změnu polohy kurzoru

K nejzákladnějším funkcím editoru patří příkazy pro změnu polohy kurzoru. Zde uvádíme jejich přehled.

h	posunutí kurzoru o jeden znak doleva
j	posunutí kurzoru o jeden řádek dolů
k	posunutí kurzoru o jeden řádek nahoru
l	posunutí kurzoru o jeden znak doprava

Některé implementace editoru vi také umožňují používat kurzorové klávesy.

w	posunutí kurzoru na začátek následujícího slova
e	posunutí kurzoru na konec následujícího slova
E	posunutí kurzoru na konec následujícího slova před mezeru
b	posunutí kurzoru na začátek předcházejícího slova
0	posunutí kurzoru na začátek řádku
^	posunutí kurzoru na první slovo v aktuálním řádku
$	posunutí kurzoru na konec řádku
<CR>	posunutí kurzoru na začátek následujícího řádku
-	posunutí kurzoru na začátek předcházejícího řádku
G	posunutí kurzoru na konec souboru
1G	posunutí kurzoru na začátek souboru
nG	posunutí kurzoru na řádek s pořadovým číslem n
Ctrl-Shift-G	zobrazení čísla aktuálního řádku
%	posunutí kurzoru na odpovídající hranatou závorku
H	posunutí kurzoru na první řádek obrazovky
M	posunutí kurzoru na prostřední řádek obrazovky
L	posunutí kurzoru na spodní řádek obrazovky
nl	posunutí kurzoru na sloupec n

Jestliže kurzor dospěje na spodní nebo horní řádek obrazovky, pak se zobrazený text vždy příslušným směrem posune. Nyní uvedeme alternativní příkazy, které umožňují posouvání zobrazovaného textu na obrazovce (tzv. rolování).

Ctrl-f	posunutí textu o stránku nahoru
Ctrl-b	posunutí textu o stránku dolů
Ctrl-d	posunutí textu o půl stránky dolů
Ctrl-u	posunutí textu o půl stránky nahoru

Právě uvedené příkazy jsou určeny k pohybu kurzoru. Některé z nich mohou být modifikovány pomocí čísla, které se uvede před příkaz. Předřazené číslo n znamená, že se příkaz bude opakovat n-krát.

Uveďme si tvar příkazu pro posunutí kurzoru o n pozic doleva:

nl	posunutí kurzoru o n pozic (znaků) doprava

Jestliže chcete například zařadit před text větší počet mezer, můžete použít podobnou modifikaci příkazu i pro vkládání textu. Zadejte počet mezer, pak příkaz i následovaný mezerou a nakonec stiskněte klávesu ESC.

ni opakované vložení textu n-krát

Příkazy odkazující na řádky mohou jako modifikátor použít číslo řádku. Ukázkovým příkladem je příkaz G:

1G posunutí kurzoru na první řádek textového souboru

Editor `vi` má velké množství příkazů, které lze použít ke změně pozice kurzoru – od jednoduchých příkazů, kde se kurzor posune o jednu pozici, až po složitější příkazy, kdy se kurzor nastavuje na předem specifikovanou pozici. Také lze zadat nastavení kurzoru na specifikovaný řádek při spouštění editoru, například:

```
vi +10 myfile.tex
```

Uvedený příkaz otevře soubor myfile.tex a umístí kurzor na desátý řádek od začátku souboru.

Vyzkoušejte si příkazy uvedené v tomto oddíle a procvičte se v jejich používání. Většina uživatelů používá pouze jistou podmnožinu uvedených příkazů. V dalším oddílu si probereme příkazy umožňující text měnit.

Modifikace textu

Nyní je naším cílem měnit obsah textového souboru a editor vi za tímto účelem nabízí spoustu příkazů.

V tomto oddíle budeme probírat příkazy určené k editování textu, rušení textu a podobně. Až oddíl dočtete, budete mít dostatek znalostí potřebných k vytvoření jakéhokoliv textového souboru. Následující oddíly jsou pak zaměřeny na další užitečné příkazy.

Při vkládání textu lze vložit více textových řádků prostřednictvím klávesy Enter. Předpokládejme, že jste udělali chybu a že jste stále na řádku, ve kterém se chyba vyskytuje.

Pak můžete použít klávesu Backspace a vrátit se k místu, kde se nachází chyba. Pokud jde o klávesu Backspace, různé implementace editoru vi se chovají různě. Některé z nich pouze posunují kurzor zpět a zapsaný text nechávají nedotčený. Jiné text postupně zprava ruší. Některé implementace dokonce umožňují používat v módu zadávání textu kurzorové klávesy. To vše však nepatří k normálnímu chování editoru `vi`. Pokud je text viditelný a použijete klávesu Esc (přitom jste na řádku, ve kterém jste použili klávesu Backspace), pak budou znaky za kurzorem zrušeny. Funkci klávesy Backspace si budete muset vyzkoušet, abyste zjistili, jak se tato klávesa ve vaší konkrétní implementaci editoru vi chová.

a	Přidání textu od aktuální pozice kurzoru
A	Přidání textu na konec řádku
i	Vložení textu vlevo od pozice kurzoru
I	Vložení textu vlevo od prvního výskytu zobrazitelného znaku (non-white character) v aktuálním řádku
o	Otevření nového řádku a vložení textu za aktuální řádek
O	Otevření nového řádku a vložení textu před aktuální řádek

Nyní máme několik příkazů pro vkládání textu a dále si uvedeme příkazy pro zrušení textu. Editor `vi` má několik příkazů pro zrušení textu, které mohou být spojeny s modifikátorem.

x	zrušení znaku, na jehož pozici je umístěn kurzor
dw	zrušení textu od aktuální pozice kurzoru do konce slova
dd	zrušení aktuálního řádku
D	zrušení textu od aktuální pozice kurzoru do konce řádku

Prostřednictvím modifikátorů se síla příkazů zvýší. Následující příklady jsou pouze podmnožinou všech možností:

nx	n-krát opakované zrušení znaku, na jehož pozici je umístěn kurzor
ndd	zrušení n řádků
dnw	zrušení n slov (stejnou funkci má příkaz ndw)
dG	zrušení textu od aktuální pozice kurzoru do konce souboru
d1G	zrušení textu od aktuální pozice kurzoru do začátku souboru
d$	zrušení textu od aktuální pozice kurzoru do konce řádku
dn$	zrušení textu od aktuální pozice kurzoru do konce n-tého řádku

Uvedené příklady ukazují, že operace zrušení textu mohou být velmi efektivní. Je to zejména evidentní tehdy, když tyto operace použijete ve spojení s příkazy pro změnu polohy kurzoru. Poznamenejme, že příkaz `D` tvoří výjimku, protože ignoruje jakékoli modifikátory.

Příležitostně nastanou situace, kdy budete chtít provedené změny vrátit zpět. Následující příkazy jsou určeny pro obnovení textu:

u	zrušení posledně provedeného příkazu
U	zrušení všech změn provedených v tomto řádku
:e!	obnova stavu souboru od okamžiku, kdy byl naposledy uložen

Editor `vi` umožňuje nejen vracet zpět provedené změny, ale také vracet zpět vrácené změny. Například pomocí příkazu `5dd` zrušíte pět řádků a pak je obnovíte pomocí příkazu `u`. Když použijete příkaz `u` znovu, budou řádky opět zrušeny.

Nové verze editoru `vi` nabízejí příkazy, které umožňují editovat text v tzv. přepisovacím módu. Znamená to, že chcete-li měnit nějaký text, pak jej nemusíte nejdříve rušit.

r c	přepsání znaku na pozici kurzoru znakem
R	přepsání textu novým textem
c w	změna textu v aktuálním slově
c $	změna textu od aktuální pozice kurzoru do konce řádku
c n w	změna následujících n slov (totéž jako ncw)
c n $	změna do konce n-tého řádku
C	změna do konce řádku (totéž jako c$)
c c	změna aktuálního řádku
S	v textu, který právě píšete, nahradí aktuální znak
n s	v textu, který právě píšete, nahradí n aktuálních znaků

Série příkazů realizujících změny vám umožní zadat řetězec znaků, který musí být ukončen klávesou Esc.

Příkaz c w platí od aktuální pozice kurzoru ve slově do konce slova. Když použijete příkaz realizující změnu a přitom zadáte „vzdálenost“, editor `vi` zobrazí znak `$` na pozici, do které bude změna provedena. Nový text může být delší nebo kratší než text původní.

Kopírování a přesouvání částí textu

Pro přesouvání textu existuje řada příkazů, jejichž kombinací lze dosáhnout kýženého efektu. V tomto oddíle popíšeme pojmenované a nepojmenované buffery spolu s příkazy umožňujícími „vystřihnout“ a „nalepit“ text.

Základní kopírování textu probíhá ve třech krocích:

1. Kopírování textu do bufferu („**vystřižení**“ textu).
2. **Nastavení kurzoru** na cílovou pozici.
3. Kopírování textu z bufferu na novou pozici („**nalepení**“ textu).

K vystřižení textu do nepojmenovaného bufferu použijte některý z následujících příkazů:

y y	Přesunutí kopie aktuálního řádku do nepojmenovaného bufferu.
Y	Přesunutí kopie aktuálního řádku do nepojmenovaného bufferu.
c y y	Přesunutí kopie následujících n řádků do nepojmenovaného bufferu.
c Y	Přesunutí kopie následujících n řádků do nepojmenovaného bufferu.
y w	Přesunutí slova do nepojmenovaného bufferu.
y n w	Přesunutí n slov do nepojmenovaného bufferu.
n y w	Přesunutí n slov do nepojmenovaného bufferu.
y $	Přesunutí textu od aktuální pozice do konce řádku.

Nepojmenovaný buffer je dočasná oblast paměti, jejíž obsah může být zrušen prostřednictvím jiných často používaných příkazů. Někdy se stane, že jistou část textu budete potřebovat po dlouhou dobu. V takovém případě byste měli použít pojmenovaný buffer. Editor `vi` nabízí 26 pojmenovaných bufferů. Jako identifikační jméno bufferu se používá jednoznakové písmeno. Pro rozlišení pojmenovaného bufferu editor `vi` používá znak ". Při odkazu na pojmenovaný buffer se používají malá písmena (pokud má být obsah bufferu zcela nahrazen) nebo velká písmena (pokud má být obsah bufferu doplněn). Uveďme si příklady:

" a y y	Přesunutí aktuálního řádku do pojmenovaného bufferu a.
" a Y	Přesunutí aktuálního řádku do pojmenovaného bufferu a.
" b y w	Přesunutí aktuálního slova do pojmenovaného bufferu b.
" B y w	Přidání aktuálního slova k obsahu pojmenovaného bufferu b.
" b y 3 w	Přesunutí následujících tří slov do pojmenovaného bufferu b.

Ke zkopírování („nalepení“) obsahu bufferu použijte příkaz `p`:

p	Zkopírování obsahu nepojmenovaného bufferu VPRAVO od pozice kurzoru.
P	Zkopírování obsahu nepojmenovaného bufferu VLEVO od pozice kurzoru.
n P	Zkopírování obsahu nepojmenovaného bufferu n-krát VLEVO od pozice kurzoru.
" a p	Zkopírování obsahu pojmenovaného bufferu a VPRAVO od pozice kurzoru.
" b 3 P	Zkopírování obsahu pojmenovaného bufferu b třikrát VLEVO od pozice kurzoru.

Jestliže používáte editor `vi` v terminálovém okně systému X Window, pak máte k dispozici ještě jeden prostředek pro kopírování textů. Oblast textu, kterou chcete kopírovat, vyznačte kurzorem myši (levé tlačítko myši přitom držte stisknuté). Vyznačená oblast textu bude zobrazena inverzně a přitom se automaticky přenese do speciálního bufferu vyhrazeného pro systém X Window. Po-

kud chcete text „nalepit“, stačí stisknout prostřední tlačítko myši. Nezapomeňte, že předtím musíte editor vi přepnout do vkládacího módu, protože jinak by mohl být vstup interpretován jako příkaz a výsledek by byl nepředvídatelný. Prostřednictvím stejné techniky lze kopírovat jednotlivá slova. Slovo se přenese do bufferu dvojnásobným klepnutím levým tlačítkem myši. Obsah bufferu se změní pouze tehdy, když se vyznačí nová oblast textu.

Přesouvání textu se rovněž odehrává ve třech krocích:

1. **Zrušení textu** a jeho současné zkopírování do pojmenovaného nebo nepojmenovaného bufferu.
2. **Nastavení kurzoru** na cílovou pozici.
3. **Kopírování textu** z pojmenovaného nebo nepojmenovaného bufferu na novou pozici („nalepení“ textu).

Proces je stejný jako kopírování textu, avšak v prvním kroku se text zruší. Když se provede příkaz `dd`, přesune se zrušený řádek do nepojmenovaného bufferu. Pak můžete obsah bufferu „nalepit“ stejným způsobem, jako při kopírování textu.

" a d d	zrušení řádku a přesunutí do pojmenovaného bufferu a
" a A d d	zrušení čtyř řádků a přesunutí do pojmenovaného bufferu a
d w	zrušení slova a přesunutí do nepojmenovaného bufferu. Další příklady na zrušení textu jsme uvedli v oddíle o modifikaci textu.

V případě, že systém zhavaruje, obsahy pojmenovaného i nepojmenovaných bufferů se ztratí. Obsah editovaného bufferu však může být obnoven.

Vyhledávání a nahrazování textů

Editor `vi` má spoustu příkazů pro vyhledávání řetězců. Můžete vyhledávat jednotlivé znaky, ale také můžete při vyhledávání použít regulární výrazy.

Hlavní vyhledávací příkazy jsou `f` a `t`:

f c	vyhledání následujícího znaku c vpravo od kurzoru
F c	vyhledání následujícího znaku c vlevo od kurzoru
t c	přesunutí kurzoru na pozici vlevo od následujícího znaku c
T c	přesunutí kurzoru na pozici vpravo od předcházejícího znaku c (V některých verzích editoru vi je tento příkaz shodný s příkazem Fc.)
;	opakování posledního příkazu f, F, t nebo T
,	stejný příkaz jako ;, ale mění směr vyhledávání původního příkazu

Jestliže hledaný znak neexistuje, upozorní vás editor `vi` zvukovým nebo jiným signálem.

Editor `vi` rovněž umožňuje v editovaném textu hledat řetězec:

/`str`	vyhledání specifikovaného řetězce od aktuální pozice kurzoru
?`str`	vyhledání specifikovaného řetězce před aktuální pozicí kurzoru
n	opakování předcházejícího příkazu / nebo ?
N	opakování předcházejícího příkazu / nebo ?, přičemž se změní směr vyhledávání

Když použijete příkaz / nebo ?, „vyčistí“ se spodní řádek obrazovky. Pak v tomto řádku zadáte hledaný řetězec a stisknete klávesu Enter.

Řetězec uvedený za příkazy / a ? může být regulárním výrazem. V regulárním výrazu se prostřednictvím pseudoznaků popisují jisté množiny znaků. Pseudoznaky používané v editoru `vi` jsou následující: `.`, `*`, `[`, `]`, `^` a `$`. Následující seznam specifikuje význam pseudoznaků:

`.`	vyhovuje jakýkoliv znak kromě znaku „nového řádku“
`\`	uvozuje jakýkoliv speciální znak
`*`	vyhovuje výskytu předcházejících znaků
`[]`	vyhovuje právě jeden znak uvedený v hranatých závorkách
`^`	vyhovuje právě jeden znak, jen pokud se nachází na začátku řádku
`$`	vyhovuje znak předcházející konci řádku
`[^]`	vyhovují znaky, které nejsou uvedeny v hranatých závorkách
`[-]`	vyhovují znaky z uvedeného rozsahu

Nejjednodušší způsob, jak se naučíte používat regulární výrazy, spočívá v tom, že je budete používat. Vyzkoušejte si následující příklady:

`c.pe`	vyhovují slova jako cope, cape, caper a podobně
`c\.pe`	vyhovují slova jako c.pe, c.per a podobně
`sto*p`	vyhovují slova jako stp, stop, stoop a podobně
`cat.*n`	vyhovují slova jako carton, cartoon a podobně
`xyz.*`	vyhovují xyz do konce řádku
`^The`	vyhovují všechny řádky začínající na The

`atime$`	vyhovují všechny řádky končící na atime
`^Only$`	vyhovují všechny řádky, ve kterých se vyskytuje pouze slovo Only
`b[aou]rn`	vyhovují slova barn, born nebo burn
`Ver[D-F]`	vyhovují slova VerD, VerE nebo VerF
`Ver[^1-9]`	vyhovují slova začínající na Ver, po kterých nenásleduje žádná číslice
`the[ir][re]`	vyhovují slova their, therr, there a theie
`[A-Za-z][A-Za-z]*`	vyhovuje jakékoliv slovo

Editor vi používá příkazový mód `ex` k realizaci operací vyhledávání a náhrady řetězců. Všechny příkazy začínající dvojtečkou představují požadavek v módu `ex`.

Příkazy pro vyhledání a náhradu řetězce umožňují, aby byly realizovány v jistém řádkovém rozsahu. Uživatel může vyžadovat, aby byl před náhradou řetězce vyzván k potvrzení, zda se má konkrétní výskyt řetězce nahrazovat nebo ne. Uveďme si obecný tvar příkazu pro náhradu řetězce a několik příkladů:

...syntaxe obecného příkazu :<start>,<finisn>s/<find>/<replace>/g

:1,$s/the/The/g	vyhledá všechny výskyty slova the a nahradí je slovem The
:%s/the/The/g	znak % znamená „celý soubor“. Tento příkaz provede stejnou funkci jako předcházející příkaz
:.,5s/^.*//g	zruší obsah editovaného souboru od aktuálního řádku po pátý řádek
:%s/the/The/gc	nahradí všechny výskyty slova the slovem The a před každou náhradou si vyžádá potvrzení
:%s/^..../g	zruší první čtyři znaky každého řádku

Jak je z předcházejících příkladů patrné, jsou příkazy pro vyhledávání a náhradu řetězců velmi výkonné, zejména když se kombinují s regulárními výrazy. Pokud se direktiva `g` v příkazu neuvede, provede se náhrada pouze u prvního výskytu hledaného řetězce.

Někdy se může stát, že původní vyhledávaný řetězec se má použít v nahrazujícím řetězci. Za tímto účelem nabízí editor `vi` některé speciální znaky:

:1,5s/help/&ing/g	v prvních pěti řádcích nahradí slovo help slovem helping
:%s/ */&&/g	v celém souboru zdvojnásobí počet mezer mezi slovy

Používání kompletního řetězce má v editoru `vi` jisté omezení, protože k stanovení rozsahu náhrady editor používá kulaté závorky `( a )`. V takovém případě je nutno použít znak `\`:

:s/^\(.*\):.*/\1/g	zruší vše, co následuje po dvojtečce včetně dvojtečky samé
:s/\(.*\):\(.*\)/\2:\1/g	vymění slova na obou stranách dvojtečky

Posledně uvedené příklady asi budete muset číst velmi pozorně. Editor `vi` nabízí velmi výkonné příkazy, které moderní editory zpravidla nemají. Za to se ovšem platí jistá cena – naučit se všechny příkazy editoru vi není vůbec snadné. Pokud jsme vás neodradili, pokuste se znovu si projít uvedené příklady a uvidíte, že se používání editoru `vi` pro vás stane zcela přirozenou záležitostí.