Obsah

1 Úvod a historie Unixových systémů

Historie OS UNIX

- V roce 1969 vznikl v Bell Laboratories operační systém Multics
- Vzhledem k některým nevýhodám tohoto systému došlo k odštěpení části vývojového týmu a začal vznikat systém UNIX. Poprvé byl implementován na počítačích PDP firmy DEC.
- V roce 1973 byl UNIX přepsán o jazyka C (který vznikl souběžně s vývojem Unixu), pouze nezbytně nutné části zůstaly v assembleru.
- Následoval vývoj 2 hlavních větví
  - v Bell Laboratories – označovaná aktuálně System V
  - na univerzitě v Berkeley – označovaná BSD
- Vznik normy POSIX – pro systémy odvozené od Unixu
- V současnosti několik desítek systémů odvozených převážně od UNIX System V, např. AIX od IBM, HP-UX od Hewlett Packard, Solaris od Sun Microsystems, UnixWare od Novellu, Xenix od Microsoftu a další. Převážně se jedná o komerční systémy, které jsou určeny pro použití ve větších podnicích.

Volně šiřitelné verze Unixu

- OpenBSD a NetBSD – odvozených od systému z univerzity v Berkeley
- Linux – inspirovaný systémem Minix (varianta UNIX System V)

Základní vlastnosti Unixu

- interaktivní
- podpora multitaskingu – běh více procesů zároveň
- víceuživatelský
  - rozlišení jednotlivých uživatelů a jasná specifikace přístupových práv
  - podpora současné práce více uživatelů najednou
- přenositelný (jedná se o první systém implementovaný ve vyšším programovacím jazyce, kterým jazyk C je)
- základní systém neobsahuje grafické rozhraní – možnost plnohodnotně spravovat systém i vzdáleně

- grafické rozhraní X-Windows je řešeno jako volitelná součást, nezávislá na samotném systému
- jeden adresářový strom – princip *mountování* - připojování diskových jednotek do podadresářů hlavního stromu (lze i za běhu)
- jednotný přístup k zařízením prostřednictvím speciálních souborů
- stručnost – většina programů v případě že se nevyskytne chyba nevypisuje nic

Historie Linuxu

- vznik projektu GNU (založil Richard M. Stallman v roce 1984) a licence GPL[1]
- v rámci projektu GNU vznikla nadace FSF[2], která měla cíl vytvořit klon unixu postavený na svobodném software
- v roce 1991 začal finský student Linus Torvalds, pracovat na vývoji vlastního jádra unixového systému založenému na Minixu, které mělo odstraňovat nevýhody tohoto systému
- později se kolem Linuse rozrostla komunita vývojářů pracující na novém systému, jádro bylo zařazeno do projektu nadace FSF a spojením s nástroji vyvinutými touto nadací vznikl systém GNU/Linux
- dnes se na vývoji Linuxu podílí rozsáhlá komunita vývojářů z celého světa, kteří na vývoji systému dobrovolně pracují a Linux se stal plnohodnotnou náhradou komerčních uzavřených řešení

Linux

- samotný Linux tvoří tzv. jádro (kernel) a je vždy zdarma
- šířen v distribucích – distribuce obsahuje jádro, základní obslužné programy a další software
- dnes existuje velké množství distribucí, některé zdarma, případně s možností připlatit za tištěný manuál nebo podporu, jiné od základu komerční (např. RedHat Enterprise Linux)

[1] General Public Licence - licence, pod kterou je vydáván tzv. svobodný software (Free Software) - podle licence je možné software dále libovolně šířit nebo využít kteroukoliv část zdrojového kódu ve vlastním projektu, za podmínky že bude také šířen pod licencí GPL, tj. v otevřené formě včetně zdrojových kódů

[2] Free Software Foundation

2 Základy práce se systémem

Přihlašování

- na výzvu `login:` zadat uživatelské jméno
- po stisku enteru se zobrazí dotaz na heslo, které nebude zobrazováno
- po přihlášení je uživatel v *domovském adresáři*
- odhlášení stiskem `Ctrl-D` na prázdném řádku (může být zakázáno, pak příkaz `logout` nebo `exit`)
- uživatel nemusí mít heslo, což ale představuje bezpečnostní riziko
- v každém systému existuje uživatel **root** (superuživatel), který má neomezená práva – je určen pro administrátora systému

passwd – změna hesla

who, w, finger – seznam přihlášených uživatelů

last, lastb – výpis naposled přihlášených uživatelů, resp. neúspěšných pokusů o přihlášení

Terminál

Terminál se označuje vstupně/výstupní zařízení určené pro komunikaci s uživatelem – nejčastěji obsahující klávesnici a monitor (dříve také tiskárna) a případně další hardware umožňující vzdálenou komunikaci (sériová linka, ethernet) a interpretaci speciálních znaků (terminály umožňují řízení kurzoru a nastavování některých zobrazovacích parametrů pomocí speciálních escape sekvencí které se nezobrazují.

Emulátor terminálu program, běžící na klasickém počítači, emulující chování terminálu (escape sekvence)

Virtuální terminály emulace více terminálů přímo na počítači kde běží systém, umožňují vícenásobné přihlášení
v Linuxu je možnost až 12 virtuálních terminálů, mezi kterými se přepíná pomocí kláves `Alt-F1` až `Alt-F12`, případně `Ctrl-Alt-F1` až `Ctrl-Alt-F12`, pokud klávesové kombinace bez Ctrl již nějaká aplikace používá

Dokumentace

Jádrem dokumentace jsou manuálové stránky, které jsou rozřazeny do několika sekcí:

sekce	obsah
1	Spustitelné programy a příkazy shellu
2	Systémové volání (funkce kernelu)
3	Knihovní funkce
4	Speciální souboru (nachází se v `/dev`)
5	Formáty souborů (např. `/etc/passwd`
6	Hry
7	Různé
8	Příkazy pro systémovou administraci
9	Kernel routines [Non standard]

man – popis k programu zadanému jako argument

whatis – hledá v názvu příkazů

apropos – hledá v názvu i popisu příkazů

info, pinfo – podobné jako `man`, ale obsahuje hypertextové odkazy

3 Práce v systému

3.1 Základy práce v shellu

Shell je příkazový interpret – obdoba `command.com` z DOSu nebo `cmd.exe` z Windows, ale s většími možnostmi.

Shell se v interaktivním režimu hlásí promptem, kterým signalizuje připravenost k zadání příkazu. Prompt obvykle končí znakem `$` a mezerou pro běžného uživatele a # pro superuživatele. V případě odeslání nedokončeného příkazu (neuzavřené uvozovky) nebo zpětného lomítka na konci řádky se zobrazí tzv. pokračovací prompt, implicitně `>`.

Příkazy se zadávají ve formátu:

```
příkaz [ volby ] [ argumenty ]
```

Argumenty i volby jsou většinou nepovinné. Pokud je parametr soubor a vynechá se, pracuje se většinou se standardním vstupem (klávesnice) nebo standardním výstupem (obrazovka) → možnost řetězení příkazů do *kolony*.

Volby začínají znakem mínus a ovlivňují chování příkazu. Lze řetězit více voleb za jeden mínus nebo je uvést zvlášť (`ls -l -a` je ekvivalentní `ls -la`) volba `-h`, případně `--help` vypíše stručně syntaxi argumentů

Vnitřní příkazy jsou součástí shellu

Vnější příkazy jsou externí programy ve stejnojmenném souboru, hledají se v definovaných adresářích

Existuje více různých shellů, které se liší hlavně syntaxí vnitřních příkazů a různými vylepšeními. Základní shell je Bourne shell (sh), dále existuje C-shell (csh), Korn shell (ksh). V Linuxu je základním shellem **bash** (Bourne Again SHell).

Speciální klávesy

klávesa	význam
`Tab`	doplnění neúplného jména souboru nebo adresáře, dvojí stisk vypíše možnosti
`Ctrl-D`	konec vstupu, pokud je první znak na řádce; jinak zmaže znak na pozici kurzoru
`Ctrl-U`	smaž text před kurzorem od začátku řádku
`Ctrl-K`	smaž text od kurzoru do konce řádku
`Ctrl-R`	zpětné inkrementální hledání v historii příkazů od aktuální pozice
`Ctrl-S`	dopředné inkrementální hledání v historii příkazů od aktuální pozice
`Ctrl-G`	zrušení vyhledávání a návrat na původní pozici
`Esc`	zrušení vyhledávání a zachování stávající pozice

Za zmínku stojí příkaz `history`, který vypíše zapamatované, dosud zadané příkazy.

3.2 Příkazy pro práci s adresáři a soubory

Jako oddělovač adresářů a souboru v cestě se používá znak `/`, který je jako jediný zakázaný uvnitř jména souboru nebo adresáře. Každý adresář obsahuje hned po založení 2 položky „`.`“ a „`..`“. První je odkaz „sám na sebe“ a druhý na nejbližší nadřazený adresář.

Absolutní cesta začíná lomítkem a obsahuje postupně lomítky oddělené jména adresářů od kořenového až k danému souboru nebo adresáři.
např.: `/home/petr/dokumenty/zprava.txt`

Relativní cesta nezačíná lomítkem, specifikuje postupně adresáře, kterými je potřeba projít od aktuálního adresáře až k danému souboru nebo adresáři. např.:
`prog/c/astar.c` nebo `../java/minimax.java`

`ls [ `*`adresář`*` | `*`soubor`*` ]` – výpis aktuálního adresáře
- `-l` dlouhý výpis
- `-a` i soubory začínající tečkou
- `-A` i soubory začínající tečkou, kromě `.` a `..`
- `-R` rekurzivně
- `-1` v jednom sloupci
- `-t` seřazené podle času modifikace
- `-u` seřazené podle času posledního přístupu
- `-r` obrácené pořadí
- `-d` zobrazení informací o samotném adresáři (jinak vypíše jeho obsah)

`pwd` – zobraz aktuální adresář

`cd` – změna aktuálního adresáře (pokud chybí argument, nastaví se domovský)

`mkdir [ -p ]` ***`adresář`*** – vytvoření adresáře
- `-p` včetně celé cesty

`rmdir` ***`adresář`*** – zrušení adresáře (musí být prázdný)

`cp [ -fdpR -a ]` *`zdroj cíl`* – kopie souborů

`-f` force – přepíše cíl i když je read-only (pokud lze smazat)

`-d` no-dereference – kopíruje symbolické linky místo souboru, na který odkazují

`-p` preserve – zachová přístupová práva (jinak jsou ovlivněna uživatelskou maskou)

`-R` recursive – *zdroj* může být adresář – zkopíruje všechny soubory a adresáře, které obsahuje `-a` archive – kopie celého stromu na jiné místo se zachováním práv a symbolických linků

`mv [ -f ]` *`původní nový`* – přejmenování/přesun souborů

`-f` force

`rm [ -rf ]` *`soubor...`* – mazání souborů

`-r` recursive – smazání adresáře včetně podadresářů (nutno opatrně) `-f` force – smazat i když je soubor read-only (v opačném případě soubor odmítne smazat)

`ln [ -s ]` *`soubor cíl`* – vytvoření hard nebo symbolického odkazu

U příkazů `cp`, `mv`, `ln` a `rm` lze použít volbu `-i`, která vygeneruje dotaz před případným přepsáním nebo smazáním souboru.

du [-sh] [*adresář*] – disk usage – zobrazí celkový prostor obsazený adresářem a jeho podadresáři v bytech (implicitně aktuální)
`-s` jen celkový součet
`-h` zobrazí podle potřeby v KB, MB, GB
Př.:
`du -sh ~` – zobrazí objem zabraný domácím adresářem

file *soubory* – identifikuje typ souboru podle prvních 512 bajtů (textový/binární, kódování textu, druh programovacího jazyka, typ binárního souboru, atd.)

find *poč_adresář kritéria* – hledání souborů podle zadaných kritérií:
`-name` *jméno* – soubor daného jména, může být i částečné za pomoci zástupných znaků * a ? (pak je nutno jméno uzavřít do apostrofů aby nedošlo k interpretaci shellem)
`-iname` *jméno* – dtto, ale nerozlišuje velikost písmen
`-user` *uživatel* – vlastník souboru je *uživatel*
`-perm` *práva* – hledání podle práv (oktalově)
`-mmin` -*n* – soubor byl změněn během posledních *n* minut
`-mtime` -*n* – soubor byl změněn během posledních *n* dní
`-type` *typ_souboru* – soubory daného typu (viz dále tabulka typů souboru)
při uvedené více podmínek najednou musí být splněny všechny, vykřičník před podmínkou znamená negaci

locate *soubor* – rychlé hledání souboru podle názvu, využívá databázi, která se automaticky aktualizuje většinou jednou denně

df – disk free, zobrazí volné místo na připojených svazcích

3.3 Připojování svazků

Svazek (volume) je logická oblast na úložném médiu, obsahující právě jeden *filesystém*. U architektury PC je většinou ekvivalentní jedné partition, případně skupině partition spojené pomocí RAID manageru nebo LVM (logical volume manager)

Filesystem je tvořen stromem adresářů a souborů (obyčejných i speciálních)

Kořenový adresář (Root) je v systému vždy právě jeden a značí se `/`. Během spouštění systému se určí který svazek se připojí na kořenový adresář. Tento svazek během činnosti systému nelze odpojit.

Připojení svazku se provádí příkazem
`mount` *`spec_soubor adresář_připojení`*, kde adresář je libovolný podadresář aktuálního adresářového stromu. Pokud adresář nebyl prázdný, dojde k *zakrytí* (znepřístupnění jeho obsahu až do odpojení svazku). Připojením se rozšíří aktuální adresářový strom a adresáře a soubory filesystemu v připojeném svazku.

Př.:

- `mount /dev/hda3 /home` připojí třetí primární partition na prvním IDE disku do adresáře `/home`
- `mount /dev/cdrom /media/cdrom` připojí medium v CD-ROM mechanice do adresáře `/media/cdrom`
- `mount /dev/fd0 /media/floppy -t vfat` připojí disketu v mechanice do adresáře `/media/floppy`, zároveň se specifikuje že na disketě je filesystém FAT s dlouhými jmény.

Odpojení svazku se provádí příkazem `umount` *`spec_soubor`* nebo `umount` *`adresář_připojení`*. Před odpojením nesmí žádný uživatel ani systémový proces pracovat se souborem nebo mít aktuální adresář v odpojované části stromu.

Př.:
`umount /dev/cdrom`

spec. soubor	s čím je spojen
`/dev/hda1`	první primární partition na master disku na primárním kanálu IDE řadiče
`/dev/hdb3`	třetí primární partition na slave na primárním kanálu IDE řadiče
`/dev/hdc5`	první logická partition na master na sekundárním kanálu IDE řadiče
`/dev/hdd6`	druhá logická partition na slave na sekundárním kanálu IDE řadiče
`/dev/sda1`	první primární partition na prvním SCSI disku
`/dev/sdb2`	druhá primátí partition na druhém SCSI disku
`/dev/cdrom`	CD-ROM mechanika (bývá symbolický odkaz)
`/dev/fd0`	první floppy mechanika

CD-ROM mechanika, disketa, USB flash disky apod. se zpravidla připojují do podadresáře v adresáři `/media` nebo `/mnt`. Pozor, před vyjmutím je potřeba médium vždy odpojit, jinak by mohlo dojít k nekonzistenci a ke ztrátě uložených dat!

3.4 Typy souborů

Typy souborů podle prvního znaku v dlouhém výpisu `ls -l`.

typ	popis
-	obyčejný soubor (regular file)
d	adresář (directory)
l	symbolický odkaz (symbolic link)
c	znakový speciální soubor (character special)
b	bloková speciální soubor (block special)
p	pojmenovaná roura (named pipe)
s	socket

Typy souborů z hlediska systému:

adresář – v podstatě soubor obsahující řadu dvojic *i-node* [3] a *jméno souboru*

symbolický odkaz – soubor obsahující cestu k jinému souboru, při požadavku o jakoukoliv standardní souborovou operaci pracuje systém se souborem, na který odkazuje

znakový speciální soubor – neobsahuje datové bloky, je spjat se znakovým zařízením (např. terminál, tiskárna, sériový port)

blokový speciální soubor – neobsahuje datové bloky, je spjat s blokovým zařízením (např. disk, CD-ROM, páska)

pevný odkaz (hardlink) – je to další jméno souboru (další odkaz na i-node souboru), hardlinky jsou narozdíl od symbolických linků vzájemně rovnocenné hardlinky nelze vytvářet na adresáře a na souboru ležící na jiné diskové oblasti

[3] i-node je speciální struktura obsahující všechny informace o souboru, kromě jména (vlastník, skupina, práva, datum a čas poslední modifikace a přístupu, délku a odkazy na datové bloky souboru; i-nodes jsou uložené mimo adresář, ve speciální vyhrazené části diskové oblasti

Samotný systém UNIX nerozlišuje soubory z hlediska obsahu – interpretace obsahu záleží čistě na aplikaci. Narozdíl některých jiných systémů se pro oddělování řádků nepoužívá sekvence CR/LF (ASCII kódy 13, 10), ale pouze LF (10) – pro konverzi existují programy `dos2unix` (odstraní znaky CR) a `unix2dos` (před každý znak LF přidá CR).

textový soubor – zdrojové kódy, prakticky všechny konfigurační soubory, většina uživatelských dat, apod. V UNIXu se dává přednost textovému uložení z důvodu jednodušších a univerzálnějších možností zpracování i za cenu mírně většího objemu dat – např databáze v CSV nebo podobném formátu, případně využití XML nebo jiného značkovacího jazyka pro složitější konfigurační soubory.

binární soubor – programy v binárním kódu, Java bytecode, zkomprimované archivy, obrázky, videosekvence, apod.

3.5 Přístupová práva

Základní definice

- každý uživatel je jednoznačně identifikován svým uživatelským jménem
- uživatel může během relace se systémem přejít na identifikaci jiným uživatelem bez nutnosti odhlášení, pokud zná jeho heslo (příkaz `su`)
- v systému existují skupiny, uživatelů, každá skupina je také jednoznačně identifikována svým jménem
- každý uživatel je členem jedné nebo více skupin, ale v danou chvíli vždy vystupuje pod jednou konkrétní skupinou
- skupinu, pod kterou vystupuje může uživatel během relace změnit, ale jen na jinou skupinu, již je členem (příkaz `newgrp`, případně `sg`)
- Každý soubor nebo adresář má definovaného uživatele vlastnícího soubor a skupinu uživatelů (jsou nastaveny podle uživatele, který soubor vytvořil a jeho skupiny.
- Vlastníka souboru může změnit jen superuživatel (příkazem `chown`
- skupinu souboru může změnit jeho vlastník, ale jen v rámci skupin, jichž je členem; superuživatel může měnit skupinu libovolně

Výpis seznamu uživatelů: `getent passwd`
Výpis seznamu skupin: `getent group`

Přístupová práva, vlastník a skupina jsou zobrazeny dlouhém výpisu `ls -l`. Jsou rozdělena do 3 skupin – první skupina se aplikuje, pokud k souboru přistupuje jeho vlastník, druhá pokud je uživatel ve stejné skupině, kterou má soubor přiřazen a třetí v ostatních případech.
Každá skupina práv obsahuje 3 práva (definované třemi bity) – Read, Write, eXecute.

Dále existují speciální práva, definované dalšími třemi bity. Mají využití např. u programů, které operují se soubory, ke kterým nemá běžný uživatel přístup (SetUID bit) – např. program `passwd` nebo u sdílených adresářů (Sticky bit) – např. `/tmp`.

Změna práv se provádí příkazem `chmod` *`práva soubory`*, práva se zadávají buď symbolicky nebo jako oktalový bitový součet pomocí 4 číslic v pořadí spec. práva, vlastník, skupina, ostatní.

Př.:
`chmod u+x,g+w,u-rx` *`soubor`* – přidá uživateli právo na spouštění, skupině právo na zápis a ostatním odebere práva na čtení a spouštění
`chmod 644` *`soubor`* – nastaví uživateli práva na čtení a zápis (6 = 4 + 2) a ostatním pouze právo ke čtení, přičemž nezáleží na původních právech souboru

právo	oktalově	význam pro soubor	význam pro adresář
r	4	čtení souboru	zobrazení obsahu adresáře[4]
w	2	zápis do souboru[5]	modifikace položek adresáře[6]
x	1	spuštění souboru	průchod adresářem[7]
SetUID	4	spuštění pod vlastníkem	nemá význam
SetGID	2	spuštění pod skupinou	nastavení stejné skupiny u nových podadresářů
sTicky	1	zachování obrazu v paměti	smazat soubor může jen jeho vlastník

Z důvodu nedostatečné jemnosti základních příst. práv existují *rozšířená přístupová práva* (acl), viz. příkazy `getfacl` a `setfacl`.

[4] nezabrání průchodu, pokud známe obsah adresáře
[5] neovlivní možnost smazání
[6] vytváření, mazání a přejmenování souborů v adresáři
[7] použití uvnitř cesty nebo vstup pomocí `cd`

3.6 Základní příkazy pro zpracování textu

`echo [ -ne ]` ***`text ...`*** – vypíše na obrazovku postupně všechny slova zadané jako parametry, `-n` potlačení přechodu na nový řádek, `-e` interpretace speciálních znaků ve výpisu.

`cat [` ***`soubory`*** **`]`** (concatenate) – zobrazení souboru nebo více souborů za sebou

`head [-n], tail [-n]|[+n]` – zobrazení začátku/konce souboru

`more`, **`less`** – stránkovaný výpis textového souboru, less lépe využívá možnosti terminálu – snadnější pohyb v souboru a navracení

`wc [ -lwc ] [` ***`soubory`*** **`]`** – spočítá počet řádek (l), slov (w) nebo znaků (c) v souborech

`grep -lvir` ***`vzor`*** **`[`** ***`soubory`*** **`]`** – prohledává vstupní soubory na výskyt zadaného vzoru a zobrazí vyhovující řádky, lze použít tzv. regulární výraz
`-l` zobrazí pouze jména souborů obsahující daný vzor
`-v` zobrazí řádky *nevyhovující* danému vzoru
`-i` nerozlišuje velká a malá písmena
`-r` prohledává i soubory v podadresářích (recursive)

`diff` – porovnání obsahu dvou textových souborů

3.7 Zástupné znaky v názvu souboru

Slouží k hromadnému výběru více souborů. Shell provede substituci všech vyhovujících souborů na místo vzoru.

znak	význam
*	jakýkoliv řetězec znaků (i prázdný)
?	právě jeden libovolný znak
[abcd]	výčet znaků, lze zadat i jako rozsah – [a-d]

např. `ab*` se nahradí všemi soubory začínajícími na ‘ab’ v aktuálním adresáři. Pokud takový není nahradí se prázdným řetězcem nebo k substituci nedojde (viz. `shopt nullglob`), vzoru `soub?` vyhoví `soubor1`, `soubor2`...`soubor9`, ale už ne `soubor10`

3.8 Přesměrování a kolona

Naprostá většina příkazů používá (prostřednictvím knihovních funkcí) tyto implicitně otevřené souborové deskriptory (*file handle*):

č. deskriptoru	název	význam
0	stdin	standardní vstup (klávesnice, pokud není přesměrován)
1	stdout	standardní výstup (monitor terminálu, pokud není přesměrován)
2	stderr	standardní chybový výstup (monitor terminálu, pokud není přesměrován)

Standardní chybový výstup slouží pro výpis chybových hlášení, která se zobrazí na obrazovce i přestože standardní výstup je přesměrován např. do souboru.

Příklady:

```
příkaz < vstup_soubor

příkaz > výst_soubor

příkaz >> výst_soubor

příkaz 2> výst_soubor

příkaz << /END/
text
text
...
/END/
```

Dochází k přesměrování standardního vstupu (deskriptor 0), standardního výstupu (deskriptor 1) nebo standardního chybového výstupu (deskriptor 2). Standardní vstup, výstup i chybový výstup je standardně nastaven na terminál.
Poslední možnost se nazývá *here document* a používá se pro zapsání vstupních dat přímo ve skriptu (viz. dále).

Přesměrování standardního i chybového výstupu do jednoho souboru.

```
příkaz > výst_soubor 2>&1
```

nebo

```
příkaz &> výst_soubor
```

Řetězení příkazů (kolona, pipeline):

```
příkaz1 | příkaz2 [ | příkaz3 ... ] [ > soubor ]
```

standardní výstup příkazu1 se pošle na standardní vstup příkazu2, jehož výstup se pošle na vstup příkazu3, atd. Je samozřejmě možné přesměrovat i vstup prvního příkazu a výstup posledního (na soubor).
Programy běží paralelně a data se předávají pomocí mechanismu systému, zvaném *pipe*. Protože většina příkazů UNIXu bez parametrů pracuje se standardním vstupem a výstupem, je možné je řadit do kolony a za pomoci relativně jednoduchých nástrojů provádět komplexní operace.

4 Pokročilejší konstrukce v shellu

4.1 Speciální znaky v shellu

Použitelné znaky se speciálním významem:

znak	význam
\	potlačuje spec. význam následujícího znaku
, ?, []	zástupné znaky ve jménech souborů, viz. výše
`>, >>, &>, <, <<, <&, |, |&`	přesměrování, viz. výše
;	oddělovač více příkazů na jedné řádce
&	příkaz ukončený ampersandem se spustí na pozadí
`‘cmd‘`	náhrada příkazu `cmd` jeho výstupem (spuštění!)
`’...’`	potlačuje interpretaci znaků uvnitř
`”...”`	interpretuje jen $ a ‘...‘
:	prázdný příkaz – akceptuje libovolný počet parametrů, ale nic nevykoná
#	zbytek řádku za # se ignoruje (komentář)
(příkazy)	provedení posloupnosti příkazů v nově spuštěné kopii shellu
{ příkazy ; }	provedení posloupnosti příkazů bez spuštění nové kopie shellu
`$[aritm. výraz]`	vyhodnocení aritmetického výrazu, lze použít +, -, *, /,

Místo `‘cmd‘` lze také použít `$(...)` a k vyhodnocení aritmetického výrazu lze také použít příkaz `expr` *výraz*, kde jednotlivé operátory a operandy výrazu musí být oddělené mezerami.

Proměnné v shellu

V shellu existují proměnné, které mohou mít pouze textový obsah. Nastavení proměnné se provede příkazem:

```
PROMĚNNÁ=text
```

Kolem znaku '=' nesmí být mezery, názvy proměnných jsou většinou z velkých písmen. Proměnné nejsou implicitně viditelné v podřízených shellech (např. ve spuštěném scriptu), pro předání proměnné do subshellu je potřeba použít příkaz `export PROMĚNNÁ`. Obsah proměnné lze dosadit použitím znaku „$", následovaným jménem proměnné v kterékoliv části příkazového řádku. V případě možné nejednoznačnosti názvu proměnné se název uzavře do složených závorek: `echo "${A}bc"`

Zrušení proměnné: `unset PROMĚNNÁ`

Spuštění programu s nastavenou/modifikovanou proměnnou: `PROMĚNNÁ=hodnota příkaz parametry`

V Shellu jsou také implicitně definované speciální proměnné, vztahující se k aktuální instanci shellu:

proměnná	význam
`$0`	jméno skriptu (tak jak byl spuštěn)
`$1--$9`	1. až 9. argument skriptu
`$#`	počet parametrů skriptu
`$*, #@`	zřetězené všechny argumenty shell
`$?`	návratová hodnota posledního provedeného příkazu
`$$`	PID aktuálního shellu (jednoznačné číslo)

$* a $# se liší pokud jsou v uvozovkách

```
"$*" --> "$1 $2 $3 ..."
"$@" --> "$1" "$2" "$3" ...
```

Lze použít i složitější konstrukce s použitím implicitních hodnot proměnné:

konstrukce	význam
`${proměnná-výraz}`	pokud je proměnná definována, je dosazena její hodnota, jinak je dosazena hodnota výrazu
`${proměnná=výraz}`	pokud je proměnná definována, je dosazena její hodnota, jinak je nastavena na hodnotu výrazu, která je i výsledkem celé konstrukce
`${proměnná+výraz}`	pokud je proměnná definována, je dosazena hodnota výrazu, pokud není definována je výsledkem prázdný řetězec
`${proměnná?výraz}`	pokud je proměnná definována, je dosazena její hodnota, jinak je hodnota výrazu vypsána na standardní chybový výstup a skript je ukončen
`${#proměnná}`	délka proměnné
`${proměnná#vzor}`	začátek obsahu proměnné je porovnán se vzorem (lze použít zástupné znaky, jako v názvech souborů) a případně je smazána nejkratší možná shodující se část
`${proměnná##vzor}`	začátek obsahu proměnné je porovnán se vzorem (lze použít zástupné znaky, jako v názvech souborů) a případně je smazána nejdelší možná shodující se část
`${proměnná%vzor}`	konec obsahu proměnné je porovnán se vzorem (lze použít zástupné znaky, jako v názvech souborů) a případně je smazána nejkratší možná shodující se část
`${proměnná%%vzor}`	konec obsahu proměnné je porovnán se vzorem (lze použít zástupné znaky, jako v názvech souborů) a případně je smazána nejdelší možná shodující se část

Další proměnné implicitně definované v shellu:

`HOME`	domácí adresář uživatele, lze také použít zkráceně znak
`HOSTNAME`	jméno počítače
`LANG`	u programů s národní podporou říká aby používali definice jiného jazyka než implicitní angličtiny (např. `LANG=cs_CZ mc`)– pouze pokud nejsou nastavené proměnné `LC_???`, které mají vyšší prioritu
`PATH`	seznam adresářů oddělených dvojtečkou pro vyhledávání příkazu, implicitně se nehledá v aktuálním a cesta většinou aktuální adresář neobsahuje – z akt. adresáře nutno spouštět pomocí `./příkaz`
`USER`	jméno přihlášeného uživatele
`SHELL`	jméno právě běžícího shellu
`TERM`	typ terminálu (pro celoobrazovkové aplikace, které musí znát speciální sekvence pro ovládání terminálu

4.2 Řídící konstrukce v shellu

Podmínky

```
if příkaz_podmínka
then příkaz
příkaz
...
fi
```

Provede se první příkaz a v případě nulové (úspěch) návratové hodnotě se provedou příkazy v bloku za `then`, vložením znaku `!` před příkaz dojde k negaci (příkaz se provede při nenulové návratové hodnotě).

Lze použít variantu rozšířenou o blok `else`, případně další bloky `elif`

```
if příkaz_podmínka1
then příkaz
příkaz
...
elif příkaz_podmínka2
then příkaz
příkaz
...
else příkaz
příkaz
...
fi
```

Větvení

```
case hodnota in
vzor1) příkazy
       ;;
vzor2) příkazy
       ;;
...
esac
```

Ve vzoru je možné použít zástupné znaky shellu (vzory pro hromadnou specifikaci souborů), případně oddělit více variant znakem `|`.

Cykly s podmínkou

```
while příkaz_podmínka
do příkaz
příkaz
...
done

until příkaz_podmínka
do příkaz
příkaz
...
done
```

Iterační cykly

```
for proměnná in seznam
do příkaz
příkaz
...
done

for ((výraz1 ; výraz2 ; výraz3 ))
do příkaz
příkaz
...
done
```

Všechny cykly lze předčasně ukončit příkazem `break`, případně si vyžádat ukončení běhu jednoho průchodu a okamžitý skok na začátek dalšího průchodu příkazem `continue`.

Příklady:

```
for i in * ; do
    if [ -d "$i" ] ; then
        stat $i
    fi
done

for ((i = 1; i < 10 ; i = i + 1))
do
    echo $i
done
```

Příkaz test

Použití hlavně u příkazu `if` a u cyklů `while` a `until`.
Má zkrácenou variantu `[` (je nutno ukončit párovou závorkou `]`.

Použití: `test` *argumenty* nebo `[` *argumenty* `]`.

Porovnávání řetězců:

argumenty	kdy je test úspěšný
řetězec	neprázdný řetězec
`-n` *řetězec*	neprázdný řetězec
`-z` *řetězec*	prázdný řetězec
řetězec1 `=` *řetězec2*	řetězce jsou stejné
řetězec1 `!=` *řetězec2*	řetězce nejsou stejné

Porovnávání čísel:

argumenty	kdy je test úspěšný
číslo1 `-eq` *číslo2*	čísla se rovnají (equal)
číslo1 `-ne` *číslo2*	čísla se nerovnají (not equal)
číslo1 `-lt` *číslo2*	první číslo je menší než druhé (less than)
číslo1 `-gt` *číslo2*	první číslo je větší než druhé (greater than)
číslo1 `-le` *číslo2*	první číslo je menší nebo rovno (less or equal)druhému
číslo1 `-ge` *číslo2*	první číslo je větší nebo rovno druhému (greater or equal)

Testování souborů:

argumenty	kdy je test úspěšný
soubor1 `-ef` *soubor2*	oba názvy ukazují na stejný i-node (hardlinky)
soubor1 `-nt` *soubor2*	první soubor je novější než druhý (newer than)
soubor1 `-ot` *soubor2*	první soubor je starší než druhý (older than)
`-e` *soubor*	existuje
`-f` *soubor*	obyčejný soubor
`-d` *soubor*	adresář
`-b` *soubor*	blokový speciální soubor
`-c` *soubor*	znakový speciální soubor
`-p` *soubor*	pojmenovaná roura
`-S` *soubor*	socket
`-r` *soubor*	přístupný pro čtení (právo r)
`-w` *soubor*	přístupný pro zápis (právo w)
`-x` *soubor*	spustitelný (právo x)
`-u` *soubor*	nastaven setuid bit
`-g` *soubor*	nastaven setgid bit
`-s` *soubor*	délka souboru je nenulová
`-L` *soubor*	soubor je symbolický odkaz

Podmínky u příkazu `test` lze řetězit pomocí binárních operátorů `-a` (and) a `-o` (or), negovat pomocí operátoru `!`, případně měnit pořadí zpracování pomocí závorek (nutno předřadit zpětné lomítko).

Vstup dat

Načtení řádky:

`read PROMĚNNÁ`

Příkaz načte z klávesnice (přesněji standardního vstupu) jeden řádek textu. Pro výpis výzvy k zadání dat se často používá příkaz typu:

```
echo -n "Zadej text: "
read VSTUP
```

Volba `-n` potlačí přechod kurzoru na nový řádek po vypsání textu.

Zobrazení možností:

```
select proměnná in možnosti
do
příkazy
...
done
```

Na začátku cyklu se vždy zobrazí očíslovaná nabídka z uvedených možností a čeká se na zadání čísla volby. Zadaná hodnota se uloží do proměnné REPLY a pokud to je správně zadané číslo volby, přiřadí se taktéž název volby do proměnné specifikované v příkazu `select`. Poté se postupně provedou všechny příkazy a opět dojde ke zobrazení nabídky. Uvnitř těla příkazu se očekává použití podmíněných příkazů nebo větvení za účelem provedení činnosti závislé na zadané volbě. Ukončit cyklus lze příkazem `break`.

5 Editory

První editor v unixu byl `ed`, jedná se o řádkový editor a přestože při jeho dobré znalosti umožňuje efektivně provádět i komplexnější úpravy v textových souborech, dnes se moc nepoužívá. Nicméně jeho znalost lze uplatnit při tvorbě příkazových sad pro dávkový editor `sed`, jehož příkazy vycházejí z příkazů tohoto editoru.

Dalším řádkovým editorem byl `ex`, který obsahuje rozsáhlejší sadu příkazů a podstatně větší možnosti nastavení, nicméně stále přetrvává značná neintuitivita ovládáni, zvláště pro začátečníky.

Prvním celoobrazovkovým editorem byl editor `vi`, což je v podstatě `ex` rozšířený o celoobrazovkový režim (možnost řádkového zadávání příkazů je zachována). Vzhledem k velkým různorodostem koncových zařízení používaných s UNIX systémy byl editor `vi` koncipován tak, aby pokud možno fungoval na co největším množství typů terminálů, z čehož plyne mírně odlišná filozofie ovládání od té, která je běžná v systémech typu Windows nebo DOS. Následující popis se bude týkat editoru `vi`.

Editor rozlišuje tři základní režimy, v kterých uživatel během práce může být a mezi kterými může libovolně přecházet:

- příkazový – v tomto režimu je editor po spuštění a kdykoliv do něj lze přejít stiskem `Esc`)
- vkládací (nebo přepisovací) – v tomto režimu lze do textu psát
- režim dolní příkazové řádky (aktivace stiskem dvojtečky, kurzor se přesune na spodní řádek obrazovky – možno zadávat příkazy editoru `ex`)

V příkazovém režimu (COMMAND MODE) se lze v textu pohybovat, vyhledávat, mazat libovolné části textu a provádět další operace. Ve vkládacím režimu (INSERT/REPLACE MODE) lze vkládat nebo přepisovat existující text jiným. V režimu dolní řádky se provádí např. ukládání a načítání souborů, dávkové nahrazování nebo změna nastavení editoru.

Příkazový režim

Zadávané příkazy se nezobrazují. Před většinu příkazů lze zadat číslo, které specifikuje počet opakování příkazu. Neúplně zadaný víceznakový příkaz lze zrusit klávesou `Escape`.

Přesun kurzoru, vyhledávání:

klávesy	význam
kurzorové klávesy	posun kurzoru v textu (nelze se dostat mimo text)
0	přesun na začátek řádku
^	přesun na první zobrazitelný znak řádku
$	přesun na konec řádku
b, B	skok na začátek předchozího slova[9]
w, W	skok na začátek následujícího slova
e, E	skok na konec aktuálního slova
{, }	skok na začátek, resp. konec odstavce[10]
G	přesun kurzoru na poslední řádek. Pokud předchází číslo, bere se jako číslo řádku.
/*text*	vyhledávání textu směrem dopředu
?*text*	vyhledávání textu směrem zpátky
n	vyhledání dalšího výskytu ve stejném směru
N	vyhledání dalšího výskytu v opačném směru
Ctrl-B, Ctrl-F	odrolování na předcházející (resp. následující) stránku
Ctrl-U, Ctrl-D	odrolování o půl stránky vzad (resp. vpřed)
m*písmeno*	uložení aktuální pozice kurzoru pod *písmeno*
‘*písmeno*	přesun kurzoru na pozici uloženou dříve příkazem `m`
’*písmeno*	jako předchozí příkaz, ale přesun kurzoru na začátek řádky
%	vyhledání protilehlé závorky `(, [, {, ), ], }`

[9] Pokud je příkaz zadán malým písmenem, může slovo obsahovat pouze písmena, číslice a podtržítko. V případě zadání velkým písmenem je součást slova všechno kromě mezer, tabulátorů a nových řádků

[10] za oddělovač řádků považuje editor prázdný řádek

Vkládání, mazání, změny a kopírování textu:
*Pozn.:*Všechny příkazy pro mazání a náhrady textu ukládají smazaný text do bufferu, odkud lze pak smazaný text vložit jinam. Při každém mazání se předchozí obsah bufferu přepisuje.

klávesy	význam
i	přepne do vkládacího režimu, kurzor zůstane na současné pozici
I	vkládací režim, kurzor na začátek řádku
A	vkládací režim, kurzor na konec řádku
o	vloží nový řádek za současný a přepne do vkládacího režimu
O	vloží nový řádek před současný a přepne do vkládacího režimu
x	smazání znaku pod kurzorem
dd	smazání řádky
dw	smazání slova
d$ nebo D	smazání textu do konce řádky[12]
cc, cw, c$, C	podobné jako mazání, ale editor se navíc automaticky přepne do vkládacího režimu (náhrada části textu jiným)
yy, Y	zkopírování aktuální rádky do bufferu
p	vložení obsahu bufferu za aktuální pozici kurzoru
P	vložení obsahu bufferu před aktuální pozici kurzoru
J	spojení aktuální řádky s následující
u	vrácení poslední operace měnící text (i vícenásobně)
U	zrušení všech změn na aktuální řádce (nelze po přesunutí kurzoru)

[12]Obecně když se zadá za d libovolný příkaz přesunu textu, bude smazán text od pozice kurzoru do cílového místa, kam by se kurzor přesunul. Tedy např. d} smaže všechno od kurzoru do první následující prázdné řádky.

Režim dolní příkazové řádky

příkaz	význam
:w [*soubor*]	uložení editovaného textu do aktuálního souboru (pokud je zadáno jméno, nastaví se tento soubor jako aktuální). Pro přepsání souboru je nutné zadat `:w!` *soubor*
:q	ukončení editoru (pokud jsou v textu neuložené změny, musí se použít příkaz `:q!`
:wq	uložení souboru a ukončení editoru
:x	ukončení editoru, pokud byl soubor změněn bude uložen (také ZZ)
:w » *soubor*	připojení textu na konec souboru
:*od*,*do*w *soubor*	uložení pouze daného rozsahu řádek
:e *soubor*	vymazání aktuálního textu a načtení obsahu jiného souboru (v případě neuložených změn nutno použít `:e!` *soubor*)
:r *soubor*	vložení obsahu souboru na aktuální pozici kurzoru
:n	přechod na další soubor v pořadí (zadaný na příkazové řádce)

6 Aplikace pod Linuxem

Firefox, Thunderbird Dvojice web browser a mail klient, používaná nejen pod Linuxem. Data jsou přenositelná mezi jednotlivými platformami, takže například pokud používáte Thunderbird pod windows, lze zkopírováním existujícího adresáře z daty plynule přejít k používaní Thunderbirdu pod Linuxem.

LICQ Jeden z mnoha ICQ klientů pro Linux. Je jednoduchý, vzhledově podobný klientu pro Windows a nabízí navíc některé užitečné funkce. Obsahuje jak textové rozhraní, tak grafické s využitím knihovny Qt.

XMMS Dá se říct, že se jedná o WinAMP pro Linux, je skinovatelný a z hlediska základních funkcí je s WinAMPem identický. Nabízí také různé druhy pluginů (zvukové efekty, vizualizace, různé pluginy pro ovládání pomocí tlačítek v dolní liště anebo také joystickem a IR ovladačem).

MPlayer univerzální přehrávač mnoha video i zvukových formátů. Umí také využít velké množství zobrazovacích grafických knihoven, ale i textové knihovny AALIB pro zobrazování grafiky v textovém režimu pomocí vhodných kombinací ascii znaků. Zvládá také přehrávání video DVD disků, ale nepodporuje menu obsažená na discích s filmy.

Ogle Tento přehrávač umí pouze přehrávání DVD video disků, ale zvládá tuto činnost lépe než MPlayer, tj. včetně podpory DVD menu.

gqview Prohlížeč obrázků, podobný IrfanView nebo ACDSee. Pomocí klávesové kombinace lze provádět rotace obrázků.

Gimp Editor obrázků, do jisté míry náhrada photoshopu, ale má odlišnou filozofii práce, která některým lidem činí potíže si zvyknout.

K3B Program na vypalování CD a DVD pro KDE, který zvládá vše co běžný vypalovací program pod Windows. Jedná se o grafickou nadstavbu nad textové utility cdrecord a balík dvd+rw-tools.

OpenOffice Kancelářský balík obsahující programy obdobné těm z Microsoft Office, jehož formáty umí číst i ukládat, ačkoliv má také své vlastní.

TEX Známý, kvalitní a hojně používaný systém pro sazbu textů, článků a knih. Narozdíl od MS Office nebo OpenOffice se nejedná o WYSIWYG systém. Uživatel nejprve připraví zdrojový text za pomoci formátovacích značek (vzdálená obdoba HTML, ale nabízí mnohem více možností, například definice maker, podmínek, atd.) a poté je dávkově proveden překlad do formátu DVI (Device Independent), který lze přímo tisknout anebo do PDF formátu, který je zase vhodnější pro distribuci elektronickou cestou.

GNUplot Nástroj pro generování grafů ze vstupních dat. Je to opět řádkový program, obsahující vlastní skriptovací jazyk, který umožňuje efektivní tvorbu jak jednoduchých grafů bez nutnosti dlouhého studování manuálů (stačí jen zkopírovat jeden z přiložených příkladů), tak i poměrně složitých grafů, zahrnujících pomocné výpočty a včetně mnoha možností popisů dat.

MGP - MagicPoint. Nástroj pro vytváření jednoduchých prezentací, vzhledově podobných např. těm z powerpointu, ale využívá textový formát vstupních dat (tedy opět to není WYSIWYG) a několik řídících příkazů, jimiž lze ovlivnit způsob zobrazení dat v jednotlivých stránkách. Po zvládnutí několika základních příkazů lze vytvářet i poměrně sofistikované prezentace efektivněji než ve WYSIWYG programech.

7 Literatura

Operační systém Linux - Příručka českého uživatele
Vilém Vychodil, Computer Press

Jemný úvod do systému UNIX
Lukáš Petrlík, Kopp