KAPITOLA 3

EDITOR VIM A INTERPRETER BASH SHELL

Použitie jednoduchých príkazov tvorí základnú zručnosť pri práci s OS Linux. Okrem jednoduchého použitia priamo na príkazovom riadku však môžeme vytvárať programy obsahujúce viacero príkazov – takzvané **skripty**. Písanie skriptov vyžaduje znalosť pokročilých techník a príkazov príkazového interpretra. V nasledujúcej kapitole sa zameriame na vytváranie skriptov s použitím interpretra **bash shell**. Pre písanie skriptov použijeme editor **vim**, ktorého základy popíšeme v úvode kapitoly.

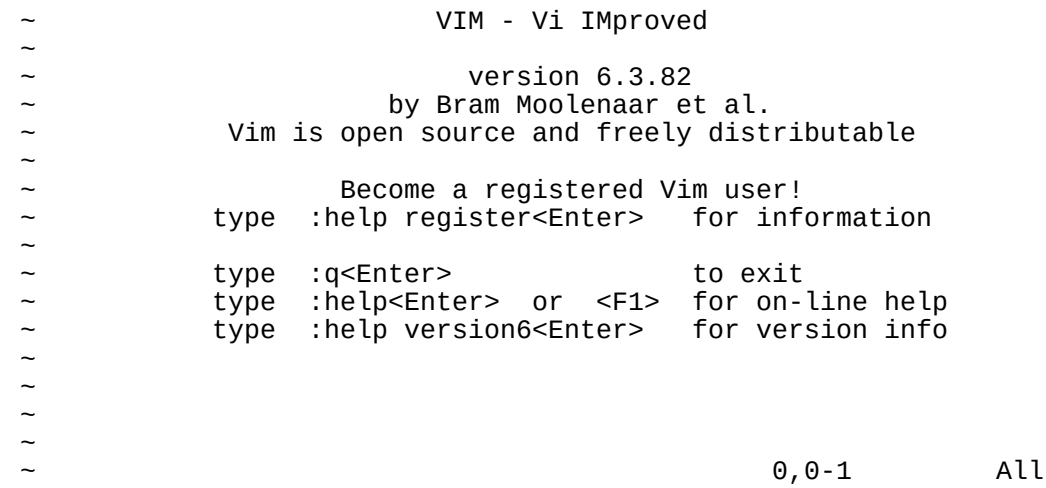
3.1 Vim

Editor **vim** (Vi IMproved) principiálne vychádza z editora **vi** (Visual Interactive). Ide o obrazovkový editor, ktorý je implementovaný vo väčšine systémov Unix/Linux. Editor má dva základné režimy – **obrazovkový** a **riadkový**, pričom okrem základnej funkčnosti obsahuje veľa vlastností užitočných pri vytváraní programov v rôznych programovacích jazykoch.

Obrazovkový mód môžeme rozdeliť na tri módy – vkladací, nahradzovací a vizuálny. **Vkladací mód** slúži na vloženie textu do aktuálne otvoreného súboru, je možné posúvať sa pomocou kurzoru v dokumente. **Nahradzovací mód** slúži na nahradenie existujúceho textu. **Vizuálny mód** slúži na označenie textu pre operácie kopírovania, kopírovania a mazania textu. Je potrebné poznamenať, že editor vi nemá vizuálny mód.

Riadkový - **príkazový mód** slúži na zadávanie príkazov pre manipuláciu s textom prípadne posun kurzora a ďalšie operácie. Napriek tomu že takýto koncept sa javí pri porovnaní s inými editormi ako mätúci, je napriek svojej prvotnej náročnosti veľmi silnou vlastnosťou editora vim. V príkazovom móde nie je problém zadať príkaz ako „zmaž riadky 3-5“, prípadne „vykonaj poslednú akciu nad riadkom“. Podobný princíp využíva aj ďalší silný editor **emacs**, ktorým sa však v ďalšom texte nebudeme zaoberať.

Základná obrazovka editora `vim` sa vyvolá po zadaní príkazu `vim` na príkazovom riadku:



Ukončene programu je možné za daním kombinácie `:q` resp. `:q!` v príkazovom móde. Do príkazového módu sa dostaneme stlačením `Esc`. Príkaz `:q` znamená ukončenie (angl. quit), príkaz `:q!` ukončenie bez uloženia.

Prepínanie medzi jednotlivými módmi sa deje zadávaním príkazov:

- **i** prepnutie do vkladacieho módu – text sa začne vkladať pred aktuálnu pozíciu kurzora (z príkazového módu)
- **a** prepnutie do vkladacieho módu – text sa začne vkladať za aktuálnu pozíciu kurzora (z príkazového módu)
- **I** premiestnenie kurzora na začiatok riadku a prepnutie do vkladacieho módu (z príkazového módu)
- **A** premiestnenie kurzora na koniec riadku a prepnutie do vkladacieho módu (z príkazového módu)
- **v** prepnutie do vizuálneho módu (z vizuálneho alebo vkladacieho módu)
- **Esc** prepnutie do príkazového módu (z vizuálneho alebo vkladacieho módu) – príkazy na riadok sa zadávajú po zadaní znaku `:`
- **R** prepnutie do nahradzovacieho módu (z príkazového módu)

Posun v dokumente je možný v príkazovom móde použitím nasledujúcich kláves:

- **0** presun na začiatok riadku
- **$** presun na koniec riadka
- **h** presun o jeden znak doľava (rovnako je možné použiť šípku doľava)

l posun o jeden znak doprava (rovnako je možné použiť šípku doprava)

j posun o jeden znak dole (rovnako je možné použiť šípku dole)

k posun o jeden znak hore (rovnako je možné použiť šípku hore)

b posun o jedno slovo dozadu

w posun o jedno slovo dopredu

{ posun o jeden odsek dozadu

} posun o jeden odsek dopredu

Posun na riadok číslo 12 je možný napríklad zadaním čísla 12 a písmen G alebo príkazom `:12` a stlačením klávesy `Enter`.

Uloženie dokumentu je možné v príkazovom móde pomocou príkazu `:w`. Dokumentáciu k príkazom a ďalšie informácie je možné získať zadaním príkazu `:help`. Z helpu sa potom odchádza rovnako, ako z iného dokumentu – zadaním príkazu `:q`.

Jednou z potrebných vlastností je **vyhľadávanie** v obsahu. Pre vyhľadanie reťazca je potrebné v príkazovom móde zadať znak `/` nasledovaný reťazcom, ktorý chceme vyhľadať. Príkaz `/int` teda nájde všetky výskyty slova `int` v obsahu a zvýrazní ich. Kurzor sa nastaví pred prvý výskyt znaku, posun na ďalší výskyt je možný pomocou samotného príkazu `/`. Pri špecifikácii vyhľadávacieho reťazca je potom možné použiť aj hviezdičkovú konvenciu prípadne vyhľadávacie vzory (`:help pattern`).

V príkazovom móde môžeme tiež zadávať priamo **príkazy** operačného systému. Nie je preto nutné „vyskakovať z editora“, ak potrebujeme napríklad spustiť kompiláciu. Stačí v príkazovom móde zadať `:!` a samotný príkaz na kompiláciu.

Výborným pomocníkom pre osvojenie si základných zručností s editorom `vim` je tutoriál, ktorý môžete spustiť priamo z príkazového riadku Linux-u príkazom:

```
zabovsky@frios:~$ vimtutor
```

Editor vim obsahuje veľa pokročilých vlastností. Okrem automatického zvýrazňovania syntaxe na základe rozpoznaného typu súboru, dopĺňania kľúčových slov, názvov funkcií a premenných, formátovania dokumentu a iným vlastností umožňuje aj nastavenie vlastného používateľského profilu. Napriek všeobecnému názoru, že ide o komplikovaný editor, môžeme z vlastných skúseností povedať, že základné osvojenie si editora pre potreby písania programov trvá rovnako dlho, ako je to u iných komplexne vybavených editoroch zabudovaných v moderných integrovaných vývojových prostrediach.

3.1.1 Najčastejšie používané funkcie editora `vim`

Prehľad najčastejšie používaných funkcií editora `vim` z pohľadu programátora uvedieme v stručnom prehľade:

- kurzor sa presúva použitím kurzorových kláves alebo pomocou kláves hjkl (h - doľava, j - dole, k - hore, l - doprava)
- editor sa spúšťa z príkazového riadku príkazom `vim <subor>`
- práca s editorom sa ukončuje pomocou príkazu `:q!` (zruší všetky zmeny) alebo pomocou `:wq` (ulož všetky zmeny a skonči).
- vymazanie znaku pod kurzorom v obrazovkovom móde: stlačiť klávesu `x`
- vloženie textu v obrazovkovom móde na pozíciu kurzora: `i vkladaný text <Esc>`
- zmazanie slova nad ktorým je umiestnený kurzor - `dw`
- zmazanie riadku od aktuálnej pozície kurzora po koniec riadku - `d$`
- zmazanie celého riadku - `dd`
- formát príkazov v obrazovkovom móde je `[cislo] prikaz`, kde číslo určuje počet opakovaní nasledujúceho príkazu a príkaz špecifikuje akciu, ktorú používateľ požaduje vykonať (`2dd` zmaže dva riadky)
- zrušenie predchádzajúcej akcie (undo) - `u`
- zrušenie zrušenia zmien (redo) - `Ctrl-R`
- vloženie riadku do clipboardu sa vykoná po stlačením kombinácie `Shift-V` (označenie riadku) a stlačením `Y` (vloženie do clipboardu), vloženie za kurzor potom stlačením `p`
- nahradenie znaku pod kurzorom - `r` a znak, ktorým sa má pôvodný znak nahradiť
- `Ctrl-g` zobrazí aktuálnu pozíciu v súbore
- `Shift-G` presunie kurzor na posledný riadok súboru
- vyhľadávanie textu smerom od začiatku súboru `/hladany_text`, smerom od konca súboru `?hladany_text`. Po vyhľadaní sa dá presunúť na ďalší vyhľadaný záznam pomocou stlačenia `n` alebo `Shift-N` na predchádzajúci záznam.
- nahradenie prvého výskytu textu `stary` na riadku za text `novy` - `:s/stary/novy`
- nahradenie všetkých výskytov textu `stary` na riadku za text `novy` - `:s/stary/novy/g`

- nahradenie všetkých výskytov textu `stary` medzi riadkami 12 a 38 za text `novy` - `:12,28s/stary/novy/g`
- nahradenie všetkých výskytov textu `stary` v súbore za text `novy` - `:%s/stary/novy/g`
- nahradenie všetkých výskytov textu `stary` v súbore za text `novy` s potvrdením - `:%s/stary/novy/gc`
- `:w` uloží zmeny v aktuálnom súbore
- `:w subor` uloží editovaný text ako súbor s názvom `subor`
- `:r subor` načíta súbor z disku a vloží ho na aktuálnu pozíciu kurzora
- `:23` presunie kurzor na riadok 23.

3.2 Awk

Programovací jazyk `awk`, ako sa bežne nazýva, je veľmi silný programovací jazyk. Programy napísané v tomto jazyku akceptujú vstup z príkazového riadku aj z dátových súborov. `Awk` spracováva vstup záznam po zázname, kde vo všeobecnosti môžeme za záznam považovať práve jeden riadok. Oddeľovačom je teda znak nový riadok. Základný program v `awk`-u vyzerá nasledovne:

```
zabovsky@frios:~$ ls -l | awk '{print $0}'
total 8
-rw-r--r-- 1 zabovsky zabovsky 55 Nov 10 19:49 s1
-rw-r--r-- 1 zabovsky zabovsky 45 Nov 10 19:49 s1~
zabovsky@frios:~$
```

Predchádzajúci príkaz čítal vstup a ten zobrazoval na obrazovke terminálu. Príklad demonštruje niekoľko charakteristík písania programov v awku:

- program v awku môže byt spustený z príkazového riadku príkazom `awk 'program'`
- awk akceptuje vstup zo štandardného vstupu a zobrazuje výstup na štandardný výstup
- výrazy sú v awku uzavreté do zložených zátvoriek
- `$0` označuje celú množinu záznamov

Každý záznam je možné rozdeliť do polí, na ktoré sú definované referencie $1, $2 atď.:

```
zabovsky@frios:~$ ls -l
total 8
```

```
-rw-r--r-- 1 zabovsky zabovsky 55 Nov 10 19:49 s1
-rw-r--r-- 1 zabovsky zabovsky 45 Nov 10 19:49 s1~
zabovsky@frios:~$ ls -l | awk '{print $1, $9}'
total
-rw-r--r-- s1
-rw-r--r-- s1~
```

Skript v awku môže vyzerať napríklad takto:

```
zabovsky@frios:~$ cat awk_script
BEGIN { print ("Start"); }
NR==2 { print ("Druhy zaznam je " $0); }
END { print ("Koniec"); }
zabovsky@frios:~$ ls -l | awk -f awk_script
Start
Druhy zaznam je -rwxr-xr-x 1 misik misik 107 Nov 10 20:50 awk*
Koniec
```

Ako vidieť z príkladu, awk obsahuje predefinované premenné. Niektoré z nich majú nasledujúci význam:

NR	číslo záznamu, ktorý bude spracovaný
NF	číslo pola (stĺpca) v danom zázname
FS	oddeľovač polí
RS	oddeľovač záznamov
FILENAME	meno aktuálneho vstupného súboru

Ako vidíme, awk je veľmi užitočný jazyk najmä pri spracovávaní textových dát. Jeho syntax nie je síce najprehľadnejšia, funkčnosť však vyvažuje tento nedostatok. Je preto v špecifických prípadoch vhodné zvážiť práve použitie tohto nástroje pre spracovanie dát v textovom tvare.

3.3 Bash shell

Shell je interpreter programovacieho jazyka. Číta príkazy z terminálu alebo zo súboru a vykonáva ich. Medzi historicky najvýznamnejšie shelly patrí **Bourne shell** (sh). Ďalší rozšírený shell je **C shell**, ktorého syntax vychádza z jazyka C.

Bash shell je kompatibilný s Bourne shellom a rozširuje ho o ďalšie užitočné vlastnosti zavedené shellmi **korn shell** (ksh) a **C shell** (csh). Je implementovaný tak, aby vyhovoval špecifikácii IEEE POSIX Shell and Utilities specification 1003.2.

Postupnosť príkazov pre shell čítaná zo súboru je tvorí vo svojej podstate program. V ďalšom texte budeme takýto program pre shell nazývať **skript**. Pri prihlásení sa vykonáva tzv. login shell, ktorý po úspešnom overení hesla hľadá v domovskom adresári používateľa skripty `.bash_profile`, `.bash_login` alebo `.profile` a vykoná ich. Pri odhlasovaní sa pokúsi vykonať skript `.bash_logout`. Rozlišujeme dva spôsoby práce

so shellom: **interaktívny**, kedy príkazy sa čítajú z terminálu a výstup sa zapisuje späť na terminál[1] a **neinteraktívny**, kedy sa príkazy sa čítajú zo súboru.

3.3.1 Jednoduché príkazy

Jednoduchý príkaz je postupnosť voliteľných priradení hodnôt premenným nasledovaná slovami vzájomne oddelenými bielymi miestami. Prvé slovo určuje príkaz, ostatné sa prevezmú ako argumenty. Príkladom jednoduchého príkazu je napríklad príkaz na zmenu adresára:

```
zabovsky@frios:~$ cd adresar
```

3.3.2 Návratové hodnoty

Každý proces po svojom skončení vracia tomu, kto ho spustil návratovú hodnotu. Vo všeobecnosti sa nulová hodnota chápe ako úspešné ukončenie. Nenulová hodnota spravidla znamená chybový stav, kde vrátená hodnota definuje číslo chyby. Návratovú hodnotu posledného spusteného príkazu je možné zistiť pomocou premennej `$?`:

```
zabovsky@frios:~$ ls pokus
adresar  novy  novy_adresar  stary
zabovsky@frios:~$ echo $?
0
zabovsky@frios:~$ ls neexistuje
ls: neexistuje: No such file or directory
zabovsky@frios:~$ echo $?
2
```

Jednoduchý príkaz zväčša vracia návratový kód v intervale 0 až 127. Hodnota 128 a vyššia znamená ukončenie procesu signálom, ktorého hodnota je daná rozdielom návratovej hodnoty a hodnoty 128.

3.3.3 Zoznamy

Zoznam je postupnosť jedného alebo viacerých spojení (spojenie napr. `ls | more`) oddelených jedným z riadiacich operátorov `|| & && ;` a je ukončená znakom `& ; <novy_riadok>`. Ak je príkaz ukončený znakom &, vykoná sa v kópii shellu na pozadí - nečaká sa na dokončenie príkazu:

```
zabovsky@frios:~$ find / -name ".profile" > vysledok 2> chyby &
[1] 2632
zabovsky@frios:~$ jobs
[1]+  Running        find / -name ".profile" >vysledok 2>chyby &
```

[1] Pri spúšťaní interaktívneho shellu sa v domovskom adresári hľadá súbor .bash_rc.

Ak sú príkazy oddelené znakom `;` budú vykonané sekvenčne (za sebou). Shell počká na dokončenie prvého príkazu a až potom spustí nasledujúci príkaz. && je logický súčin (AND) a `||` logický súčet (OR). Pre lepšie pochopenie uvedieme na porovnanie niekoľko príkladov:

- `cd adresar; ls` - je ekvivalentný príkaz ako separátny zápis príkazu `cd adresar` a `ls`
- `cd adresar & ls` - vytvorí sa kópia shellu na vykonanie prvého príkazu a v nej sa príkaz spustí. Druhý príkaz sa vypíše v pôvodnom shelly. Na ukončenie prvého príkazu sa nečaká.
- `cd adresar && ls` - druhý príkaz sa vykoná iba ak prvý skončí s nulovým návratovým kódom. Ak teda zadaný adresár neexistuje, príkaz `ls` sa nevykoná.
- `cd adresar || mkdir adresar` - ak neexistuje adresár, vytvorí sa. Ak existuje, nastaví sa ako pracovný adresár.

3.3.4 Zátvorkovanie

Zoznamy je možné uzatvárať do zátvoriek dvoma spôsobmi. Zoznam uzavretý do guľatých zátvoriek sa vykoná v kópii shellu. Premenné nastavené vo vnútri zátvoriek, či iné modifikácie prostredia shellu sa po ukončení posledného príkazu uzavretého v zátvorkách strácajú (ukončí sa kópia shellu). Zoznam uzavretý do zložených zátvoriek sa vykoná v aktuálnom shelli.

```
zabovsky@frios:~/pokus$ ls -la
total 16
drwxr-xr-x 4 zabovsky zabovsky 4096 Sep  2 20:42 .
drwxr-x--- 6 zabovsky zabovsky 4096 Sep  3 18:00 ..
drwxr-xr-x 2 zabovsky zabovsky 4096 Sep  2 17:15 adresar
-rw-r--r-- 1 zabovsky zabovsky    0 Sep  2 20:11 novy
drwxr-xr-x 2 zabovsky zabovsky 4096 Sep  2 20:45 novy_adresar
-rw-r--r-- 1 zabovsky zabovsky    0 Sep  2 20:11 stary
zabovsky@frios:~/pokus$ (pwd; cd adresar; pwd); pwd
/home/zabovsky/pokus
/home/zabovsky/pokus/adresar
/home/zabovsky/pokus
zabovsky@frios:~/pokus$ { pwd; cd adresar; pwd;}; pwd
/home/zabovsky/pokus
/home/zabovsky/pokus/adresar
/home/zabovsky/pokus/adresar
```

3.3.5 Potlačenie významu metaznakov

Použitie znakov, ktoré sú operačným systémom chápané ako riadiace môže v určitých prípadoch spôsobovať používateľovi značné problémy. Majme napríklad nasledujúci príklad:

```
zabovsky@frios:~/pokus$ echo Otec&Syn
[1] 2814
Otec
-bash: Syn: command not found
[1]+  Done                    echo Otec
```

Vidíme, že znak & spôsobuje neželané chovanie sa príkazu. V ďalšom texte tejto podkapitoly si ukážeme, ako sa základným problémom s riadiacimi znakmi vyhnúť.

Znak bezprostredne za **obráteným lomítkom** \ stráca svoj riadiaci význam. Výnimkou je znak \ bezprostredne nasledovaný novým riadkom - ide o tzv. pokračovací riadok. Ak potrebujeme zapísať obrátené lomítko, musíme lomítka zdvojiť.

```
zabovsky@frios:~/pokus/adresar$ ls -la
total 8
drwxr-xr-x 2 zabovsky zabovsky 4096 Sep  2 17:15 .
drwxr-xr-x 3 zabovsky zabovsky 4096 Sep  3 18:15 ..
zabovsky@frios:~/pokus/adresar$ cd ..
zabovsky@frios:~/pokus$ c
zabovsky@frios:~/pokus$ ls -la
total 12
drwxr-xr-x 3 zabovsky zabovsky 4096 Sep  3 18:15 .
drwxr-x--- 6 zabovsky zabovsky 4096 Sep  3 18:14 ..
drwxr-xr-x 2 zabovsky zabovsky 4096 Sep  2 17:15 adresar
zabovsky@frios:~/pokus$ touch a b
zabovsky@frios:~/pokus$ ls -la
total 12
drwxr-xr-x 3 zabovsky zabovsky 4096 Sep  3 18:15 .
drwxr-x--- 6 zabovsky zabovsky 4096 Sep  3 18:14 ..
-rw-r--r-- 1 zabovsky zabovsky    0 Sep  3 18:15 a
drwxr-xr-x 2 zabovsky zabovsky 4096 Sep  2 17:15 adresar
-rw-r--r-- 1 zabovsky zabovsky    0 Sep  3 18:15 b
zabovsky@frios:~/pokus$ touch a\ b
zabovsky@frios:~/pokus$ ls -la
total 12
drwxr-xr-x 3 zabovsky zabovsky 4096 Sep  3 18:59 .
drwxr-x--- 6 zabovsky zabovsky 4096 Sep  3 18:14 ..
-rw-r--r-- 1 zabovsky zabovsky    0 Sep  3 18:15 a
-rw-r--r-- 1 zabovsky zabovsky    0 Sep  3 18:59 a b
drwxr-xr-x 2 zabovsky zabovsky 4096 Sep  2 17:15 adresar
-rw-r--r-- 1 zabovsky zabovsky    0 Sep  3 18:15 b
zabovsky@frios:~/pokus$ rm a b
zabovsky@frios:~/pokus$ ls -la
total 12
drwxr-xr-x 3 zabovsky zabovsky 4096 Sep  3 18:59 .
drwxr-x--- 6 zabovsky zabovsky 4096 Sep  3 18:14 ..
-rw-r--r-- 1 zabovsky zabovsky    0 Sep  3 18:59 a b
drwxr-xr-x 2 zabovsky zabovsky 4096 Sep  2 17:15 adresar
zabovsky@frios:~/pokus$ rm a\ b
zabovsky@frios:~/pokus$ ls -la
total 12
drwxr-xr-x 3 zabovsky zabovsky 4096 Sep  3 18:59 .
```

```
drwxr-x--- 6 zabovsky zabovsky 4096 Sep  3 18:14 ..
drwxr-xr-x 2 zabovsky zabovsky 4096 Sep  2 17:15 adresar
```

Uzavretím do **apostrofov** strácajú riadiace znaky svoj význam:

```
zabovsky@frios:~/pokus$ echo 'Otec&Syn'
Otec&Syn
zabovsky@frios:~/pokus$ echo 'Otec\Syn'
Otec\Syn
```

Uzavretím do uvodzoviek strácajú riadiace znaky svoj význam. Výnimkou sú znaky `$`, `'` a `\`. Znak `$` slúži na prístup k obsahu premennej:

```
zabovsky@frios:~/pokus$ Syn=" a ja som syn"
zabovsky@frios:~/pokus$ echo 'Otec&Syn'
Otec&Syn
zabovsky@frios:~/pokus$ echo 'Otec$Syn'
Otec$Syn
zabovsky@frios:~/pokus$ echo "Otec$Syn"
Otec a ja som syn
```

3.3.6 Expanzie

Shell po načítaní celého príkazového riadku vykoná tzv. expanzie. Napríklad znak * sa nahradí zoznamom mien z bežného adresára oddelených medzerou a až potom sa spustím príkaz využívajúci expanziu.

```
zabovsky@frios:~/pokus$ ls -la
total 12
drwxr-xr-x 3 zabovsky zabovsky 4096 Sep  3 20:59 .
drwxr-x--- 6 zabovsky zabovsky 4096 Sep  3 18:14 ..
drwxr-xr-x 2 zabovsky zabovsky 4096 Sep  3 20:59 adresar
-rw-r--r-- 1 zabovsky zabovsky    0 Sep  3 20:59 subor1
zabovsky@frios:~/pokus$ ls -la adresar
total 8
drwxr-xr-x 2 zabovsky zabovsky 4096 Sep  3 20:59 .
drwxr-xr-x 3 zabovsky zabovsky 4096 Sep  3 20:59 ..
-rw-r--r-- 1 zabovsky zabovsky    0 Sep  3 20:59 subor2
zabovsky@frios:~/pokus$ ls -la *
-rw-r--r-- 1 zabovsky zabovsky    0 Sep  3 20:59 subor1

adresar:
total 8
drwxr-xr-x 2 zabovsky zabovsky 4096 Sep  3 20:59 .
drwxr-xr-x 3 zabovsky zabovsky 4096 Sep  3 20:59 ..
-rw-r--r-- 1 zabovsky zabovsky    0 Sep  3 20:59 subor2
```

Bash shell spracováva niekoľko typov expanzií:

- pomocou zložených zátvoriek

- vlnovkovú expanziu
- expanziu parametrov
- nahradenie výstupom príkazu
- aritmetickú expanziu
- nahradenie procesom
- rozdelenie slov
- expanziu mien
- odstránenie znakov riadiaceho významu

Expanzia pomocou zložených zátvoriek je určená na vytváranie podobných reťazcov, ktoré sa odvodzujú podľa presne zadaných pravidiel.

```
zabovsky@frios:~/pokus$ ls -la
total 8
drwxr-xr-x 2 zabovsky zabovsky 4096 Sep  3 21:07 .
drwxr-x--- 6 zabovsky zabovsky 4096 Sep  3 18:14 ..
zabovsky@frios:~/pokus$ mkdir
/home/zabovsky/pokus/{old,new,dist,bugs}
zabovsky@frios:~/pokus$ ls -la
total 24
drwxr-xr-x 6 zabovsky zabovsky 4096 Sep  3 21:08 .
drwxr-x--- 6 zabovsky zabovsky 4096 Sep  3 18:14 ..
drwxr-xr-x 2 zabovsky zabovsky 4096 Sep  3 21:08 bugs
drwxr-xr-x 2 zabovsky zabovsky 4096 Sep  3 21:08 dist
drwxr-xr-x 2 zabovsky zabovsky 4096 Sep  3 21:08 new
drwxr-xr-x 2 zabovsky zabovsky 4096 Sep  3 21:08 old
```

Vlnovková expanzia sa aplikuje sa ak argument začína znakom ~. V takomto prípade sa všetky znaky za vlnovkou až po lomítko interpretujú ako používateľské meno. Ak používateľské meno existuje, expanduje sa kombinácia s menom na absolútnu cestu k domovskému adresáru. Ak neexistuje, potom sa expanzia nevykoná.

```
zabovsky@frios:/$ pwd
/
zabovsky@frios:/$ cd ~zabovsky
zabovsky@frios:~$ pwd
/home/zabovsky
zabovsky@frios:~$ cd /
zabovsky@frios:/$ pwd
/
zabovsky@frios:/$ cd ~
zabovsky@frios:~$ pwd
/home/zabovsky
```

Pri **expanzii parametrov** sa shell pokúsi parameter expandovať vtedy, ak je zapísaný v tvare `${param}` alebo iba `$param`. Pri modifikácii expandovaných parametrov sú k dispozícii nasledujúce operátory:

- `${param:-hodnota}` - použije sa implicitná hodnota `hodnota` v prípade, že neexistuje parameter `param`
- `${param:=hodnota}` - parametru `param` sa priradí implicitná hodnota `hodnota`

```
zabovsky@frios:~$ echo Cesty: ${PATH:=/bin:/usr/bin}
Cesty: /usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games
zabovsky@frios:~$
```

- `${param:?[hodnota]}` - oznámi sa chyba, ak je parameter prázdny alebo nedefinovaný

```
zabovsky@frios:~$ echo ${ODPOVED:?'Nebola zadana odpoved!'}
-bash: ODPOVED: Nebola zadana odpoved!
zabovsky@frios:~$ ODPOVED=ano
zabovsky@frios:~$ echo ${ODPOVED:?'Nebola zadana odpoved!'}
ano
zabovsky@frios:~$
```

- `${#param}` - oznámi dĺžku reťazca `param`

Nasledujúce expanzie sú určené pre prácu s reťazcami:

- `${param%hodnota}` - expanzia odstráni najkratší sufix `hodnota` z parametra `param`

```
zabovsky@frios:~$ FILE=subor.c
zabovsky@frios:~$ echo ${FILE%.c}
subor
```

- `${param%%hodnota}` - expanzia odstráni najdlhší sufix `hodnota` z parametra `param`
- `${param#hodnota}` - expanzia odstráni najkratší prefix `hodnota` z parametra `param`

```
zabovsky@frios:~$ FILE=subor.c
zabovsky@frios:~$ echo ${FILE#subor}
.c
```

- `${param##hodnota}` - expanzia odstráni najdlhší prefix hodnota z parametra `param`

Nahradenie výstupu príkazom dovoľuje vložiť do príkazového riadku obsah štandardného výstupu iného príkazu:

```
zabovsky@frios:~/pokus$ echo $(ls | sort -r)
old new dist bugs
zabovsky@frios:~/pokus$ echo `ls | sort -r`
old new dist bugs
zabovsky@frios:~/pokus$ ls | sort -r
old
new
dist
bugs
```

Aritmetická expanzia je určená na vyhodnocovanie aritmetických výrazov:

```
zabovsky@frios:~/pokus$ a=1
zabovsky@frios:~/pokus$ b=2
zabovsky@frios:~/pokus$ echo $((a+b))
3
zabovsky@frios:~/pokus$ echo $[a+b]
3
```

Pri aritmetickej expanzii je možné použiť nasledujúce operátory:

+ -	unárne plus a mínus
! ~	logická negácia a inverzia bitov
*** / %**	násobenie, delenie a zvyšok po delení
+ -	sčítanie a odčítanie
<< >>	bitový posun doľava a doprava
<= => < >	porovnanie
== **!=**	rovnosť a nerovnosť
&	bitový súčin (AND)
^	exkluzívny bitový súčet (XOR)
|	bitový súčin (OR)
&&	logický súčin
||	logický súčet
= *= /= %= += -= <<= >>= &= ^= !=	operácie priradenia výsledku

Expanzia **nahradenia procesom** využíva pomenované rúry (FIFO), do ktorých sa buď zapisuje, alebo sa z nich číta.

3.3.7 Komentáre

Komentáre začínajú znakom #. Od tohto znaku sa všetky ďalšie príkazy na riadku ignorujú. Použitie komentárov v skriptoch je veľmi vhodné a časté, keďže účel použitia väčšiny skriptov je značne špecifický. Komentáre môžete nájsť napríklad v rôznych konfiguračných súborov.

3.3.8 Skript

Skript je súbor s príkazmi pre shell. Súbor so skriptom je možné **spustiť** viacerými spôsobmi. Ak je skript možné iba čítať, je môžeme ho spustiť buď príkazom `bash nazov_skriptu` alebo `. nazov_skriptu`. Ak je skript možné vykonať, môžeme ho spustiť buď príkazom `nazov_skriptu` alebo `./nazov_skriptu`.

Príklad jednoduchého skriptu môže vyzerať napríklad nasledovne: príkaz na prvom riadku určuje shell, ktorý bude vykonávať skript (pre aký shell je skript napísaný). V prípade, že tento riadok chýba bude použitý implicitný shell, čo môže viesť k chybám z dôvodu nedostatočnej kompatibility jednotlivých interpretrov príkazov.

```
zabovsky@frios:~/pokus$ ls -la
total 12
drwxr-xr-x 2 zabovsky zabovsky 4096 Sep  3 22:03 .
drwxr-x--- 6 zabovsky zabovsky 4096 Sep  3 18:14 ..
-rw-r--r-- 1 zabovsky zabovsky   40 Sep  3 22:02 skript
zabovsky@frios:~/pokus$ cat skript
#!/bin/bash

echo "Toto je skript."
zabovsky@frios:~/pokus$ . skript
Toto je skript.
```

3.3.9 Premenné

Premenné v skripte vytvoríme tak, že priradíme jej názvu hodnotu (napr. `MENO="Meno"`) a použijeme tak, že zapíšeme výraz `$MENO` alebo `${MENO}`.

```
zabovsky@frios:~$ cat skript
#!/bin/bash

ADRESAR=/home/zabovsky/export
ls $ADRESAR
mkdir $ADRESAR/projekt
```

3.3.10 Pozičné parametre

Pozičným parametrom sa implicitne priradia obsahy argumentov príkazového riadku. Označujeme ich číslom a zapisujeme v tvare `$n` alebo `${n}`.

```
zabovsky@frios:~$ cat skript
#!/bin/bash

echo Parameter jedna = $1
echo Parameter dva = $2
echo Parameter tri = $3
echo Pocet parametrov $#
zabovsky@frios:~$ . skript jedna dva tri styri pat
Parameter jedna = jedna
Parameter dva = dva
Parameter tri = tri
Pocet parametrov 5
```

3.3.11 Zvláštne parametre

Niektoré parametre spracováva shell zvláštnym spôsobom. Sú to napríklad tieto parametre:

$* parameter expanduje do všetkých zadaných pozičných parametrov

$# parameter expanduje do počtu aktuálne nastavených pozičných parametrov

$? parameter expanduje do návratového kódu posledného ukončeného príkazu bežiaceho v popredí

$$ parameter expanduje do čísla procesu aktuálne bežiaceho shellu

$0 parameter expanduje do mena aktuálneho shellu

3.4 Zložené príkazy

Zložené príkazy slúžia na vytváranie komplexnejších programových štruktúr v skriptoch. Keďže sú väčšinou založené na konštrukciách, ktoré bežne poznáme s programovacích jazykov, uvedieme v tejto kapitole iba ich stručný prehľad.

3.4.1 for

Príkazom `for` typicky vykonávame rovnakú činnosť pre danú skupinu argumentov. Majme úlohu, kedy chceme zistiť počet položiek (súborov, podadresárov a pod.) aktuálneho adresára. Výsledný skript je uvedený vo výpise príkladu 3.1

Príklad 3.1: Skript s použitím príkazu `for`

```
#!/bin/bash

POM=0
for POLOZKA in *
do
    POM=$[$POM+1]
done
```

```
echo "Pocet poloziek: $POM"
```

Okrem príkazu `for` poznáme a príkaz `select`, ktorý je interaktívnym variantom príkazu `for`.

3.4.2 `if` a `case`

Príkaz `if` realizuje vetvenie programu. Príklad použitia príkazu `if`, ktorý v prípade existencie súboru vypíše tento súbor a v prípade, že súbor neexistuje ho vytvorí, je uvedený vo výpise 3.2.

Príklad 3.2: Skript s použitím príkazu `if`

```
#!/bin/bash

SUBOR=$1
if [ -f $SUBOR ]
then
    more $SUBOR
else
    touch $SUBOR; chmod 644 $SUBOR
fi
```

Príkaz case je rovnako príkazom vetvenia, ktorý však použijeme v prípade, že potrebujeme vykonať určitú vetvu skriptu na základe špecifikovaného kľúča.

Príklad 3.3: Skript s použitím príkazu `case`

```
#!/bin/bash

echo -n "Zadaj akciu a nazov suboru: "
read AKCIA SUBOR

case $AKCIA in
    zmaz|remove|delete) rm $SUBOR;;
    vytvor|create)      touch $SUBOR
                        chmod 644 $SUBOR;;
esac
```

Majme skript z výpisu 3.3, ktorý z klávesnice načíta riadok so špecifikáciou príkazu a súboru a vykoná príslušnú akciu. Jeho použitie potom môže vyzerať nasledovne:

```
zabovsky@frios:~/pokus$ ls -la
total 12
drwxr-xr-x 2 zabovsky zabovsky 4096 Sep  4 09:26 .
```

```
drwxr-x--- 6 zabovsky zabovsky 4096 Sep  3 18:14 ..
-rwxr--r-- 1 zabovsky zabovsky  179 Sep  4 09:26 skript
-rw-r--r-- 1 zabovsky zabovsky    0 Sep  4 09:14 subor
zabovsky@frios:~/pokus$ ./skript
Zadaj akciu a nazov suboru: zmaz subor
zabovsky@frios:~/pokus$ ls -la
total 12
drwxr-xr-x 2 zabovsky zabovsky 4096 Sep  4 09:26 .
drwxr-x--- 6 zabovsky zabovsky 4096 Sep  3 18:14 ..
-rwxr--r-- 1 zabovsky zabovsky  179 Sep  4 09:26 skript
zabovsky@frios:~/pokus$ ./skript
Zadaj akciu a nazov suboru: vytvor subor
zabovsky@frios:~/pokus$ ls -la
total 12
drwxr-xr-x 2 zabovsky zabovsky 4096 Sep  4 09:26 .
drwxr-x--- 6 zabovsky zabovsky 4096 Sep  3 18:14 ..
-rwxr--r-- 1 zabovsky zabovsky  179 Sep  4 09:26 skript
-rw-r--r-- 1 zabovsky zabovsky    0 Sep  4 09:26 subor
```

3.4.3 while a until

Príkaz `while` opakovane vykonáva dva zoznamy, kým nie je návratový kód prvého z nich nulový. Použitie príkazu možno ukázať na príklade 3.4, kde skript ktorý čítať používateľský vstup a zapisovať ho na obrazovku, kým nebude zadané slovo koniec.

Príklad 3.4: Skript s použitím príkazu `while`

```
#!/bin/bash

echo -n "Zadaj slovo (<koniec> pre ukoncenie): "
read SLOVO
while [ $SLOVO != "koniec" ]
do
    echo $SLOVO
    echo -n "Zadaj slovo (<koniec> pre ukoncenie): "
    read SLOVO
done
```

Príkaz `until` opakovane vykonáva, podobne ako príkaz `while`, dva zoznamy, kým nie je návratový kód prvého z nich nenulový.

3.4.4 Funkcie

Funkcie sa v skriptoch používajú pre uloženie často sa opakujúcej sekvencie príkazov pod jedným menom. Zabezpečujú modulárne a viacnásobné použitie častí kódu, pričom je možné tieto časti parametrizovať.

Príklad 3.5: Skript s funkciou

```
#!/bin/bash

function zmaz () {
    echo 'Mazem subor $1'
    rm $1
    echo 'Subor $1 bol zmazany'
}

SUBOR=$1
zmaz $SUBOR     # volanie fukncie
```

Poznamenajme, že argumenty sa vo vnútri funkcie stávajú pozičnými parametrami.

3.5 Vstavané príkazy

Vstavané príkazy sú priamo začlenené do shellu. Vykonávajú také činnosti, ktoré by externé programy vykonávať nemohli alebo by ich vykonávali neefektívne. Zoznam vstavaných príkazov uvedieme v stručnom abecednom prehľade:

. a **source** zo zadaného súboru sa v aktuálnom shelli načítajú a vykonajú príkazy

alias príkazom sa nastavujú aliasy (napríklad `l=ls –la`)

bg príkaz zaháji pokračovanie pozastaveného príkazu na pozadí

break príkazom sa ukončujú cykly `for`, `while` a `until`

builtin príkaz vyhľadá zadané meno v zozname vstavaných príkazov a vstavaný príkaz vykoná

cd príkazom sa mení pracovný adresár

command príkaz vykoná príkaz zadaný reťazcom ako vstavaný alebo externý príkaz

continue príkazom sa ukončí vykonávanie aktuálneho cyklu a prejde sa na ďalší

declare a **typeset** deklarujú premenné resp. ich typy

dirs zobrazí obsah zásobníka zapamätaných adresárov

echo opisuje argumenty na štandardný výstup

enable zapína/vypína vykonávanie vstavaných príkazov shellu

eval vykoná argumenty ako príkaz

exec zadaným príkazom sa nahradí shell

exit ukončuje shell

export označuje objekty pre exportovanie do prostredia premenných potomkov

fg	umiestni úlohu do popredia
getopts	pomáha pri spracovaní parametrov príkazového riadku
hash	obsahuje pomocnú pamäť externých príkazov
help	vypisuje help k vstavaným príkazom
jobs	vypisuje aktuálny stav úloh
kill	posiela špecifikovaným procesom zadaný signál
let	vyčísli aritmetický výraz
local	definuje lokálnu premennú funkcie
logout	ukončuje login shell
popd	odstraňuje položku z vrcholu zásobníka ciest k adresárom
pushd	vkladá na vrchol zásobníka adresárov položku alebo rotuje obsah zásobníka
pwd	vypíše cestu k pracovnému adresáru
read	načíta obsahy premenných
readonly	nastaví zadaným menám atribút „iba na čítanie“
return	ukončuje funkciu
set	má niekoľko špecifických významov (viď. `man set`)
shift	posúva obsahy medzi pozičnými parametrami
suspend	pozastaví vykonávanie úloh
test a **[**	vracia návratový kód na základe zadanej podmienky
times	vypíše súčet časov strávených v používateľskej a systémovej fáze shellom a procesmi ním spustenými
trap	definuje, aká akcia sa má vykonať v prípade, že shell príjme zadaný signál
type	príkaz vypíše pre každé zadané meno, ako sa bude interpretovať
ulimit	nastavuje limitné hodnoty pre chovanie systému
umask	nastavuje implicitný formát masky prístupových práv pre vytvárané objekty
unalias	ruší alias
unset	odstraňuje premenné z premenných prostredia

3.6 Záver

V tejto kapitole sme sa zoznámili s technikami súhrnne označovanými ako programovanie v príkazovom interpretri. Takto vytvárané programy - skripty sú neoddeliteľnou časťou operačného systému a ich vytváranie sa pokladá za jednu zo základných programátorských zručností.

3.7 Literatúra

[1] Brandejs M., UNIX - LINUX, GRADA Publishing, 1996
[2] Skočovský L., Principy a problémy operačního systému UNIX, Science, 1993