(c) Petr Bilek <sallyx@centrum.cz>

PROGRAMOVANI V SHELLU

SPUSTENI PRIKAZOVE PROCEDURY - SCENARE

Prikazove procedury lze spustet nekolika zpusoby.
1. prikazem sh, coz je v podstate nove spusteni interpretu prikazu shell.

```
$ sh < program <Enter>
```
(Takto ovsem nelze procedure predavat pozicni parametry z prikazove radky)

```
$ sh program <Enter>
```
(V takto spustene prikazove procedure se mohou vyuzivat pozicni parametry z prikazove radky, program nemusi mit nastaveno pravo pro spusteni - x).

2.
Pokud je prikazova procedura odladena a ma se pouzivat je mozne nastvit souboru s prikazy pristupova prava na spustseni a potom prikazovou procedureu spoustet pouze jejim jmenem. Takto spustene procedure lze opet predavat pozicni parametry z prikazove radky.

```
 $ ls -l program
 -rw-r--r-- 1 krtek hromada 485 Nov 12 11:25 program
 $ chmod u+x program
 $ ls -l program
 -rwxr--r-- 1 krtek hromada 485 Nov 15 11:25 program
 $ program
```

PROMENNE SHELLU

Promenne v shellu jsou jen retezce znaku, tj. cislo 10 je retezec znaku "1" a "2". Promenne se proto nemusi deklarovat (vsechny jsou jednoho typu) - deklaruji se tim, ze se jim priradi hodnota.

jmeno=ret	jmeno je jemeno promenne, ret retezec znaku (=Praha,=,=1) Jmeno je posloupnost znaku (krome znaku mezera, tabulator a $).
jmeno="ret ret"	vyskytuje-li se v retezci mezera ci metaznak, musi se retezec uzavrit do uvozovek.
jmeno=$jmeno2	promenne jmeno je prirazena stejna hodnota, jako promenne jmeno2
'..'(apostrofy)	potlaci specialni vyznam vsech znaku v retezci!
".."(uvozovky)	potlaci specialni vyznam vsech znaku v retezci, ovsem pritom je mezi uvozovkami respektovana substituce priznaku (viz. zpetene apostrovy) a substituce promennych. (Jako v C++)
`..`	priradi do promenne vystup prikazu. adr=`pwd` do promenne adr priradi retez obsahujici jmeno pracovniho adresare. (echo Pracujeme v adresari`pwd`)
unset jmeno	zrusi promenou jmeno
$jmeno	je hodnota promenne jmeno (tedy ret)
readonly jmeno	promenna jmeno bude jen pro cteni. Pokusime-li se ji zmenit, shell zobrazi chybove hlaseni.
readonly	bez argumentu zobrazuje seznam promennych jen pro cteni.

Argumenty z prikazove radky (pozicni parametry)

$0	- promenna obsahujici jmeno spusteneho souboru
$1	- prvni z 9 argumentu uvedenych na prikazovem radku ($1..$9)
$#	- pocet argumentu uvedenych v prikazovem radku
$*	- reprezentuje vsechny argumenty. "$*" je ekvival. "$1 $2 ..."
$@	- reprezentuje vsechny argumenty. "$@" je ekvival. "$1" "$2" ...

Pokud ma soubor vice jak devet argumentu, nemuzeme k nim pristupovat primo. Muzeme vsak pouzit prikaz shift. Timto prikazem se ze druheho argumentu stane prvni (je reprezentovan promennou $1), ze tretiho druhy atd. Soucasne se hodnota $# snizi o 1. Hodnota prvniho argumentu je ovsem nenavratne stracena.

VYHRAZENA JMENA - vnitrni promenne sehellu

Promenne je mozno menit z prikazoveho radku, nebo v souboru ~/.profile.
Nastavuji se prirazenim 'promenna = hodnota' (ovsem jen pro ulitu)

export promenna - nastavena promenna se predava dale i do programu, ktere se z ulity spousteji. 'export promenna = hodnota'
$ - odkaz v prikazove radce na promennou. (tj.'$promenna')
Napriklad hodnotu promenne EDITOR muzeme zjistit prikazem 'echo $EDITOR'.

CDPATH - obsahuje prohledavane cesty pro prikaz cd.
IFS - definuje oddelovace argumentu prikazovaeho radku, standardne to jsou mezera, tabulator a znak noveho radku.
LOGNAME - prihlasovaci jmeno uzivatele
MAIL - jmeno souboru, ve kterem je ulozena elektronicka posta.
MAILCHECK - po kolika vterinach basch zjistuje doslou postu.
MAILPATH - obsahuje seznam jmen souboru oddelene dvojteckou
PATH - urcuje cesty k adresarum, kde ma iterpret hledat prikazy. Implicitne je PATH=:/bin:/usr/bin tzn. pracovni adresar, adresar /bin, adresar /usr/bin a aktualni adresar.
TERM - definuje typ pouziteho terminalu.
TERMINFO - identifikuje adresr, kde se maji hledat inforamce o pouzitem typu terminalu.
EDITOR - Standardni textovy editor, ktery je automaticky spusten nekterymi programi.
HISTSIZE - pocet prikazu, ktere se uchovavaji v prikazove historii.
HISTFILE - jmeno souboru, do nehoz se zapisuje prikaz. hist.
HISTFILESIZE - maximalni pocet radek obsazeny v souboru pro prikazovou historii
RANDOM - obsahuje nahodne cislo
PS1 - prompt. Muze obsahovat nektere spec. sekvence,
jejichz vyznam je tento:
\t presny cas ve formatu HH:MM:SS
\d datum, napr. `Sun Oct 2'
\n prechod na novou radku
\s jmeno ulity, tj. vetsinou 'bash'
\w aktualni adresar (kompletni cesta)
\W aktualni adresar (bez cesty)
\u jmeno aktualniho uzivatele
\h jmeno PC
\# poradi prikazu od spusteni ulity
\! poradi prikazu v prikazove historii
\$ zobrazi $ pro bezneho uzivatele a # pro superuzivatele
\nnn znak, jehoz osmickovy ASCII kod je nnn
\\ obracene lomitko
priklad PS1='\h:\w\$ '.

PS2 - sekundarni prompt, kterym nas shell vyziva, abychom dokoncili prikaz, nevejde-li se na jednu radku.

Nektere z promennych nastavuje automaticky operacni system a obvykle neni dobre je menit. Jsou to napr:
HOME - domaci adresar aktualniho uzivatele.
HOSTTYPE - retezec jednoznacne urcujici typ PC, na kterem je bash spusten.
PWD - aktualni adresar
OLDPWD - Predchozi pracovni adresar

RANDOM - generator nahodnych cisel v rozmezi 0 - 32767(i vice)
TZ=zzzX[ddd] - TZ urcuje casove pasmo. Tuto promennou nastavuje root. zzz je jmeno pasma, X je casovy rozdil v hodinach vzhledem k GMT, ddd je jmeno mistniho casoveho pasma pro zmenu casu.

Prommene aktualizovane dle soucasneho stavu

? navratovy kod posledniho provedeneho prikazu
pocet pozicnich prarametru prikazove procedury
* obsah prikazove radky procedury, ktera je provadena.
$ PID soucasne instalace shellu.
! PID posledniho procesu spousteneho na pozadi
(hodnoty zjistime pomoci echo $? atd..)

VSTUP Z TERMINALU A Z TEXTU PRIKAZOVE PROCEDURY
---
read - read jmeno_promenne - nacte do promenne radek zadany uzivatelem (echo $jm)
- read jmeno1 jmeno2 jmeno3 - nacte tri promenne naraz. Je-li na radku vice promennych, do prvnich promenych se nacte vzdy jedna a do posledni promenne se nactou zbyvajici (promenne jsou retezce!)

Pokud v prikazovem souboru uvedeme jen $jmeno, bude hodnota jmeno interpretovana jako prikaz.
Priklad: echo Zadej povel
read povel
$povel
echo Konec povelu.

<< - presmerovani standardniho vstupu na cast textu prikazove procedury

Format: prikaz << oddelovac ... vstupni radky ... oddelovac
PRIKLAD: (v tomto pripade je oddelovac !)

```
mail $1 <<!
          Prosim zavolejte mi na..
!
echo Zprava odeslana uzivateli $1
```

PRIKLAD:

```
$ cat edg
ed $3 <<%
g/$1/s//$2/g
w
q
%
$ edg unix UNIX kap
```

- tento prikaz meni v souboru kap vsechny vyskyty retezce unix na retezec UNIX. V tomto pripade je oddelovacem retezec %. Shell odesila vse, co je mezi dvema oddelovaci, jako standardni vstup danemu procesu. Muzeme to chapat tak, ze jsme presmerovali standardni vstup do editoru ze souboru, ale tento soubor je uvnitr samotneho skriptu.

Navratove kody

Navratovy kod posledniho provedeneho prikazu je ulozen v promenne ? a proces jej vraci svemu rodicovskemu procesu.
0 znamena, ze prikaz skoncil uspesne, jinak neuspesne.
Navratovy kod muzeme specifikovat pomoci prikazu exit, ktery ukonci vykonavani prikazoveho souboru.

RIDICI STRUKTURY SHELLU

Cyklus for

```
----------
for prom
    [ in seznam_hodnot ]
do
        prvni_prikaz
        druhy_prikaz
        .
        .
        posledni prikaz
done
```

Posloupnost prikazu mezi klicovymi slovy do a done se cykikcy opakuje. Hodnoty ridici promenne prom se prirazuji ze seznamu, ktery je uveden za klicovym slovem in, dokud nejsou vycerpany, nebo neni cyklus nasilne ukoncen.

PRIKLAD: vypise obsah vsech souboru v aktualnim adresary:

```
         for i in *
         do cat $i
         done
```

Seznam hodnot mohou byt argumenty prikazove radky $1, $2, $3, ... Maji-li byt hodnotami vsechny argumenty, potom zapis "for i in $*" ma stejny vyznam jako "for i".

Cyklus while

```
------------

while
        prvni_prikaz
        druhy_prikaz
        .
        .
        .
        posledni_prikaz
do
        prvni_prikaz
        .
        .
        posledni_prikaz
done
```

Posloupnost prikazu mezi klicovymi slovy do a done se cyklicky opakuje dokud posledni prikaz ze seznamu prikazu uvednych za while vraci nulovy navratovy kod, tzn. konci uspechem.

PRIKLAD: program pro vytvoreni vice copii z jednoho souboru. Prvni argument je jmeno kopirovaneho souboru, dalsi argumenty jsou jmena kopii:

```
        while test $# -gt 1
        do
        cp $1 $2
        shift
        done

        B=1
        while [ $B -lt 3 ]; do echo $B; B=$[$B+1]; done
        #volba -lt je vysvetlena nize
```

Cyklus until

```
------------
until testovany_prikaz
do
        prvni_prikaz
        druhy_prikaz
             .
             .
```

```
        posledni_prikaz
done
```

Prikaz while opakuje prikazy tak dlouho, dokud testovany_prikaz vraci hodnotu true. Prikaz until naopak, dokud testovany_prikaz vraci false. Jakmile vrati true, cyklus se ukonci.

PRIKLAD: cyklus cekajici (po 5 minutach testuje ...) na vznik souboru data:

```
        until test -f data
        do
        sleep 300
        done
```

PRIKLAD: srovnani prikazu while a until:

```
        while test $# -ne 0     until test $# -eq 0
        do                      do
        echo $1                 echo $1
        shift                   shift
        done                    done
```

Podmineny prikaz if

```
if <Enter>
        prvni_prikaz
        druhy_prikaz
             .
             .
        posledni_prikaz                 - napr. viz. dale prikaz test
 then <Enter>
        prvni_prikaz
             .
        posledni_prikaz
[ else <Enter>
        prvni_prikaz
             .
        posledni_prikaz <Enter> ]
fi <Enter>
```

Posloupnost prikazu za klicovym slovem then se provede, kdyz posledni prikaz ze seznamu prikazu uvedenych za if vraci nulovy navratovy kod, tzn. konci uspechem.
  Pokud konci neuspechem a konstrukce obsahuje nepovinnou cast else, provede se posloupnost prikazu za else, pokud ji neobsahuje, pokracuje se dasim prikazem za klicovym slovem fi.

Konstrukce prikazu if ve vetvi else lze nahradit vyrazem elif (else if ..).

PRIKLADY:

```
if (test $# = 0)        - zavorky jsou zde pouzity jen pro vetsi nazornost
then
echo Musite zadat aspon jeden argument
exit 1
fi
echo Konec programu
```

POZOR: Tyto programovaci techniky jsou urceny pro Bourne shell

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# showlog: program na zobrazeni souboru logfile bezici pod C-shellem !
# kdyz logfile existuje, zobrazi se.
# Jinak kdyz existuje old. logfile, zobrazi se.
# Jinak se zobrazi chybove hlaseni.

if (-f logfile) then
        echo "Soubor logfile byl nalezen."
        more logfile
elif (-f old.logfile) then
```

```
        echo "Byl nalezen pouze soubor old.logfile."
        more old.logfile
else
        echo "Soubor logfile nebyl nalezen"
        echo "Konzultujte se systemovym administratorem."
fi
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
....................................
echo -e "slovo1: \c"    # v nekterych versich schellu se uziva echo -n "..."
read s1
echo -e "slovo2: \c"
read s2
if (test "$s1" = "$s2")
then
echo Jsou stejne
else
echo Jsou ruzne
fi
echo Konec programu
....................................

prikaz1 && prikaz2      - prikaz2 se vykona pouze tehdy, vrati-li prikaz1 true
prikaz1 || prikaz2      - prikaz2 se vykona, vrati-li prikaz1 hodnotu false

(Tyto prepinace lze s vyhodou pouzivat i na prikazove radce. Napr:
$ sort bigfile > temp && mv temp bigfile)

Prepinac case
-------------

case ret <Enter>
  in <Enter>
    vzor_1) <Enter>
           prvni_prikaz
                .
                .
           posledni_prikaz
           ;; <Enter>
      .
      .
    vzor_n) <Enter>
           prvni_prikaz
                .
                .
           posledni_prikaz
           ;; <Enter>
         *) <Enter>
           prvni_prikaz
                .
                .
           posledni prikaz
           ;; <Enter>
esac

  Provede se pouze ta vetev, kde se vzor shoduje s textovym retezcem ret.
Pokud neni shoda s zadnym vzorem, tak vzor * je splnen vzdy (obdoba
expanzniho znaku *, ktery nahrazuje libovolny retezec znaku). Dalsi pouzitelne
vzory jsou : ? (jeden znak) [..] (prave jeden ze znaku uvnitr zavorky - pomlcka
urcuje interval znaku) a | (oddeluje jednotlive moznosti, ktere vyhovuji urcite
vetvi)
PRIKLAD: echo "Chcete soubor opravdu vymazat? [a/n]: \c!:"
         read znak
         case $znak in
         a|A) rm soubor ;;
```

```
        n|N) ;;
        *) echo "Neplatna odpoved" ;;
        esac
```

>>

ODDELOVANI A SDRUZOVANI PRIKAZU

Shellu musime nejak zdelit, kde konci jeden prikaz a zacina druhy. Jednim z oddelovacu je znak "novy_radek", tj. <Enter>. Ekvivalentni vyznam ma strednik (;). Prikaz $a;b;c je totozny s prikazy $a

```
                                        $b
                                        $c
```

Dalsi oddelovac je kanal (pipe) |

```
$ a|b|c         - spusti prikaz a, jeho vystup preda prikazu b atd. po skonceni
                  vsech prikazu dostaneme prompt
```

Dalsim oddelovacem je symbol ulohy bezici na pozadi.

```
$ a&b&c         - vykona ulohy a a b na pozadi a c na popredi. Prompt dostaneme
                  po vykonani ulohy c.
$ a&b&c&        - vsechny ulohy bezi na pozadi a prompt dostaneme ihned.
$ a|b|c&        - vsechny ulohy jsou spusteny jako jedna na pozadi. Prompt
                  dostaneme okamzite.

$ (a;b)&c       - zavorky zdruzuji prikazy tak, ze je shell chape jako jednu
skupinu prikazu, se kterou zachazi jako s jednou
                  ulohou. Prikazy a a b se vykonaji postupne na pozadi, zatimco
                  prikaz c pobezi na popredi.
$ (a;b)&(c;d)&  - postupne se vykonaji prikazy a a b na pozadi a soucasne se na
                  pozadi budou postupne vykonavat prikazy c a d. Prompt
                  obdrzime okamzite.
```

Existuji dve moznosti sdruzovani prikazu: pomoci obycejnych zavorek a pomoci slozenych zavorek. Obycejne zavorky rozdvoji puvodni shell, vykonaji se pozadovane prikazy a rizeni se vrati puvodnimu shellu. Pokud jsou prikazy sdruzeny pomoci slozenych zavorek {}, prikazy se vykonaji v ramci aktualniho shellu. Rozdil dokumentuje nasledujici prikaz:

```
$ (pwd;cd /bin;pwd);pwd
/home/jelen
/bin
/home/jelen
$ { pwd;cd /bin;pwd;};pwd       # pozn. - prvni mezera a posledni ; v {} nutne!
/home/jelen
/bin
/bin
```

PRIKAZY SHELLU

PROCESY -viz. dokument root.

```
seq 1 9                 vytiskne radu cisel od 1 do 9
# libovolny text        vse za znakem # je chapano jako poznamka a neni
                        interpretovano jako prikaz.
prikaz >/dev/null       Pouzijeme-li tuto konstrukci, vystup z prikazu skonci v
                        kosi, kterym je specialni soubor /dev/null.
sleep n                 pozastavi program na "n" sekund.
echo zprava...          zkopiruje svuj standardni vstup na standardni vystup
echo -e "zprava \c"     \c zpusobi, ze se neprejde na novy radek (pouze s -e !)
echo -n zprava          -n zpusobi, ze se neprejde na novy radek (plati)
echo $HOME              vypise se hodnota promenne HOME (tj. domovsky adresar)
set                     vypise vsechny promenne a jejich hodnoty
set arg1 arg2 ...       priradi argumenty arg1,arg2 ... promennym $1,$2, ...
Priklad: (soubor dateset)               (vysledek)
set `date`
```

```
echo $*
echo                                    Mon Apr 18 20:35:10 CET 1994
echo "Argument 1: " $1                  Argument 1: Mon
echo "Argument 2: " $2                  Argument 2: Apr
echo "Argument 3: " $3                  Argument 3: 18
echo "Argument 4: " $4                  Argument 4: 20:35:10
echo
echo $2 $3, $6                          Apr 18, 1994
```

```
exit cislo              ukonci program a vrati rodicovskemu programu navratovy
                        kod zadaneho cisla.
return cislo            to same jako exit.
```

Prikazy test a expr

Prikaz test je pomocnym prikazem pro ridici struktry shellu. Jeho navratova hodnota je dana vyhodnocenym vyrazem. Ma format: test vyraz < Enter >
kde vyraz muze vyjadrovat vztahy textovych retezcu a atributy souboru. Pomoci operatoru ! muzeme kazdy podvyraz negovat.
priklady vyrazu:

```
                HODNOTA VYRAZU JE TRUE, KDYZ:
-r soubor       soubor existuje a je dostupny (ma pravo) pro cteni
-w soubor       soubor existuje a je dostupny pro zapis
-x soubor                      a ma povoleny pristup pro spousteni
-f soubor                      a je to obycejny soubor
-d soubor                      a je adresarem
-c soubor                      a je to specialni znakovy soubor
-b soubor                      a je to blokovy specialni soubor
-p soubor                      a je to pojmenovany kanal (pipe)
-u soubor                        ma nastaven set user ID bit
-g                               ma nastaven set group ID bit
-k                               ma nastaven sticky bit
-s                             a jeho velikost je vice nez 0 bajtu
-t deskriptor   otevreny soubor s cislem deskriptoru deskriptor je spojen s
                terminalem. Pokud deskriptor neni uveden, predpoklada se 1
                (standardni vystup).

s1              retezec s1 neni prazdny retezec
-z s1           delka textoveho retezce s1 je nulova
-n s1                                  s1 je vetsi nez nula
s1 = s2         retezce s1 a s2 jsou si rovny
s1 != s2        retezce s1 a s2 si nejsou rovny
vyraz -a vyraz2 oba vyrazy jsou pravdive ( logicky soucin )
vyraz -o vyraz2 alespon jeden vyraz je pravdivy ( logicky soucet )
n1 op n2        vyhodnoceni algebraickeho porovnani celych cisel. Za op lze
                dosadit:
                -eq     rovnost
                -ne     nerovnost
                -gt     vetsi nez
                -ge     vetsi nebo rovno
                -lt     mensi nez
                -le     mensi nebo rovno
```

```
Pozn:   Prikaz  test vyraz
        lze v novych shellech nahradit konstrukci [ vyraz ] (mezery nutne!)
napr: if test "s1" != "s2"; then
      if [ "s1" != "s2" ]; then
```

Druhym pomocnym prikazem je expr, ktery vyhodnocuje sve argumenty jako aritmeticky vyraz. Ma fomrat: expr arg ....
Kazdy clen vyrazu je povazovan za zvlastni argument arg (musi byt oddeleny mezerou). Argumenty mohou vyjadrovat nasledujici operace:

```
op + op         soucet argumentu
op - op
op * op
op / op         celociselne deleni
op % op         zbytek po celociselnem deleni
op rop op       porovnani rop je jeden z relacnich operatoru <,<=,=,!=,>=,>
```

Porovnani v relacnim vyrazu je numericke, pokud jsou oba argumenty celociselne, jinak lexikograficke (podle ASCII).

Priklad pouziti: (zvetseni promenne o 1)

```
krok=`expr $krok +1`
```

Konstrukce prikazu expr musi byt v obracenych apostrofech, protoze produkuje vysledek na standardni vystup. Jedna se tedy o klasicke nahrazeni prikazu svym vystupem.

```
expr "vyraz"    - slouzi k vyhodnocovani vyrazu. Vyraz je slozen z dalsich
                  vyrazu a retezcu oddelenych mezerou. Poradi priorit lze
                  menit zavorkami.
vyrazy (dle prioryty):
:       operator porovnani - porovnava dva retezce, konci porovnanim posledniho
                             znaku druheho retezce. Pokud dojde ke shode,
                             zobrazi se pocet znaku 2. retezce, jinak 0.
*       operator nasobeni  --------
/       operator deleni           |
%       operator zbytek po deleni |--- tyto oper. pracuji jen s cislicemi.
+               scitani           |
-               odecitani ---------
<, <=, = !=, >=, >         -------- tyto operatory pracuji jak s numerickymi,
                           tak i s nenumer. argumenty.(vystupem je 1, nebo 0)
&       logicky soucin  - pokud zadny z argumentu nema hodnotu 0 nebo to neni
                          prazdny retezec, zobrazi hodnotu 1. argumentu. Tento
                          operator se musi oznacit.
|       logicky soucet  - vyhodnoti svuj prvni argument. Jestlize to neni 0 ani
                          prazdny retezec, zobrazi hodnotu 1. argumentu. V opac-
                          nem pripade zobrazi hodnotu 0. Operator se musi
                          oznacit.
priklady: $ expr 21 + 9 '*' 2 / 6
          24
          $ echo $x
          3
          $ x= `expr $x = 1`
          $ echo $x
          4
          $ expr bojkot : boj
          0
          $ X=abc
          $ expr $X : '.*'
          3                     /* pocet znaku v retezci X */
```

```
Pozn:   Prikaz expr vyraz
        lze nahradit konstrukci $((vyraz)) ,ktera je mnohem rychlejsi
napr:   x= `expr $x + 1`
        x=$(($x+1))
```

Prikazy break , continue a : (dvojtecka)

```
break -         nestandardni ukonceni cyklu. Zpusobi vystup z cyklu a
                pokracovani interpretace prikazove procedury za telem cyklu.

continue -      zpusobi ignorovani zbytku tela cyklu a prichod na dalsi
                iteraci (dalsi pruchod cyklem).
```

```
: (dvojtecka) - prikaz, ktery neprovadi zadnou cinnost a vraci kod true.
                Vyuziva se napr. v cyklech while nebo until nebo v prikazu
                trap.
                PRIKLAD while :
                          do
                          ..
                          break
                          ..
                          done

Prikaz trap
-----------
trap [[posloupnost_prikazu] seznam_signalu]
- tento prikaz zachytava signaly, tj. zpravy zaslane procesu, ktere proces
  inforumuji o vyskytu nejake udalosti.
  Nejcastejsi signaly: 0 - ukonceni shellu ( z libovolnych duvodu )
                       1 - odpojeni se od terminalu
                       2 - preruseni z klavesnice
                       3 - konec s ulozenim obsahu pameti
                       9 - nasilne bezpodminecne ukonceni procesu
                           POZOR! Tento signal nelze chytit ani ignorovat.
                      15 - softwarove ukonceni.
 Posloupnost prikazu je jediny argument, a proto je-li prikazu vice, musi byt
 omezeny uvozovkami nebo apostrofy. Seznam signalu jsou cisla signalu, pri
 jejichz chyceni se provedou dane prikazy.
 Pokud shell chyti dany signal, prerusi vykonavani skriptu, provede zadanou
 posloupnost prikazu a pokracuje v miste, kde byl skript prerusen.

 PRIKLAD: trap "echo PROGRAM PRERUSEN; exit 1 " 2
          while (true)
          do
          echo PROGRAM PRACUJE
          sleep 10
          done

Pokud bezi program na popredi, klavesa <DEL> vygeneruje signal cislo 2 a
program ukonci. V nasem skriptu se misto toho signal zachyti a provedou se
dva prikazy v uvozovkach.

PRIKLAD: $ (trap '' 1; dlouhotrvajici_program)&

Prikazy trap a dlouhotrvajici_program budou vykonany v samostatnem shellu - to
zpusobi zavorky a pobezi na pozadi. Pokud se uzivatel odloguje ( odpoji se od
terminalu), program provede prazdny prikaz '' a pokracuje dal. Normalne by byl
ukoncen!

Prostredi a exportovani promennych
----------------------------------
. (tecka) -     vykona prikazovy soubor jako soucast aktualniho procesu a
                pracuje s promennymi rodicovskeho procesu. Zmeni-li takovou
                promennou, pri skonceni dostane rodicovsky proces tuto novou
                hodnotu. Hodnota promenne $0 je i pri provadeni detskeho
                procesu jmeno rodice!
Priklad:        echo $0 PID=$$
                . id2
                echo Tento text se zobrazi.
(Soubor id2 nemusi mit nastaveno pravo pro spousteni. Muze vypadat treba takto:
                echo id2:PID=$$ )

source /cesta/soubor    - provedou se prikazy souboru .. totez co "tecka" vise

exec -          puvodni proces je prekrit novym procesem a do puvodniho procesu
                se uz nemuze nikdy vratit. Prikaz exec vyzaduje, aby soubor mel
```

prideleno pravo spusteni.

Priklad:
```
echo $0 PID=$$
exec id2
echo Tento text se nezobrazi
```

Pozn.: soubor je take mozno spustit jen pouzitim jeho jmena (bez tecky ci rikazu exec). Takto spusteny proces ovsem neprejme deklarovane promenne od sveho rodice - musi se pouzit prikaz export.

export jmeno tento prikaz preda hodnotu promenne jmeno ostatnim procesum.

Priklad:

```
$ cat extest
jazyk=anglictina
echo $0 $$ $jazyk
$ cat zmena
echo $0 $$ $jazyk
jazyk=cestina
echo $0 $$ $jazyk
$ extest
extest 12252 anglictina
zmena 12253
zmena 12253 cestina
extest 12252 anglictina
```

```
$ cat extest1
export jazyk
jazyk=anglictina
echo $0 $$ $jazyk
zmena
echo $0 $$ $jazyk
$ extest1
extest1 12258 anglictina
zmena 12259 anglictina
zmena 12259 cestina
extest3 12258 anglictina
```

```
$ cat extest2
jazyk=anglictina
echo $0 $$ $jazyk
. zmena
echo $0 $$ $jazyk
$ extest2
extest2 12264 anglictina
extest2 12264 anglictina
extest2 12264 cestina
extest2 12264 cestina
```

==========

==========