Programování v C shellu

Podmíněné příkazy

Příkaz if

Syntaxe příkazu má následující varianty.

```
if ( výraz ) příkaz
```

*Sémantika:*Vyhodnotí se *výraz* a pokud je jeho hodnota **true** (tj. různá od 0), provede se *příkaz*.

```
if ( výraz ) then
  příkazy
endif
```

*Sémantika:*Vyhodnotí se *výraz* a pokud je jeho hodnota **true** (tj. různá od 0), provedou se *příkazy*.

```
if ( výraz ) then
  příkazy1
else
  příkazy2
endif
```

Sémantika: Vyhodnotí se *výraz* a pokud je jeho hodnota **true** (tj. různá od 0), provedou se *příkazy1*. Pokud je jeho hodnota **false** (tj. rovna 0) , provedou se *příkazy2.*

```
if ( výraz1 ) then
  příkazy1
else if ( výraz2 ) then
  příkazy2
else
  příkazy3
endif
```

Sémantika: Vyhodnotí se *výraz1* a pokud je jeho hodnota **true** (tj. různá od 0), provedou se *příkazy1*. Pokud je jeho hodnota **false** (tj. rovna 0) , vyhodnotí se *výraz2***.** Pokud je hodnota *výrazu2* **true**, provedou se *příkazy2.* Pokud není, provedou se *příkazy3*.

Příklad 1: Následující podmíněný příkaz vypíše, zda je obsah proměnné x dělitelný 7.

```
#! /bin/csh
set x=37
if ( $x % 7 ) then
  echo "promenna x neni delitelna 7"
else
  echo "promenna x je delitelna 7"
endif
```

Podmínka příkazů **if** a **while** (o příkazu **while** viz dále) je vyjádřena výrazem. Je-li hodnota výrazu různá od nuly, je podmínka splněna, jeli rovna nule, podmínka splněna není.

Výrazy – file inquiry operátory

Jak se vytvářejí výrazy bylo vysvětleno v minulé kapitole. Syntaxe výrazů je ale bohatší, než bylo v minulé kapitole řečeno. Součástí výrazu totiž ještě může být jednoduchý výraz utvořený pomocí tzv. ***file inquiry*** operátoru a názvu souboru. Obvykle jsou k disposici file inquiry operátory **r , w , x , e , f , d** .

Jednoduchý výraz utvořený pomocí *file inquiry* operátoru má následující syntaxi:

-file_inquiry_operátor jméno_souboru

Sémantika *file inquiry* operátorů je následující. Výraz má hodnotu 1 pokud:

-e	*soubor*	*soubor* existuje
-f	*soubor*	*soubor* je normální soubor
-z	*soubor*	*soubor* má délku 0
-d	*soubor*	*soubor* je adresář
-o	*soubor*	uživatel *soubor* vlastní
-r	*soubor*	*soubor* má nastaveno přístupové právo **r**
-w	*soubor*	*soubor* má nastaveno přístupové právo **w**
-x	*soubor*	*soubor* má nastaveno přístupové právo **x**

V opačném případě má výraz hodnotu 0.

Následující výrazy jsou proto správně utvořenými výrazy C shellu:

```
( $1 > 20 ) && ( -d $file )
( ( $1 * $2 ) > 10 ) || ( -e $file )
```

File inquiry operátory lze sdružovat. Například lze psát

```
( -fo $file1 )
```

Tento výraz je splněn pokud soubor, jehož jméno obsahuje proměnná `file1` je obyčejný soubor, který vlastní vlastník shellu, který script provádí.

Příklad 2: Následující skript testuje, zda obsah proměnné `file` označuje v systému existující adresář nebo zda obsah proměnné `file` označuje v systému existující normální soubor, který vlastní uživatel, který script spouští.

```
#! /bin/csh
set file=/etc/
if ( -d $file ) echo ”soubor $file je adresar”
if ( -fo $file ) then
  echo ” $file je vlastni normalni soubor”
else
  echo ” $file je neco jineho”
endif
```

Výběr z více alternativ

Příkaz switch

Syntaxe

```
switch (řetězec)
  case vzor1 :
     příkazy
     [breaksw]
  case vzor2 :
     příkazy
     [breaksw]
  . . . . . .
  default :
     příkazy
     [breaksw]
endsw
```

Sémantika: Řetězec je obvykle zadán jako hodnota proměnné. *Vzory* jsou výrazy , které obsahují normální znaky a expansní znaky * , ? , [,] . Znak * označuje libovolný řetězec, ? libovolný znak a znaky [] alternativu. Postupně se berou jednotlivé *vzory* a zkoumá se, zda jim *řetězec* vyhovuje. Pokud řetězec *vzoru* vyhovuje, provedou se následující *příkazy* a pokud posledním příkazem není příkaz **breaksw**, porovnávání řetězce a vzorů pokračuje. Pokud řetězec nevyhovuje žádnému *vzoru*, provedou se *příkazy* uvedené za klíčovým slovem **default**.

Parametry příkazového řádku

Parametry příkazového řádku jsou v C shellu přístupné dvojím způsobem:

- Jako poziční parametry *0, 1, . . . 9* (další s pomocí příkazu **shift**)
- Jako prvky seznamu slov `argv`: `$1` je totéž co `$argv[1]` , `$2` je totéž co `$argv[2]` atd. Při použití tohoto způsobu označení parametrů není třeba používat příkaz **shift** . Celkový počet parametrů příkazového řádku lze označit jako `#argv` a všechny parametry příkazového řádku jako `argv`.

Příkaz shift

Syntaxe příkazu je následující

`shift` [*proměnná*]

Sémantika: Provede se posuv slov, které *proměnná* obsahuje, o jednu posici vlevo a sníží se hodnota obsažená v proměnné *#proměnná* o 1 (hodnota v proměnné *#proměnná* se rovná počtu slov proměnné *proměnná*). Pokud *proměnná* není explicitně uvedena, provede se posun hodnot v proměnné `argv`. Pokud se provádí příkaz **shift** na proměnné, která obsahuje nulový počet slov, příkaz končí chybou.

Příklad 3: Následující příkaz zkontroluje, zda první parametr, se kterým byl script spuštěn začínal na `-a` nebo `-l` . Pokud ano, vypíše: `prvni parametr O.K.` Pokud ne, vypíše : `nepovolena hodnota prvniho parametru`

```
#! /bin/csh
switch ( $argv[1] )
     case -a*:
          echo ”prvni parametr O.K”
          breaksw
     case -l*:
          echo ”prvni parametr O.K.”
          breaksw
     default:
          echo ”nepovolena hodnota prvniho parametru”
endsw
```

Vytváření cyklů

Příkaz foreach

Syntaxe příkazu je následující

```
foreach proměnná (seznam_slov)
  příkazy
end
```

Sémantika: Při každém průběhu cyklu je nejdříve *proměnné* přiřazeno další slovo ze *seznamu_slov* a pak jsou provedeny uvedené *příkazy*. Cyklus skončí po vyčerpání všech položek *seznamu_slov*.

Příklad 4: Skript obsahuje následující příkazy

```
#! /bin/csh
set buffer1=(jedna dva tri ctyri pet sest sedm)
set buffer2=(1 2 3 4 5 6 7)
set pointer=1
foreach slovo ($buffer1)
  set buffer2[$pointer]=$slovo
  @ pointer++
  echo $buffer2
end
```

Tento úsek skriptu zkopíruje seznam slov uložený v proměnné `buffer1` do proměnné `buffer2`.

K přesunu seznamu slov z proměnné `y` do proměnné `x` nestačí napsat

```
x =$y
```

Při tomto zápisu by se do `x` překopírovalo pouze první slovo. Je třeba nejdříve definovat velikost cílové proměnné a potom pomocí cyklu do ní seznam slov překopírovat.

Příklad 5: Napište skript, který vypíše obsah všech souborů domácího adresáře, které začínají znaky `.c` .

Řešení

```
#!/bin/csh
cd ~
foreach x (~/.c*)
     if( -f $x ) then
          echo "******** $x ********"
          cat $x
     else
          echo "$x neni obycejny soubor"
     endif
end
```

Příklad 6: Napište script, který vypíše počet adresářů a normálních souborů obsažených v zadaném adresáři. Pokud adresář není zadán, vypíše se počet adresářů a normálních souborů obsažených v aktuálním adresáři.

Řešení

```
#!/bin/csh
if ( $#argv == 0 ) then
  set dir="."
else
  set dir=$argv[1]
endif
if ( ! -d $dir ) then
  echo "$0 : $dir neni adresar"
  exit 1
endif
@ fc=0
@ dc=0
cd $dir
foreach file (*)
  if ( -f $file ) then
     @ fc++
  else if ( -d $file ) then
     @ dc++
  endif
```

```
end
echo "$dir : $fc souboru $dc adresaru "
```

Změna řízení uvnitř cyklu

break Řízení se předá instrukci následující za instrukcí cyklu.

continue Přeskočí se zbytek instrukcí uvnitř cyklu a řízení se vrátí na počátek cyklu.

goto *label* Provede se skok na řádku, která začíná *label :*

exit *číslo* Ukončení skriptu a nastavení návratového kódu na hodnotu *číslo*

Příkaz while

Syntaxe příkazu je následující

```
while ( výraz )
  příkazy
end
```

Sémantika: Provádějí se uvedené *příkazy* dokud je výraz pravdivý.

Příklad 7: Pomocí příkazů **while** a **shift** napište script `opis`, který opíše argumenty příkazového řádku.

Řešení

```
#!/bin/csh
while ( $#argv )
     echo $argv[1]
     shift
end
```

Příklad 8: Upravte skript z příkladu 1, který spočte počet adresářů a obyčejných souborů v daném adresáři tak, aby mu bylo možno zadat požadavek na zpracování více adresářů najednou.To jest musí být možné jej spustit např takto:

```
pocetadr  /bin   /etc  /home
```

Použijte k tomu příkazy ***while*** a ***shift***. Nezapomeňte také, že v případě, že nezadáte žádný parametr, skript musí zpracovat aktuální adresář.

Řešení

```
#!/bin/csh
#nums /verse 2/: script vypise pocet souboru a adresaru
if ( $#argv == 0 ) then
  set dir="."
else
  set dir=$argv[1]
endif
while(1)
     if ( ! -d $dir ) then
          echo "$0 : $dir neni adresar"
     else if  ( ! `ls $dir | wc /c` ) then
          echo "$0 : adresar $dir je prazdny"
     else
          echo "$dir : "
          @ fc=0
          @ dc=0
          cd $dir
          foreach file (*)
               if ( -f $file ) then
                    @ fc++
               else if ( -d $file ) then
                    @ dc++
               endif
          end
          echo "$fc souboru $dc adresaru"
     endif
     if ( $#argv <= 1 ) then
          break
     else
          shift
          set dir=$argv[1]
     endif
end
```

Přesměrování v C-shellu

V C shellu jsou možnosti přesměrování omezenější. Jsou možná tato přesměrování:

<x	*Přesměrování standardního vstupu do soubor* x
>x	*Přesměrování standardního výstupu do souboru* x
>&x	*Přesměrování standardního i chybového výstupu do souboru* x

V obou shellech lze znak > zdvojit, tj. psát například >>x, což znamená přesměrování s připojením. Výstup z programu se nezačne psát od počátku souboru x , tj. obsah souboru x se nepřepíše, ale výstup z programu se připojí za konec souboru x.

Příklad 9: Napište script, který:

- Zkontroluje, zda studenti nezabírají více než v parametrovém řádku zadanou velikost diskové paměti.
- Pokud ano, script zapisuje výsledek pravidelné kontroly obsazení diskové paměti studenty do souboru ~/studenti .
- Script napište tak, aby po svém spuštění na pozadí tuto kontrolu prováděl průběžně po zadaném časovém intervalu.
- Velikost diskové paměti se scriptu zadá při spuštění jako 1. parametr a velikost časového intervalu se scriptu zadá při spuštění jako 2. parametr.

Řešení

```
#!/bin/csh
if ( $#argv != 2 ) then
     echo "pouziti: $0 pamet interval_spousteni"
     exit 1
endif
while(1)
     date >>~/studenti
     (/bin/du -sk /home/x* | awk '( $1'" > $1 )" >>~/studenti)
       >&/dev/null
     sleep $2
end
```

Ošetření příchodu signálu

Script může být také ukončen vynuceným způsobem, to jest zasláním signálu procesu, který jej provádí. Při tomto nestandardním ukončení je často vhodné ještě před tím, než script svoji činnost skončí, provést některé akce.

Aby bylo možné po příchodu signálu provést ještě nějaké příkazy, je třeba mít možnost příchod signálu ošetřit. C shell umožňuje nastavit ošetření příchodu signálů příkazem **onintr** . Syntaxe příkazu je následující:

onintr *label*

Po příchodu libovolného signálu, který by jinak znamenal okamžité ukončení činnosti scriptu, se řízení předá příkazu, který je označen návěštím *label:* (samozřejmě toto ošetření se nevztahuje na signál 9, který ošetřit nelze). Sekvence příkazů, která se má vykonat po příchodu signálu, se obvykle zařazuje na konec skriptu. Pokud má skript po vykonání této sekvence příkazů skončit, musí jejím posledním příkazem být příkaz **exit** .

Například:

```
onintr catch
    Příkazy skriptu
catch:
    Příkazy
    exit 1
```

Příkaz **onintr** může mít ještě následující syntaxi

```
onintr -
```

Znamená to, že signály budou ignorovány

Je také možná syntaxe

```
onintr
```

Tento zápis příkazu **onitr** znamená návrat k původnímu nastavení, tj. signály nebudou nadále ošetřeny.