Programování v Bourne shellu

Sekvenční a podmíněné provádění

Sekvenční provádění znamená vykonávání jednoho příkazu za druhým bez ohledu na okolnosti. Pro oddělení příkazů při sekvenčním provádění se používá středník. Každý příkaz takovéto sekvence, zleva doprava, je proveden bez ohledu na úspěšnost předchozího příkazu.

Příklad 1:

```
$mkdir Flinstonovi ; cd Flintosnovi ; touch Barney
```

První příkaz byl úspěšně proveden, druhý byl díky překlepu neúspěšný. Třetí příkaz byl ale přesto proveden a vytvořil soubor Barney.

Podmíněné provádění umožňuje naopak provádět odlišné příkazy v závislosti na okolnostech, tedy na základě úspěšnosti či neúspěšnosti předchozích příkazů.

Návratový kód

Interpret potřebuje vědět, zda se poslední příkaz provedl úspěšně či nikoliv. Když program ukončí svoji činnost, vrátí zpět nějaké číslo – ***návratový kód***. Nula většinou znamená úspěšné provedení příkazu, zatímco jiné číslo indikuje chybu.

Interpret zpřístupňuje návratový kód posledně provedeného příkazu v proměnné **?** (otazník). Ke zjištění návratového kódu lze použít příkaz **echo $?**.

Příklad 2:

```
$mkdir tmp
$echo $?
0

$cd Dracula
bash: Dracula: No such file or direktory
$echo $?
1

$Dracula
Bash: Dracula: command not found
$echo $?
127
```

Operátory &&, || a !

Tyto operátory se používají podobně jako sekvenční provádění, středník je ale nahrazen příslušnými symboly. K provedení příkazu za předpokladu, že předchozí příkaz byl proveden úspěšně, se oddělí příkazy znaky &&. K provedení příkazu v případě neúspěchu předchozího příkazu se použijí znaky ||.

Příklad 3:

```
$cd smoulove && echo "Adresar existuje" || mkdir smoulove &&
 echo "Adresar neexistoval, vytvořil jsem novy"
Adresar neexistoval, vytvořil jsem novy

$cd smoulove && echo "Adresar existuje" || mkdir smoulove &&
 echo "Adresar neexistoval, vytvořil jsem novy"
Adresar existuje
```

Operátor ! není totéž, co historie příkazů. Obrátí význam návratového kódu příkazu následujícího za operátorem. Za operátorem musí následovat mezera, aby systém rozpoznal, kdy je ! ve významu negace a kdy ve významu historie.

Příklad 4:

```
$echo "Test"
Test
$echo $?
0

$ ! echo "Test"
Test
$echo $?
1
```

Příkaz if

Příkaz má následující syntaxi:

```
if příkaz
  then příkazy
fi
```

Poznamenejme, že příkazy následující po then se obvykle zapisují do řádek samostatně. Pokud je jich zapsáno na jednom řádku více najednou, je třeba je oddělit středníkem. První z příkazů může být na stejném řádku jako then nebo může být až na řádku následujícím. *Sémantika:* Provede se příkaz a pokud je jeho návratová hodnota rovna 0 (true), provedou se příkazy následující po klíčovém slově then . Pokud je návratová hodnota různá od nuly (false), příkazy se neprovedou.

```
if příkaz
  then příkazy1
  else příkazy2
fi
```

Sémantika: Provede se příkaz a pokud je jeho návratová hodnota rovna 0 (true), provedou se pouze příkazy1, následující po klíčovém slově then . Pokud je návratová hodnota různá od nuly (false), provedou se pouze příkazy2 následující po klíčovém slově else.

```
if příkaz1
  then příkazy1
  elif příkaz2
    then příkazy2
  else příkazy3
fi
```

Sémantika: Provede se příkaz1 a pokud je jeho návratová hodnota rovna 0 (true), provedou se pouze příkazy1 následující po klíčovém slově then . Po provedení příkazů1 provádění příkazu if skončí. Pokud je návratová hodnota příkazu1 různá od nuly (false), provede se příkaz2 následující po klíčovém slově elif (zkratka else if). Pokud je návratová hodnota příkazu2 0 (true), provedou se pouze příkazy2. Pokud je návrátová hodnota příkazu2 různá od nuly (false), provedou se příkazy3. Poznamenejme, že v příkazu může být více řádek elif . Naopak řádek else může v příkazu chybět. Sémantika těchto variant příkazu if je zřejmá.

Příkaz case

Syntaxe příkazu je následující:

```
case řetězec in
  vzor_1) příkazy_1 ;;
  vzor_2) příkazy_2 ;;
  . . . . . . . . . .
  vzor_n) příkazy_n ;;
esac
```

Příkazy se mezi sebou oddělují středníkem.

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. Vzorem také může být více alternativních vzorů. Alternativní vzory jsou odděleny znakem | . Postupně se berou jednotlivé vzory a zkoumá se, zda jim řetězec vyhovuje. Pokud řetězec vzoru vyhovuje, provedou se následující příkazy až po ukončující znaky ;; a činnost příkazu case se ukončí.

Příkaz test

Příkaz test lze zapsat dvojím způsobem:

```
test výraz
```

nebo

```
[ výraz ]
```

První syntaxi příkazu lze použít ve všech verzích shellu, druhou pouze v takovém shellu, který obsahuje test jako vnitřní příkaz. Příkaz test zjistí, zda je výraz splněn. Pokud ano, skončí se statusem 0. Pokud není, skončí se statusem 1.

File inquiry operátory v Bourne shellu

-f *soubor* soubor existuje a jedná se o obyčejný soubor
-x *soubor* soubor existuje a jedná se o spustitelný soubor
-r *soubor* soubor existuje a lze jej číst
-w *soubor* soubor existuje a lze do něj zapisovat
-s *soubor* soubor existuje a má nenulovou délku (není prázdný)
-d *soubor* soubor existuje a jedná se o adresář

Operátory pro porovnávání číselných hodnot v Bourne shellu

výraz1 **-eq** *výraz2* hodnota výrazu1 se rovná hodnotě výrazu2
výraz1 **-ne** *výraz2* hodnota výrazu1 se nerovná hodnotě výrazu2
výraz1 **-ge** *výraz2* hodnota výrazu1 je větší nebo rovna hodnotě výrazu2
výraz1 **-le** *výraz2* hodnota výrazu1 je menší nebo rovna hodnotě výrazu2
výraz1 **-gt** *výraz2* hodnota výrazu1 je větší než hodnota výrazu2
výraz1 **-lt** *výraz2* hodnota výrazu1 je menší než hodnota výrazu2

Operátory pro porovnávání řetězců v Bourne shellu

-n *řetězec* řetězec existuje
-z *řetězec* řetězec neexistuje
řetězec1 **=** *řetězec2* řetězec1 je identický s řetězcem2
řetězec1 **!=** *řetězec2* řetězec1 není identický s řetězcem2

Logické operátory v Bourne shellu

! *výraz* negace
výraz1 **-o** *výraz2* disjunkce
výraz1 **-a** *výraz2* konjunkce

Příklady

Příklad 5: Příklad na otestování souborů.

a) `[ -f /bin/ls ];echo $?`

b) `[ -d /bin/ls ];echo $?`

Příklad 6: Příklad na porovnávání číselných hodnot.

```
export x=17
export y=7
```

a) `[ $x -eq `expr $y + 10` ];echo $?`

b) `[ $x -gt $y ];echo $?`

c) `[ $x -le $y ];echo $?`

Příklad 7: Příklad na porovnávání řetězců.

```
export x=abc
export y=abd
```

a) `[ $x = $y ];echo $?`

b) `[ $x != $y ];echo $?`

c) `[ $x = "abc" ];echo $?`

d) `[ -n "$x" ];echo $?`

e) `[ -z "$x" ];echo $?`

Příklad 8: Příklad na logické operátory.

a) `[ -d $HOME -a -w $HOME ];echo $?`

Příklad 9: Příklad na použití if-then-else. Příkaz otestuje, zda zadaný adresář je skutečně adresář.

```
$ if test -d tmp
> then
>   echo "Ano, jsem adresar."
> else
>   echo "Ne, nejsem adresar."
> fi
```

Příklad 10: Příklad na použití if-then-elif-else.

```
$ if false
> then
>   echo "Prvni klausule je pravdiva."
> elif true
>   echo "Druha klausule je pravdiva."
> else
>   echo "Ani prvni ani druha klausule neni pravdiva."
> fi
```

Příklad 11: následující skript zkontroluje, zda první parametr, se kterým byl script spuštěn začínal na -a , -A , -l nebo -L . Pokud nezačíná, vypíše : nepovolena hodnota prvniho parametru .

```
#!/bin/bash
case $1 in
     -a*|-A*) echo "prvni parametr je typu a";;
     -l*|-L*) echo "prvni parametr je typu l";;
     *) echo "nepovolena hodnota prvniho parametru";;
```

Příkaz for

Syntaxe příkazu je následující:

```
for proměnná in seznam_slov
 do příkazy
done
```

Sémantika: Slovo je řetězec znaků neobsahující mezeru. Seznam_slov je posloupnost slov oddělených mezerami. 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.

Příkaz while

Syntaxe příkazu je následující:

```
while příkaz
 do příkazy
done
```

Sémantika: Při každém průběhu cyklem se nejdříve provede příkaz uvedený za while . Pokud je návratová hodnota 0 (true), provedou se příkazy, uvedené mezi klíčovými slovy do a done . Pokud je návratová hodnota různá od nuly (false), vykonávání příkazu while je ukončeno.

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.

Ukončení scriptu

Script se ukončuje příkazem **exit**. Příkaz **exit** má syntaxi:

```
exit n
```

kde *n* definuje status ukončení scriptu.

Ošetření signálu

Násilné ukončení scriptu signálem lze ošetřit příkazem **trap**. Příkaz **trap** má následující syntaxi:

trap 'příkazy' seznam_signálů
Příkazy jsou odděleny středníkem. Signály jsou v seznamu_signálů označeny svými čísly a jsou odděleny mezerami.

Sémantika příkazu trap je následující:
Po příchodu některého signálu ze seznamu_signálů jsou vykonány příkazy. Pokud řetězec v uvozovkách je prázdný, je příchod signálu ignorován. Pokud má skript po příchodu některého ze signálů ze seznamu_signálů skončit, musí být posledním příkazem v uvozovkách příkaz exit .

Příklady - Bourne shell (bash)

Příklad 12: Ve svém domovském adresáři vytvořte podadresář Superhrdinove a v něm soubory:

```
Aqua-man   Spider-man  Barman  Super-man  Woman-cat  Octopus
```

Vytvořte pro všechny soubory záložní kopie tak, že se vytvoří nový soubor se stejným jménem a navíc koncovkou .zaloha

Řešení

```
cd ~ ; mkdir Superhrdinove
touch Agua-man Spider-man Barman Super-man Woman-cat Octopus

for foo in Agua-man Spider-man Barman Super-man Woman-cat
Octopus
> do
>   cp $foo $foo.zaloha
> done
```

Příklad 13: Ve všech souborech z předchozího příkladu zaměňte pomlčky (-) za podtržítka (_)

Řešení

```
for foo in *
> do
>   mv $foo `echo $foo | sed 's/-/_/g'`
> done
```

Příklad 14: Ve svém domovském adresáři vytvořte 100 podadresářů s názvy `test1` – `test100`

Řešení

```
DIRCOUNT=0
while [ $DIRCOUNT -ne 100 ]
> do
>   DIRCOUNT=`expr $DIRCOUNT + 1`
>   mkdir test -$DIRCOUNT
> done
```

Příklad 15: Ve svém domovském adresáři vytvořte podadresář `tmp` a v něm podadresář `tmp`, tento postup ještě několikrát zopakujte – získáte tak sekvenci vnořených adresářů `tmp`. Napište příkaz, který způsobí přepnutí aktuálního adresáře na nejvíce vnořený adresář `tmp`.

Řešení

```
cd ~
while cd tmp
> do
>   pwd
> done
```

Příklad 16: Pomocí příkazů while a shift napište script, který opíše argumenty příkazového řádku.

Řešení

```
#!/bin/bash
#opis parametru prikazoveho radku
while [ $# -gt 0 ]
do
     echo -n "$1 "
     shift
done
echo
```

Příklad 17: 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. Scriptu lze zadat požadavek na zpracování více adresářů najednou. Script řešící tuto úlohu byl již napsán v C shellu.

Řešení

```
#!/bin/bash
#nums : script vypise pocet souboru a adresaru
if [ $# -eq 0 ]
     then dir="."
     else dir=$1
fi
```

```
while :
do
     if [ ! -d $dir ]
          then echo "$0 : $dir neni adresar"
     else
          echo "$dir :"
          fcount=0
          dcount=0
          for file in $dir/*
          do
               if [ -f $file ]
                    then fcount=`expr $fcount + 1`
               elif [ -d $file ]
                    then dcount=`expr $dcount + 1`
               fi
          done
          echo $dcount adresaru $fcount obycejnych souboru
     fi
     if [ $# -le 1 ]
          then break
     else
          shift
          dir=$1
     fi
done
```

Příklad 18: V Bourne shellu napište script , který vypíše všechny soubory z adresáře /etc, které začínají řetězcem host .

Řešení

```
#!/bin/bash
for i in  /etc/host*
do
     echo  \
"************************************************
          Soubor $i
************************************************"
     cat $i
done
```

Příklad 19: Script z příkladu 3 modifikujte tak, aby vypsal soubory, které uvedete v příkazovém řádku.

Řešení

Řádek `for` změňte na `for i in $*`

Příklad 20: V Bourne shellu napište script, který bude načítat parametry příkazové řádky a bude je předkládat uživateli ke schválení. Pokud uživatel odpoví y (yes), skript parametr opíše na standardní výstup. Pokud odpoví q (quit), skript bude ukončen.

Řešení

```
#/!/bin/bash
#vezmi: Skript nacita prikazove radky z klavesnice a predklada
#je uzivateli ke schvaleni. Pokud uzivatel odpovi y, skript
# parametr opise. Pokud odpovi q, je skript ukoncen.
#Syntaxe  vezmi parametry

PATH=/bin:/usr/bin


#cyklus pres vsechny parametry prikazoveho radku
for i
do
     echo -n "$i?" >/dev/tty
     read response
     case $response in
     y*) echo $i;;
     q*) break
     esac
done </dev/tty
```