Základy programování v shellu Bash

Tomáš Kühr

KATEDRA INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI

Bash

- interpret příkazů v unixových systémech
- název je akronym pro Bourne Again SHell
- původní autor Brian Fox
- vytvořen 1989 v rámci GNU projektu
- plnohodnotný programovací jazyk
- doplněný konstrukcemi pro práci se základními nástroji (GNU Core Utilities) a dalšími programy
- existuje mnoho alternativ: Bourne shell, C shell, Korn shell, Z shell
- spouští se při přihlášení uživatele (lokálním i vzdáleném)
- přihlašovací (login) vs. nepřihlašovací (non-login) shell
- interaktivní vs. neinteraktivní shell

Vytváření skriptů

- skript je běžný textový soubor
- s nastaveným právem pro spouštění (klasicky pomocí chmod)
- na první řádku skriptu by měla být tzv. hlavička
  `#!/bin/bash`
- skript pak lze spouštět jako libovolný program
  `./skript ... parametry ...`
- znak # také uvozuje komentáře
- ignorováno je vše od # do konce řádku

Proměnné

- identifikátor proměnné může obsahovat jen písmena, číslice a znak podtržítko
- Bash pracuje s hodnotami proměnných jako s řetězci
- proměnné mohou být deklarované jako pouze pro čtení (pomocí readonly)

Přiřazení

- proměnnou není třeba předem definovat
- při přiřazení je proměnná vytvořena a je jí nastavena hodnota
- příklad:
  ```
  NAME="Tomas Kuhr"
  ```

Dereference

- slouží k získání hodnoty proměnné
- obvykle bývá součástí složitějšího příkazu
- příklad:
  ```
  echo $NAME
  ```

Proměnné

Příkaz read

- slouží k zadání hodnoty proměnné uživatelem
- příklad:

```
#!/bin/bash
echo "What is your name?"
read PERSON
echo "Hello, $PERSON"
```

Vnitřní proměnné

- v shellu máme přístup i k mnoha vnitřním proměnným operačního systému
- například: HISTFILE, HISTFILESIZE, HOME, HOSTNAME, PATH, UID, USER, PWD, RANDOM, SHLVL, ...

Speciální proměnné

- jejich identifikátor nesplňuje výše uvedená omezení
- nemohou být vytvářeny a měněny běžným způsobem

$$

- proměnná obsahující PID shellu

$!

- PID posledního příkazu, který byl spuštěn na pozadí

$?

- návratová hodnota posledního dokončeného procesu

$0

- jméno souboru skriptu

$#

- počet argumentů uvedených při spuštění skriptu

$*n*

- n-tý argument uvedený při spuštění skriptu

Program test

- umožňuje klasifikaci souborů, porovnávání řetězců i celých čísel a porovnávání souborů dle jejich stáří
- obvykle místo `test` *výraz* píšeme `[` *výraz* `]` (pozor na mezery)

Možnosti zadaní výrazu

- test, zda je řetězec str nenulový – `[ str ]`
  příklad: `[ $PATH ]; echo $?`
- test, zda je řetězec str nulový – `[ -z str ]`
  příklad: `[ -z $PATH ]; echo $?`
- řetězce str1 a str2 jsou shodné – `[ str1 = str2 ]`
  příklad: `[ $USER = kuhrtoma ]; echo $?`
- řetězce str1 a str2 jsou různé – `[ str1 != str2 ]`
  příklad: `[ $USER != kuhrtoma ]; echo $?`

Program test

Možnosti zadání výrazu

- čísla n1 a n2 jsou shodná – `[ n1 -eq n2 ]`
  příklad: `[ 1 -eq 01 ]; echo $?`
- čísla n1 a n2 jsou různá – `[ n1 -ne n2 ]`
  příklad: `[ 1 -ne 01 ]; echo $?`
- číslo n1 je menší nebo rovno n2 – `[ n1 -le n2 ]`
  příklad: `[ 1 -le 01 ]; echo $?`
- číslo n1 je menší než n2 – `[ n1 -lt n2 ]`
  příklad: `[ 1 -lt 01 ]; echo $?`
- číslo n1 je větší nebo rovno n2 – `[ n1 -ge n2 ]`
  příklad: `a=2;[ $a -ge 1 ]; echo $?`
- číslo n1 je větší než n2 – `[ n1 -gt n2 ]`
  příklad: `a=2;b=1;[ $a -gt $b ]; echo $?`

Program test

Možnosti zadání výrazu

- test, zda soubor file existuje – `[ -e file ]`
- test, zda je soubor file adresář – `[ -d file ]`
- test, zda je soubor file obyčejný soubor – `[ -f file ]`
- test, zda je soubor file symbolický odkaz – `[ -L file ]`
- test, zda je soubor file možné číst – `[ -r file ]`
- test, zda je do souboru file možné zapisovat – `[ -w file ]`
- test, zda je soubor file spustitelný – `[ -x file ]`
- test, zda je soubor file neprázdný – `[ -s file ]`
- test, zda je soubor f1 novější než f2 – `[ f1 -nt f2 ]`
- test, zda je soubor f1 starší než f2 – `[ f1 -ot f2 ]`
- příklady:

```
[ -f skript.sh ]; echo $?
[ pokus.sh -ot skript.sh ]; echo $?
```

Program test

Logické operace

- slouží po konstrukci složitějších podmínek přímo v programu test
- logická spojka NEBO (-o ve výrazu)
- logická spojka A (-a ve výrazu)
- negace (! ve výrazu)
- příklady:

```
[ $a -lt 10 -o 20 -le $a ]; echo $?
[ $a -lt 10 -a 20 -le $b ]; echo $?
[ ! $a -lt 10 -a 20 -le $b ]; echo $?
```

Oddělovače příkazů

Základní oddělovač

- pokud zadáváme více příkazů za sebou, oddělíme je středníkem
- hodí se jak ve skriptech, tak při složitějších příkazech zadávaných do promtu
- příklad: `sleep 3; echo "*"`

Podmíněné vykonání příkazu

- konstrukce umožňující provést příkaz na základě úspěšného vykonání předchozího příkazu
- příkaz se vykoná, jen když se předchozí neskončil chybou (konstrukce &&)
- příkaz se vykoná, jen když předchozí skončil chybou (konstrukce ||)
- příklady:

```
cd bla && echo "*"
cd . && echo "*"
cd bla || echo "*"
cd . || echo "*"
```

Větvení

Konstrukce if

- základní možnost větvení programu
- možné tvary konstrukce if:
  `if` *podmínka*`; then` *příkazy*`; fi`
  `if` *podmínka*`; then` *příkazy*`; else` *příkazy*`; fi`
  `if` *podmínka*`; then` *příkazy*`; elif` *podmínka*`; then` *příkazy*`; ...`
  `else` *příkazy*`; fi`
- v podmínce odpovídá hodnota nula (korektní konec programu) pravdě, nenulová hodnota (chyba při vykonávání programu) pak nepravdě
- část `elif` *podmínka*`; then` *příkazy*; se může vyskytovat vícekrát (s různými podmínkami a příkazy)
- příklad:

```
if [ ! -e $1 ]; then echo "Neexistuje";
elif [ ! -s $1 ]; then echo "Je prazdny";
else echo "Existuje a je neprazdny";
fi
```

Větvení

Konstrukce case

- zápis odpovídá následujícímu tvaru:
  `case` *výraz* `in` *vzory*`)` *příkazy*`;;` `...` `esac`
- část *vzory*`)` *příkazy*`;;` se může opakovat (s různými vzory a příkazy)
- vyhodnotí se *výraz* a porovnává se postupně s jednotlivými vzory
- pokud se nalezne odpovídající větev, provedou se dané příkazy a pokračuje se za konstrukcí case
- větev může odpovídat více vzorům (oddělovač |)
- vzory mohou obsahovat metaznaky *, ?, [,] (pozor nemají význam jako v regulárních výrazech, ale jako při expanzi v shellu)
- příklad:

```
case $1 in
   [0-9]*|Ahoj) echo "Zacina cislici nebo je ahoj";;
   [a-z]*[a-z]) echo "Zacina a konci malym pismenem";;
   -?@) echo "Pomlka Neco Zavinac";;
   *) echo "Neco jineho";;
esac
```

Cykly

Konstrukce for

- pravděpodobně nejpoužívanější konstrukce cyklu v Bashi
- obecný tvar zápisu:
  `for` *proměnná* `in` *seznam*`; do` *příkazy*`; done`
- seznam může obsahovat čísla i řetězce
- může být zadán výčtem, jako číselná posloupnost nebo pomocí expanze jmen souborů
- další možnosti je využít jako seznam výsledek jiného příkazu (viz příště)
- příklad:

```
for i in 1 2 3 4 5;
do
   echo $i;
done

for s in Aa Bb Cc Dd Ee Ff;
do
   echo $s;
done
```

Cykly

– příklady:

```
for j in {1..5}
do
   echo $j;
done

for j in {1..10..2}
do
   echo $j;
done

for name in *;
do
   echo $name;
done
```

Cykly

Konstrukce while

- obecný tvar zápisu:
  `while` *podmínka*`; do` *příkazy*`; done`
- pokud je podmínka cyklu pravdivá, provede se další průchod cyklem a opět se přejde k testu podmínky
- lze vytvořit i nekonečný cyklus (viz příkazy přerušení)
- příklad:

```
while read word;
do
   echo "$word $word";
done
```

Cykly

Konstrukce until

- obecný tvar zápisu:
  `until` *podmínka*`; do` *příkazy*`; done`
- pokud je podmínka cyklu nepravdivá, provede se další průchod cyklem a opět se přejde k testu podmínky
- příklad:

```
read n;
until [ 1 -le $n ] && [ $n -le 10 ];
do
   echo "Zadejte cislo od 1 do 10: ";
   read n;
done
```

Cykly

Příkazy přerušení

- mohou ovlivnit cyklus z libovolného místa uvnitř cyklu
- typicky se používají ve složitějších cyklech, společně s větvením programu
- možnost okamžitého ukončení cyklu (příkaz break)
- možnost okamžitého přechodu k dalšímu průchodu cyklem (příkaz continue)
- příklad:

```
for name in *;
do
   if [ -d $name ]; then
      break;
   fi
done

ls -l $name
```

Bodované úlohy

1. Vytvořte skript, který pro 3 zadaná čísla vypíše jejich minimum. (1 body)
   Příklad použití:
   ```
   [kuhrtoma@phoenix xunix]$./min.sh 2 1 3
   1
   ```
2. Vytvořte skript, který vytvoří adresáře A až F a v každém z nich (stačí prázdné) soubory 1.log až 99.log. (2 body)