Základy programování v shellu Bash

Tomáš Kühr

KATEDRA INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI

Aritmetika

- Bash podporuje základní celočíselnou aritmetiku
- existuje několik možností, jak aritmetické operace do programu zapsat

Pomocí expr

- expr je program pro vyhodnocování aritmetických výrazů
- podporován i ve starších shellech (např. Bourne shell)
- příklad:

```
a=2
s=`expr $a + 3`
echo $s
```

Pomocí let

- let je příkaz modernějších shellů (např. Bash, Korn shell)
- není nutné používat dereference
- příklady:

```
let s=$a+$b
echo $s
let s=a+b
echo $s
```

Aritmetika

Pomocí závorek

- nejspíše nejpoužívanější možnost v Bashi
- není tolik háklivé na použití mezer v zápisu
- uvnitř závorek se nepoužívají dereference
- příklady:

```
s=$((a+b))
echo $s
((s=a+b))
echo $s
(( s = a + b ))
echo $s
```

Výpočty s desetinnými čísly

- nejsou podporovány žádnou z výše uvedených alternativ
- lze realizovat například pomocí programu bc

Aritmetika – operátory

Aritmetické operátory

- s obvyklým významem: +, -, *, /, % (zbytek po dělení), ++ (inkrementace), -- (dekrementace), ** (umocnění)

Podmínkové operátory

- s obvyklým významem: <, >, <=, >=, == (rovnost), != (nerovnost)
- příklad:

```
if (( a > b )); then
   echo "a > b"
fi
```

Logické operátory

- s obvyklým významem: && (a zároveň), || (nebo), ! (negace)
- příklad:

```
if (( (1 < a) && (a <= 10) )); then
   echo "ano";
fi
```

Pole – definice

Definicí jednotlivých prvků

- klasicky pomocí operátoru indexu a operátoru přiřazení
- příklad:

```
NAME[0]="Adam"
NAME[1]="Barbora"
NAME[2]="Cyril"
NAME[3]="Dana"
NAME[4]="Eva"
```

Definicí celého pole

- pomocí operátoru přiřazení a kulatých závorek vytvářejících pole
- příklad:

```
NAME=("Adam" "Barbora" "Cyril" "Dana" "Eva")
```

Přístup k poli

Přístup k jednotlivým prvkům

- klasicky pomocí operátoru indexu
- příklad:

```
echo "Prvni jmeno: ${NAME[0]}"
echo "Posledni jmeno: ${NAME[4]}"
```

Zpracování celého pole

- obecně: `${`*jméno_pole*`[*]}` nebo `${`*jméno_pole*`[@]}`
- příklady:

```
echo "Jmena: ${NAME[*]}"
echo "Jmena: ${NAME[@]}"
```

Délka pole

- obecně: `${#`*jméno_pole*`[*]}` nebo `${#`*jméno_pole*`[@]}`
- příklady:

```
echo "Pocet: ${#NAME[*]}"
echo "Pocet: ${#NAME[@]}"
```

Cykly přes pole

Cyklus přes prvky pole

- obecně: za in ve for cyklu uvedeme ${*jméno_pole*[@]}
- příklad:

```
for n in "${NAME[@]}"
do
   echo $n
done
```

Cyklus přes indexy v poli

- obecně: za in ve for cyklu uvedeme ${!*jméno_pole*[@]}
- příklad:

```
for i in "${!NAME[@]}"
do
   echo "$((i+1)). jmeno: ${NAME[$i]}"
done
```

Funkce – vytvoření

- definice funkce obecně:

```
function jméno_funkce (){
    příkazy těla funkce
}
```

- alternativně také:

```
jméno_funkce (){
    příkazy těla funkce
}
```

- mezi výše uvedenými způsoby definice funkce není při dalším použití žádný rozdíl
- funkce musí být definována dříve, než je ve skriptu použita
- příklady:

```
function pozdrav () {
   echo "Ahoj svete"
}
pozdrav2 () {
   echo "Nazdar svete"
}
```

Funkce – použití

- na rozdíl od jiných jazyků se v Bashi při volání funkce nepoužívají kulaté závorky
- příklad:

```
pozdrav
```

Parametry funkce

- při definici funkce se nikam neuvádějí
- při volání funkce se předávané hodnoty uvedou za jménem funkce
- v těle funkce k nim přistupujeme pomocí proměnných $1, $2, ...
- příklad:

```
pozdrav_me() {
   echo "Ahoj, $1!"
}
pozdrav_me Pepo
```

Funkce – použití

Návratová hodnota funkce

- nastavuje se pomocí příkazu `return`, který zároveň ukončí výpočet funkce
- v Bashi se nicméně používá vesměs pro oznámení, zda výpočet proběhl úspěšně (návratová hodnota 0) nebo ne (kód chyby v návratové hodnotě)
- vypočtené výsledky se obvykle ukládají do nějaké proměnné
- příklad:

```
soucet(){
   suma=0
   for i in "$@" # cyklus pres parametry
   do
      ((suma=suma+i))
   done
   return 0
   echo "Nevypise se"
}
soucet 1 2 3 4 5 6
echo $? $suma
```

Rozsah platnosti proměnných

- pokud není uvedeno jinak, všechny proměnné jsou globální (v rámci daného shellu)
- lokální proměnnou lze vytvořit uvnitř funkce pomocí klíčového slova `local`
- případná globální proměnná se stejným jménem je pak uvnitř funkce překryta touto lokální proměnnou
- příklad:

```
cislo=2
test() {
   echo 1 $cislo
   local cislo=3
   echo 2 $cislo
}
echo 3 $cislo
test
echo 4 $cislo
```

Kvotování

- při programování v Bashi mají některé znaky (tzv. metaznaky) speciální význam
  `* ? [ ] ' " ` \ $; & ( ) | ^ < >`
- pokud chceme potlačit tento speciální význam, můžeme před daným znakem použít `\`
- příklad: `echo \* \? \[ \] \' \" \` \\ \$ \; \& \(\) \| \^ \< \>`
- podobným způsobem lze zapsat také některé bílé znaky: `\t \n \v`
- pokud potřebujeme potlačit speciální význam více metaznaků (včetně mezer), můžeme použít jednoduché (') nebo dvojité (") uvozovky
- dvojité uvozovky nepotlačí význam metaznaků: `" ` \ $`
- příklad: `echo "* ? [ ] ' ; & ( ) | ^ < >"`
- jednoduché uvozovky nepotlačí pouze význam metaznaku `'`
- příklad: `echo '* ? [ ] " ` \ $; & ( ) | ^ < >'`

Substituce příkazů

- Bash disponuje také konstrukcí pro použití výsledku jednoho příkazu uvnitř jiného příkazu
- vnitřní příkaz je ohraničen znaky `
- příklady:

```
DATE=`date`
echo "Dnes je $DATE."

USERS=`who | wc -l`
echo "Pocet prave prihlasenych uzivatelu: $USERS"

UP=`date ; uptime`
echo "Uptime: $UP"
```

Substituce – příklad

Soubor sub1.sh

```
#!/bin/bash
count=0
for i in `find $1*/*.txt 2> /dev/null`
do
   ((count++))
done
echo $count
```

Soubor sub2.sh

```
#!/bin/bash
for n in a b c d e
do
   count=`./sub1.sh $n`
   echo $n $count
done
```

Bodované úlohy

1. Vytvořte skript, který vypíše geometrickou posloupnost se zadanými parametry (první člen, kvocient, počet vypisovaných členů). (1 bod)
   Příklad použití:
   ```
   ./geom.sh 1 2 5
   1 2 4 8 16
   ```
2. V operačních systémech bývá zvykem, že se u logů provádí takzvané rotace. Soubor xyz se přejmenuje na xyz.0, xyz.0 na xyz.1 atd. až do nějaké horní hranice. Soubory končící číslem nad touto hranicí se vymažou. Napište program, který dostane dva parametry – jméno souboru a maximální počet uložených kopií a provede rotaci. (2 body)

Dodatečné úlohy k zápočtu

1. Vytvořte skript, který vypíše všechna prvočísla až po zadanou horní mez. (1 bod)
2. Vytvořte skript, který vypíše informace o všech souborech v daném adresáři (obdobně jako ls), jejichž velikost je větší než parametrem zadaná dolní hranice. (2 body)
3. Vytvořte skript, který vypíše všechny uživatele, kterým v daný okamžik běží více než parametrem zadaný počet procesů. (2 body)
4. Vytvořte skript, který pro počet dní zadaný jako parametr vypíše všechny uživatele, kteří byli (nebo stále jsou) v systému přihlášeni od aktuálního času po zadaný počet dní do minulosti. Pro informace o přihlášení uživatelů použijte program last. (4 body)