Skriptování v Linuxu (bash) a Windows (cmd)

Příkazové interprety (v Linuxu shelly) obsluhující příkazový řádek mají schopnost interpretovat (provádět řádek po řádku, resp. blok po bloku) soubory s příkazy, běžně nazývané skripty nebo (ve Windows) dávky. Výbava interpretů se snaží pokrýt postupy dostupné v běžných (imperativních procedurálních) programovacích jazycích, konkrétně: proměnné, vyhodnocování znakových a (celočíselných) aritmetických výrazů, podmíněné příkazy, cykly a volání procedur; v nabídce je i řada specializovaných či historických postupů, někdy působících spíše jako triky. Požadavky nad rámec interní výbavy interpretů (např. reálná aritmetika) lze samozřejmě řešit voláním externích příkazů. Následující ukázky byly ověřeny v linuxovském shellu bash 4.2.24, MinGW bash 3.1.17 a příkazovém interpretu cmd ve Windows 7; jde o nikoliv kompletní přehled.

Obecné postřehy

Identifikace skriptu. Skripty v Linuxu mohou mít libovolné jméno a mají-li být samostatně spustitelné, musí obsahovat příslušné právo (chmod u+x file); alternativně je lze spouštět v běžícím shellu (interním) příkazem . nebo source:

(Linux) ./script ; . script ; source script

Skripty ve Windows musí mít příponu bat a v okně příkazového interpretu se spouští zadáním svého jména; uvedení přípony je nezbytné, pokud je v témže adresáři stejnojmenný soubor s příponou com nebo exe. Možné je i uvedení skriptu jako argumentu nového interpretu:

(Windows) script & script.bat & cmd /c script

První řádek. Skripty typicky začínají specifickým řádkem: skripty pro bash obsahují tzv. shebang (sharp-bang, #!) řádek s pokynem pro interpretaci pomocí bash, dávky pro cmd vypínají defaultní ozvěnu (opsání) příkazů na obrazovku příkazem echo off platným pro zbytek dávky a zavináčem @ pro vypnutí ozvěny prvního příkazu.

```
!#/bin/bash                    @echo off
```

Lokalita. Nastavení proměnných a řada jiných změn se může a nemusí projevit vně skriptu. V Linuxu ... Ve Windows se změny vně implicitně projevují; potlačit to lze časným uvedením příkazu setlocal ve skriptu nebo spuštěním skriptu jako argumentu nového interpretu (cmd /c). Ve skriptech Windows bývá někdy vhodné nastavit možnost odložit získání hodnoty proměnné na okamžik jejího použití, což je jindy než v moment načtení a interpretace řádku nebo bloku (default); děje se pomocí příkazu setlocal enabledelayedexpansion nebo spuštěním skriptu v novém interpretu s parametrem cmd /v:on.

Syntaxe. V příkazech i jménech proměnných bash rozlišuje velká písmena od malých, cmd nikoliv. Zarovnávání nebo rozmáchlé mezerování nemá jiný význam než zvýšení čitelnosti. Na jednom řádku lze sdružovat více příkazů, oddělovačem je (bash) ; nebo (cmd) &. Ve Windows lze vytvářet bloky příkazů jejich uzavřením do kulatých závorek ( ), v Linuxu jsou bloky v podmíněných nebo cyklických konstrukcích možné bez zvláštní syntaxe. Řádkové komentáře:

```
# comment                      rem comment
                               :comment
```

Druhá Windows varianta je prvoplánově návěštím řádku dostupného pomocí skoků goto nebo call.

Proměnné a (znakové a aritmetické) výrazy

přiřazení	var=value		set var=value	
výpis	echo $var	při zřetelném oddělení	echo %var%	vyhodnotí se při načtení příkazu (bloku)
	echo ${var}	v těsném kontaktu	echo !var!	vyhodnotí se v momentu použití
znakové výrazy	ch=3.14 ; ch=${ch}159		set ch=3.14& set ch=!ch!159	vrátí 3.14159
integer výrazy	n=1; n=$((n+1))		set n=1 & set /a n=(n+2*3)/4	vrátí 1 (netřeba %n%)
	n=1; let n=n+1; ((n=n+1))		pozor: set n=010	vrátí 010 (znaky)
operátory	+-, */%, **, ==, !=, >= ad.		set /a n=010	vrátí 8 (osmičková soustava)
int proměnné	declare -i n=1; n=n+1		set /a n=0x10	vrátí 16, set /a n=x10 vrátí 0
výstup příkazu	var=`cmd`; var=$(cmd)		for /f "delims=" %s in ('cmd') do set var=%s	v dávce %%s
real výrazy	var=`echo | awk "{print 1/3}"`;	var=`echo "scale=10; 1/3" | bc`	Windows: v echu bez uvozovek	
zrušení	unset var		set var=	
výpis všech	declare; env		set	
interní prom.	$RANDOM, $USER		%date%, %random%, %time%, %username%	

bc je vhodný pro (integer i real) výrazy s neomezenou přesností, podporuje však jen několik matematických funkcí s netypickými jmény; pro výrazy s funkcemi se zdá vhodnější např. gmtmath či gnuplot:

```
pi=`echo "a(1)*4" | bc -l`                  bc -l pro připojení matematické knihovny a nastavení scale=20
pi=`gmtmath -Q 1 ATAN 4 MUL =`              reverzní polská (postfixová) notace
pi=`echo "print atan(1)*4" | gnuplot 2>&1`   2>&1 pro přesměrování chybového výstupu na standardní
```

Podmíněné příkazy

if ...; then ...; elif ...; then ...; else ...; fi	if [not] ... (...) else (...) operátory equ, neq, lss, leq, gtr, geq
if (($n>=0); then echo $n positive; fi	if !n! geq 0 echo !n! positive == pro znakový test
testy ((n>=0)) ==, !=, <, <=, >, >=, !	set n=+1 & if !n! equ 1 ... & if !n!==1 ... vrátí ano a ne
[$n -ge 0] -eq, -ne, -lt, -le, -gt, -ge, !	set n=1. & if !n! equ 1 ... vrátí ne (přechod na znakový test)
existence - proměnné if [$n]; then echo $n; fi	if defined n echo !n!
- souboru if [-e $file]; then echo $file exists; fi	if exist !file! (echo !file! exists) else echo !file! not found
- adresáře if [! -d $dir]; then mkdir $dir; fi	if not exist !dir! (mkdir !dir!) else echo !dir! exists

Cyklus s řídicí proměnnou

```
for (( n=0; n<=$nmax; n++)); do
  echo $n # blok
done
```

```
for /L %%n in (0,1,%nmax%) do (
  echo %%n
)
```

Cyklus s podmínkou

```
while ...; do ...; done
```

Cyklus s výčtem

```
for var in ...; do ...; done
```

```
for %%n in (1 2 3) do (blok)
```

Cyklus pro řádky souboru

```
for /F "tokens=1-3" %%i in (files) do (echo %i %j %k)
```

Příkaz skoku (bez návratu)

Volání externího skriptu

Volání interní procedury (s návratem)

```
call :proc parameters
```

Deklarace interní procedury

```
:proc
  echo %1 %2  volání až 9 parametrů, více s pomocí shift
goto :eof
```

Přesměrování vstupu a výstupu

kanály

jména speciálních (virtuálních) souborů

30. 3. 2014