Robert Meolic
meolic@uni-mb.si

Skriptni jezik lupine Windows command

interno gradivo za predmet VSO, 2006/07

1. LITERATURA

Pri sestavljanju gradiva o skriptnem jeziku lupine Windows command sem uporabljal internet in naslednji dve knjigi (prva je res dobra):

- William R. Stanek: Microsoft Windows Command-Line Administrator's Pocket Consultant, Microsoft Press, 2004
- Jerry Lee Ford, JR: Microsoft Windows Shell Script Programming for the absolute beginner, Premier Press, 2004

2. UVOD

Skritpni jezik lupine Windows command je v operacijskih sistemih Windows 2000, Windows Server 2003 in Windows XP dopolnjen glede na to, kar smo imeli na voljo npr. na Windows 98 in Windows NT. Opis v nadaljevanju se nanaša na novo različico. V starih sistemih ne bodo delovali vsi opisani ukazi. Različica skriptnega jezika je zapisana v spremenljivki `CMDEXTVERSION`. V novejših sistemih ima ta spremenljivka vrednost 2.

Naša prva skripta v lupini Windows command bo imela samo dve vrstici, ki ju vpišemo v datoteko `hello.bat`.

```
@ECHO OFF
ECHO Hello World
```

Pri izvajanju skript v lupini Windows command je privzeto, da se vsak ukaz najprej izpiše in šele potem izvede. Če izpisa ukaza ne želimo, moramo pred njega dodati znak `@`. Prva vrstica naše skripte določa, da naj se izključi izpisovanje vseh nadaljnjih ukazov in naj se izpisujejo samo rezultati. Z znakom `@` pred ukazom `ECHO` smo dosegli, da se prvi ukaz ne bo izpisal. Druga vrstica programa izpiše pozdrav. Ukaze lahko pišemo z malimi ali pa z velikimi črkami, bolj običajen pa je zapis z velikimi črkami. Če želimo izpisati prazno vrstico, takoj za ukazom `ECHO` naredimo piko.

Kot drug primer vzemimo program, ki prekopira vse datoteke v trenutnem imeniku v imenik `Trash`, nato pa ta imenik skupaj z njegovo vsebino zbriše.

```
MKDIR Trash
COPY * Trash
RMDIR Trash /s /q
```

3. SPREMENLJIVKE IN ARGUMENTI

V skriptah lahko uporabljamo **spremenljivke**. Vrednost jim priredimo z ukazom SET. Spremenljivk pred tem ni potrebno deklarirati. Imena spremenljivk so lahko poljubna in lahko vsebujejo tudi presledke in posebne znake. Vrednost spremenljivke dobimo tako, da njeno ime zapišemo med dva znaka za procent. Spremenljivko odstranimo tako, da za enačaj ne napišemo ničesar.

```
@ECHO OFF
SET STR=Hello World
SET STR S PRESLEDKI = Ali res razumes spremenljivke?
ECHO %STR%
ECHO %STR S PRESLEDKI %
SET STR=
SET STR S PRESLEDKI =
```

Lupina Windows command vse spremenljivke obravnava kot nize znakov. Zato zna tudi nekaj operacij nad podnizi. Zelo uporabni sta naslednji operaciji:

- %Variable:~Position,Length% - vrne del vrednosti spremenljivke Variable, vrnjeni podniz se začne od mesta Position in ima dolžino Length
- %Variable:Replacement=Original% - vrne vrednost spremenljivke Variable, vendar pa v njej vse podnize Replacement zamenja z nizom Original

Z naslednjim stavkom preverimo, ali se vrednost spremenljivke %ODGOVOR% začne s črko Y:

```
IF /I "%ODGOVOR:~,1%" EQU "y"
```

Če želimo s spremenljivkami računati, v stavku SET dodamo kretnico /A. V tem primeru se presledki in posebni zanki obravnavajo drugače ter jih ne smemo uporabljati v imenih spremenljivk, imena spremenljivk pa pišemo brez znakov za procent.

```
@ECHO OFF
SET A=5
SET /A B = 2*A-1
SET /A B = B+1
ECHO A=%A%, B=%B%
```

Podprte so aritmetične operacije seštevanje (+), odštevanje (-), množenje (*), celoštevilčno deljenje (/) in ostanek pri deljenju (%).

Vrednost spremenljivke lahko uporabimo tako, da tvorimo novo ime, novo vrednost spremenljivke ali pa celo sestavimo kakšen ukaz.

```
@ECHO OFF
SET A=ENA
SET D=DIR
SET E=*.BAT
SET %A%DVA=TRI%A%
ECHO %ENADVA%
MK%D% Test
%D% %E%
```

Ukaz SET omogoča tudi interakcijo z uporabnikom. V tem primeru uporabimo kretnico /P. Na primer, naslednji ukaz izpiše na ekran „Vnesi število: „ in potem počaka, da uporabnik nekaj vnese. Vnešeno besedilo se shrani v spremenljivko STEVILO.

```
SET /P STEVILO="Vnesi število: "
```

Skriptom lahko ob zagonu podamo **argumente**, ki se obravnavajo pozicijsko:

- %0 je ime skripte,
- %1 je prvi argument,
- %2 je drugi argument itd.

Če argument vsebuje presledke, ga podamo v narekovajih.

4. KRMILNI STAVKI

Na voljo imamo tri **pogojne stavke**, katerih splošna oblika je naslednja:

```
IF pogoj (statements) ELSE (statements)
IF NOT pogoj (statements) ELSE (statements)
IF DEFINED variable (statements)
```

Stavek ELSE ni obvezen, tudi oklepaje lahko v nekaterih primerih izpustimo. Če želimo pogojni stavek napisati čez več vrst, moramo vrstico deliti takoj za okroglim oklepajem, tako da dobimo naslednjo obliko:

```
IF pogoj (
  statements
) ELSE (
  statements
)
```

Pogoj, ki ga uporabimo v stavku IF, je vedno primerjava nizov znakov. Uporabimo lahko naslednje operatorje:

- s1 == s2 (ali sta niza enaka?)
- s1 EQU s2 (ali sta niza enaka?)
- s1 NEQ s2 (ali sta niza različna?)
- s1 LSS s2 (ali je prvi niz manjši od drugega?)
- s1 LEQ s2 (ali je prvi niz manjši ali enak od drugega?)
- s1 GTR s2 (ali je prvi niz večji od drugega?)
- s1 GEQ s2 (ali je prvi niz večji ali enak od drugega?)

Pri primerjavi nizov znakov se male in velike črke razlikujejo. Če ne želimo razlikovanja, dodamo za ukazom IF kretnico /I.

Primer uporabe pogojnega stavka je tukaj:

```
@ECHO OFF
IF /I "%1" == "ena" (
  ECHO prvi parameter
  ECHO je ena
) ELSE (
  ECHO prvi parameter ni ena
)
```

Zanke v skripti lahko tvorimo s stavkom FOR. Na voljo je več oblik tega ukaza.

```
FOR %%variable IN (fileSet) DO (statements)
FOR /R path %%variable IN (fileSet) DO (statements)
FOR /D %%variable IN (directorySet) DO (statements)
FOR /R path /D %%variable IN (directorySet) DO (statements)
FOR /L %%variable IN (init,step,end) DO (statements)
FOR /F "options" %%variable IN (fileSet) DO (statements)
```

Pri vseh oblikah stavka FOR velja, da je ime spremenljivke označene kot variable lahko dolgo le en znak! Njeno vrednost v zanki dobimo tako, da zapišemo %%variable. Če želimo stavek FOR napisati čez več vrstic, moramo vrstico deliti takoj za okroglim oklepajem.

Primeri različnih oblik stavka FOR so naslednji:

```
FOR %%N IN (*) DO (ECHO %%N)
FOR /R C:\ %%N IN (*) DO (ECHO %%N)
FOR /D %%N IN (*) DO (ECHO %%N)
FOR /R C:\ /D %%N IN (*) DO (ECHO %%N)
FOR /L %%N IN (0,2,10) DO (ECHO %%N)
FOR /F "skip=10 tokens=1-3,5" %%A IN (STUDENTS.TXT) DO (
  ECHO %%A %%B %%C %%D
)
```

Prvi stavek FOR izpiše vse datoteke v trenutnem imeniku.

Drugi stavek FOR izpiše vse datoteke v podanem imeniku in v vseh njegovih podimenikih.

Tretji stavek FOR izpiše vse imenike v trenutnem imeniku.

Četrti stavek FOR izpiše vse imenike v podanem imeniku in v vseh njegovih podimenikih.

Peti stavek FOR je C-jevska zanka for in izpiše števila od vključno 0 do vključno 10 s korakom 2, torej se izpišejo števila 0,2,4,6,8,10.

Zadnji stavek FOR pregleduje tekstovno datoteko STUDENTS.TXT, najprej preskoči prvih 10 vrstic, nato pa v vsaki naslednji vrstici vzame prve tri besede in peto besedo, ter jih izpiše.

Stavek FOR nam omogoča elegantne operacije nad polji, kakor kaže naslednji primer. Moramo pa upoštevati, da lupina Windows command polj pravzaprav ne pozna, za njo so npr spremenljivke T[1], T[2], T[3] itd. čisto navadne spremenljivke s podobnim imenom.

```
@ECHO OFF
SET T[1]=5
SET T[2]=10
SET T[3]=15
SET T[4]=20
SET SUM=0
FOR /L %%A IN (1,1,4) DO (
  SET /A SUM = SUM + T[%%A]
)
ECHO SUM=%SUM%
```

Še posebej pomemben je stavek FOR zato, ker omogoča shranjevanje rezultata poljubnega ukaza v spremenljivko. To dosežemo z uporabo kretnice /F in ključno besedo "USEBACKQ" kot opcijo. Ukaz, ki se naj izvede, mora biti podan v vzvratnih narekovajih. Tukaj je primer:

```
FOR /F "USEBACKQ DELIMS== TOKENS=2" %%T IN (`ASSOC .GIF`) DO SET NAME=%%T
FOR /F "USEBACKQ DELIMS== TOKENS=2" %%T IN (`FTYPE %NAME%`) DO SET PROG=%%T
ECHO %PROG%
```

Prvi ukaz FOR izvede ukaz ASSOC .GIF. Rezultat ukaza se razdeli na besede, ki so med seboj ločene z znakom =. Druga beseda v rezultatu se shrani v spremenljivko NAME. Nato se izvede drugi FOR. Ta izvede ukaz FTYPE %NAME%, pri čemer uporabi prej nastavljeno vrednost spremenljivke NAME. Rezultat se spet razdeli v besede ločene z znakom = in druga beseda se shrani v spremenljivko PROG. Ta se nato s stavkom ECHO izpiše.

Za bolj zahtevne skripte imamo na voljo stavek GOTO, ki ga uporabimo na naslednji način:

```
@ECHO OFF
ECHO zacetek
GOTO END
ECHO sredina
:END
ECHO konec
```

Podoben je tudi stavek CALL, s katerim skočimo v podprogram. Podprogramu lahko podamo parametre. Ta stavek je zelo pomemben, saj z njegovo pomočjo obidemo neprijetno omejitev lupine Windows command, da se vrednosti vsem spremenljivke znotraj enega bloka določijo na začetku in se potem ne spreminjajo več. Kot primer si oglejmo naslednjo skripto:

```
@ECHO OFF
SET UKAZ=NULL
FOR /L %%A IN (1,1,5) DO (
  SET /P UKAZ="Vnesi ukaz: "
  ECHO Pod zaporedno stevilko %%A si vnesel ukaz %UKAZ%.
)
```

Ta skripta bo petkrat vprašala, naj vnesemo ukaz, vendar pa ne bo vnesenega ukaza nikoli izpisala ampak se bo petkrat izpisalo, da smo vnesli ukaz NULL. Problem je v tem, da se vse spremenljivke v telesu zanke FOR nadomestijo s trenutnimi vrednostmi, še preden se zanka začne izvajati. Da bi dosegli željeno delovanje skripte, moramo uporabiti podprogam.

```
@ECHO OFF
SET UKAZ=NULL
FOR /L %%A IN (1,1,5) DO (
  CALL :OBDELAVA %%A
)
GOTO :EOF

:OBDELAVA
SET /P UKAZ="Vnesi ukaz: "
ECHO Pod zaporedno stevilko %1 si vnesel ukaz %UKAZ%.
GOTO :EOF
```

V isti vrstici lahko naštejemo več ukazov. To imenujemo veriženje ukazov. Operatorji, ki jih lahko uporabimo pri veriženju ukazov, so:

- ukaz1 & ukaz2 - najprej izvede ukaz1 nato pa ukaz2
- ukaz1 && ukaz2 - najprej izvede ukaz1 in če NE pride do napake nato izvede ukaz2
- ukaz1 || ukaz2 - najprej izvede ukaz1 in če pride do napake nato izvede ukaz2
- () - z okroglimi oklepaji lahko določimo vrstni red, v katerem se bodo ukazi izvajali.

Nekaj primerov uporabe veriženja ukazov je tukaj:

```
DIR *.LOG || ECHO Nisem nasel nobene datoteke *.LOG
IF EXIST C:\Vaje\Naloga.txt DEL C:\Vaje\Naloga.txt & ECHO Zbrisano
CD C:\VSO && (COPY *.TXT E:\ & COPY *.BAK E:\) && (DEL *.TXT & DEL *.BAK)
```

Med izvajanjem skripte lahko preklopimo v novo lupino. To storimo z ukazom CMD, enako kot pri zagonu lupine Windows command iz menuja Start->Zaženi. Ukaz CMD pozna tudi nekatere kretnice, s katerimi vplivamo na izvajanje ukazov in skript v novi lupini. Te kretnice so še posebej koristne, če tvorimo bližnjico do skripte. Najpomembnejši kretnici sta /C in /K. Vse, kar jima sledi, je ukaz, ki se bo izvedel v novi lupini. Razlika med njima je v tem, da se pri kretnici /C po dokončanju podanega ukaza nova lupina zapre, pri kretnici /K pa ne.

V skripti (ne pa tudi v menuju Start->Zaženi) lahko novo lupino zahtevamo tudi z ukazom START. Razlika med START in CMD je v tem, da se pri ukazu START po privzetem odpre novo okno, pri ukazu CMD pa ne. Razlika je tudi v tem, da ima START drugačne kretnice. Med katerimi so najbolj zanimive naslednje:

- /Dpot - začetni imenik
- /B - zagon programa brez novega okna.
- /MIN - zažene minimirano okno
- /MAX - zažene maksimirano okno

Dodatno lahko pred vsemi kretnicami in imenom programa navedemo niz v narekovajih, v tem primeru se ta niz uporabi kot tekst v naslovni vrstici novega okna. Primer ukaza, ki v novem oknu z naslovom „Vsebina imenika“ povečanem na največjo velikost izpiše vsebino imenika `C:\` je tukaj:

```
START "Vsebina imenika" /DC:\ /MAX DIR
```

Pri pisanju skript včasih potrebujemo naključno število. V lupini Windows command to ni noben problem, saj spremenljivka `%RANDOM%` v vsakem trenutku vsebuje naključno število od vključno 0 do vključno 32767.

Na koncu naštejmo še nekaj drugih ukazov, ki jih pozna lupina Windows command. Podrobnejši opis vsakega ukaza dobimo, če uporabimo ukaz `HELP.`

- `CLS` - zbriše ekran
- `COLOR` - nastavi barvo ozadja in barvo teksta,
- `DOSKEY` – prikroji mehanizem hranjenja zgodovine ukazov
- `SETLOCAL` - shrani trenutne vrednosti vseh spremenljivk
- `ENDLOCAL` - povrne shranjene vrednosti spremenljivk
- `EXIT` - zaključi skripto in zapre okno
- `PAUSE` - ustavi izvajanje skripte, dokler uporabnik ne pritisne poljubne tipke
- `REM` - vse do konca vrstice je komentar
- `SHIFT` - odstrani prvi parameter, drugi parameter postane prvi, tretji postane drugi itd.
- `PROMPT` – spremeni pozivni niz
- `TITLE` - spremeni tekst, ki je napisan zgoraj v imenu okna
- `VER` - izpiše, katero različico operacijskega sistema uporabljamo

5. OBDELAVA TEKSTOVNIH DATOTEK

Najpomembnejši element obdelave tekstovnih datotek je iskanje določenega niza. Lupina Windows command ima za to operacijo dva ukaza: `FIND` in `FINDSTR`. Razlikujeta se v tem, da `FINDSTR` omogoča bolj zahtevna iskanja in uporabo regularnih izrazov (regularnim izrazom je posvečeno posebno poglavje v nadaljevanju).

Tukaj je nekaj primerov.

Poišči in izpiši vse vrstice, ki vsebujejo niz „skripta“ v datoteki `Besedilo.txt`, ob tem izpiši tudi številke vrstic:

```
FIND /N "skripta" Besedilo.txt
```

Poišči in izpiši vse vrstice, ki vsebujejo niz „skripta“ v datoteki `Besedilo.txt`, pri čemer ne loči med malimi in velikimi črkami:

```
FIND /I "skripta" Besedilo.txt
```

Poišči in izpiši vse vrstice, ki NE vsebujejo znaka „.“ v datoteki `Besedilo.txt`:

```
FIND /V "." Besedilo.txt
```

Preštej in izpiši število vrstic, ki vsebujejo niz „skripta“ v vseh datotekah s končnico TXT v trenutnem imeniku:

```
FIND /C "skripta" *.TXT
```

Poišči in izpiši vse vrstice, ki vsebujejo niz „ime“ kot samostojno besedo v datoteki `Besedilo.txt` (levo in desno od besede so lahko presledki ali pa posebni znaki kot npr. vejica, enačaj itd.):

```
FINDSTR "\<ime\>" Besedilo.txt
```

Lupina Windows command pozna tudi ukaz `SORT`, ki po abecedi uredi vrstice, ki jih vnesemo. Računalnik najprej čaka, da vnesemo besedilo. Vnos besedila končamo tako, da pritisnemo kombinacijo tipk CTRL+Z in nato tipko ENTER. Nato se izpišejo vnesene vrstice urejene po abecedi. Ukaz `SORT` je zelo uporaben v kombinaciji s preusmeritvami, ki jih bomo obravnavali v nadaljevanju.

6. PREUSMERITVE

Preusmeritve so preprost in zelo koristen mehanizem za kombiniranje ukazov. V splošnem obstajata dva načina delovanja ukazov:

- ukaz bere podatke s tipkovnice oz. piše rezultate na ekran,
- ukaz bere podatke iz datoteke oz. piše rezultate v datoteko.

Bolj zanimiv je prvi primer. Če ukaz bere podatke s tipkovnice lahko vzamemo poljubno tekstovno datoteko in mu povemo, da naj se obnaša tako, kot da mu jo bomo natipkali. To naredimo tako, da uporabimo operator „<“. Na primer. z naslednjim ukazom po abecedi uredimo vrstice tekstovne datoteke `Imena.txt`, rezultat se izpiše na ekran:

```
SORT < Imena.txt
```

Če ukaz piše na ekran, lahko njegov izhod preusmerimo v datoteko. To dosežemo tako, da uporabimo operator „>“ ali pa „>>“. Razlika me njima je v tem, da operator „>“ tvori novo datoteko s podanim imenom, morebitno obstoječo z nekim imenom pa zbriše. Operator „>>“ pa tvori novo datoteko le v primeru, da datoteka z navedenim imenom še ne obstaja, drugače pa doda izpis na konec obstoječe datoteke. Tukaj je primer, kako preurejeno datoteko zapišemo v novo datoteko in ne na ekran:

```
SORT < Imena.txt > Urejeno.txt
```

Poseben primer nastopi, če pride med izvajanjem ukaza do kakšne napake. V tem primeru se obvestilo o napaki izpiše na ekran ne glede na to, ali ukaz piše na ekran ali v datoteko. Če želimo tudi obvestila o napaki preusmeriti v datoteko uporabimo operator „2>“ ali pa „2>>“. Primer:

```
DIR Vaje > Seznam.txt 2> Napaka.txt
```

V zgornjem primeru se vsebina imenika `Vaje` napiše v datoteko `Seznam.txt`, če imenik Vaje ne obstaja, pa se obvestilo o napaki napiše v datoteko `Napaka.txt`. Rezultata in napak ne moremo preusmeriti v isto datoteko!

Kaj pa, če želimo izhod enega ukaza uporabiti kot vhod drugega ukaza. Lahko bi izhod enega ukaza preusmerili v neko datoteko, to datoteko pa bi potem uporabili kot vhod v drug ukaz. Enostavneje to naredimo z operatorjem „|“, ki mu rečemo cev (v angleščini „pipe“). Npr. ukaz, ki po abecedi uredi datoteko `Imena.txt` in rezultat napiše na ekran, bi lahko zapisali takole:

```
TYPE Imena.txt | SORT
```

Pri preusmeritvi s cevjo se ukazi izvajajo od leve proti desni tako, da je izhod levega ukaza vhod desnemu ukazu. Če želimo, da se nek dolg izpis ustavi na koncu vsake strani, uporabimo preusmeritev s cevjo v ukaz `MORE`, npr. `DIR | MORE`. Pri ukazu `DIR` bi enak učinek dosegli s kretnico `/P`, nimajo pa vsi ukazi take kretnice.

Pa še en primer, ki po abecedi izpiše vse procese, ki so v stanju „Running“:

```
TASKLIST /V | FIND "Running" | SORT
```

Poseben primer preusmeritve je preusmeritev izpisa na odložišče. To dosežemo z uporabo besede `CLIP`.

```
TASKLIST | CLIP
```