Systemy Operacyjne

Część II
Zarządzanie/Administracja Systemem

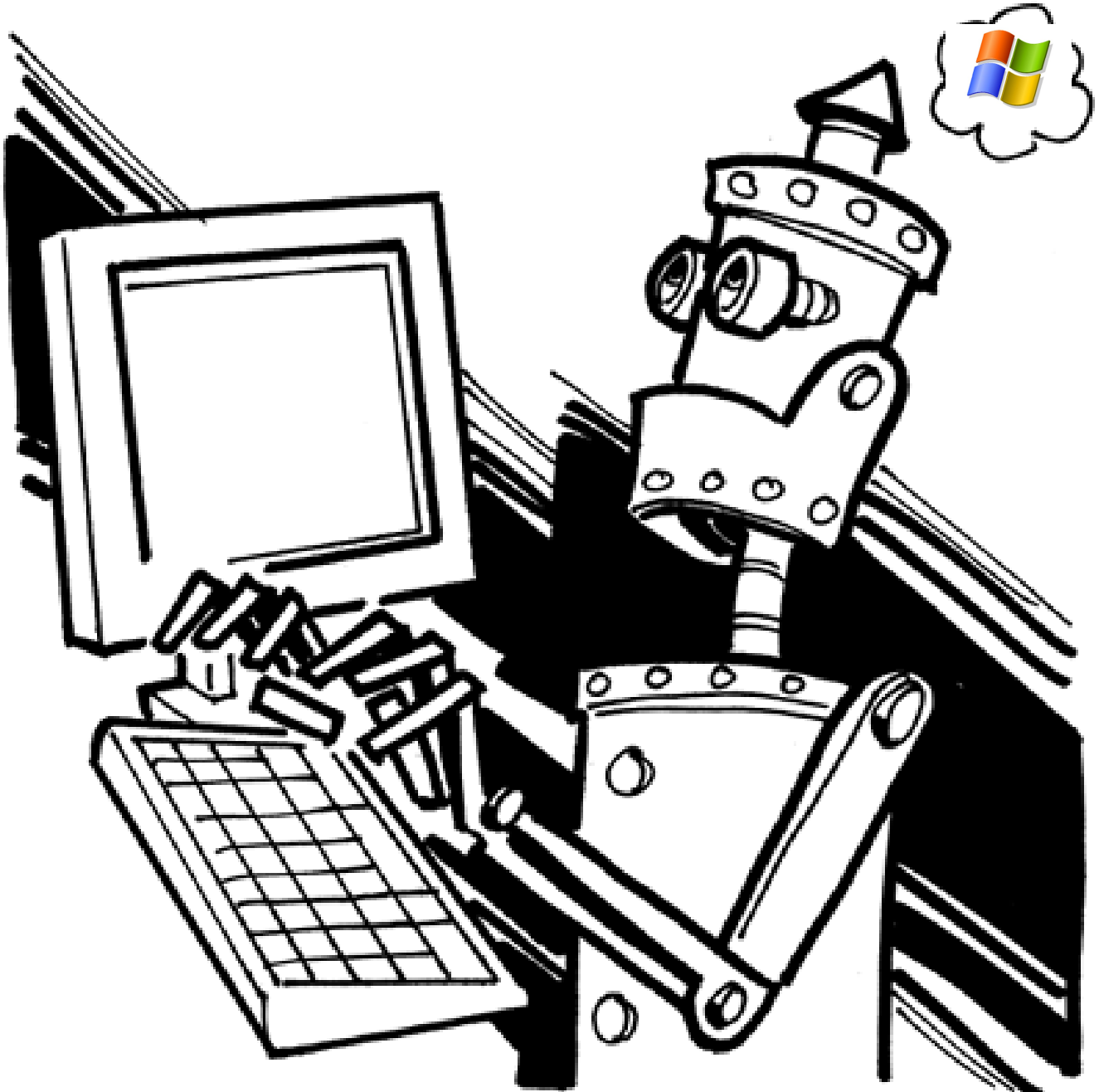
5: “Skrypty”

autor: mgr inż. Andrzej Woźniak

Czym jest skrypt?

- plik tekstowy zawierające polecenia interpretera poleceń
- polecenia są wykonywane jedno po drugim od początku do końca pliku
- aplikacja wykonywana w środowisku tekstowym (terminal, konsola)

Windows shell Scripting



Argumenty skryptu

- C:\>skrypt1 c: e:\ /p

- %0 %1 %2 %3 ... %9

Polecenia wewnętrzne i zewnętrzne

- Wewnętrzne są wbudowane w interpreter
  - assoc, call, cd, cls, color, copy, date, dir, dpath, echo, endlocal, del, exit, for, ftype, goto, if, md, move, path, pause, popd, prompt, pushd, rem, rename, rd, set, setlocal, shift, start, time, title, type, ver
- Zewnętrzne są plikami wykonywalnymi

Zmienne PATH i PATHEXT

- PATH – definiuje ścieżkę przeszukiwań

  path c:\bin;c:\skrypty;d:\winnt

- PATHEXT – definiuje listę rozszerzeń plików wykonywalnych

  set PATHEXT=.COM;.EXE;.BAT;.CMD

Kontrolowanie wyjścia skryptu

- REM – komentarz
- CLS – czyszczenie okna konsoli
- COLOR – określa kolory konsoli
- TITLE – zmienia tekst w pasku tytułu okna
- @ - wyłącza echo dla pojedynczego polecenia
- echo – wyświetla teksty
- echo on/off – włącza/wyłącza echo polecenia

Przeadresowanie wejścia i wyjścia

- >plik - wyjście do pliku (zastępowanie)
- >>plik - wyjście do pliku (dopisywanie)
- <plik - wejście z pliku
- 2>plik - wyjście błędów do pliku
- 2>&1 - wyjście błędów = wyjście polecenia
- p1 | p2 - potok: wyjście polecenia p1 jest wejściem polecenia p2 (bez wyświetlania na konsoli)

Polecenia składane

- p1 & p2 - wykonanie p1 a następnie p2
- p1 && p2 - wykonanie p1 a następnie p2, gdy p1 wykonało się poprawnie
- p1 || p2 - wykonanie p1 a następnie p2, gdy p1 nie wykonało się poprawnie
- () - złożone sekwencje poleceń

Zmienne środowiskowe

- Wbudowane zmienne systemowe
- Środowisko systemowe (HKEY_LOCAL_MACHINE)
- Wbudowane zmienne użytkownika
- Środowisko użytkownika (HKEY_CURRENT_USER)
- Polecenia set z autoexec.bat
- Polecenia set ze skryptu logowania

Ustawianie zmiennych

- set LOK=d:\bin - ustawienie
- set LOK - wyświetlenie
- set LOK= - usunięcie
- echo %LOK% - użycie

Zasięg zmiennych

- Polecenia set wpływają tylko na środowisko bieżącego interpretera
- zmiany są globalne w ramach środowiska interpretera
- SETLOCAL i ENDLOCAL umożliwiają ograniczanie zasięgu zmiennych w skrypcie

Zaawansowana składnia set

- Operatory arytmetyczne (+ - * / %)

  set /a x=12+14/5
- Formaty liczb (18=0x12=0b10010)
- Operatory bitowe (<< >> & ^ |)
- Operatory przypisania (+= /= ...)

Zastępowanie łańcuchów

%zmienna:łańcuch1=łańcuch2%

```
set PATH=c:\bin;c:\dos;c:\winnt
set PATH=%PATH:c:=d:%
set PATH
PATH=d:\bin;d:\dos;d:\winnt
```

Indeksowanie łańcuchów

%zmienna:~od_znaku, długość%

```
set X=.CMD;.BAT;.EXE
echo %X:~5,4%
.BAT
```

Sterowanie pracą skryptu

- Etykiety i polecenie GOTO
- Polecenie CALL (procedury)
- Polecenie IF
  - if ERRORLEVEL poziom polecenie
  - if CMDEXTVERSION wersja polecenie
  - if DEFINED nazwa_zmiennej polecenie
  - if [/i] tekst1==tekst2 polecenie
  - if exist plik polecenie
  - if [/i] wartość operator wartość polecenie

Polecenie FOR

- Iterator plików

  for %zmienna in (zbiór) do polecenie

  for %i in (*.bat) do echo %i

Polecenie FOR

- Iterator katalogów

  for /d %zmienna in (zbiór) do polecenie

  for /d %i in (c:\) do echo %i

Polecenie FOR

- Iterator plików w drzewie

  for /r [ścieżka] %zmienna in (zbiór) do polecenie

  for /r c:\ %i in (*.bat) do echo %i

Polecenie FOR

- Iterator liczbowy

  for /l %zmienna in (start,krok,koniec) do polecenie

  for /l %i in (1,1,5) do echo %i

  1

  2

  3

  4

  5

Polecenie FOR

- Analiza tekstów

  for /f [opcje] %zmienna in (źródło) do polecenie

  żródło:

  nazwa lub zbiór nazw plików
  tekst w cudzysłowie
  polecenie otoczone apostrofami

  for /f %i in (autoexec.bat) do echo %i

Polecenie FOR

- Opcje analizy tekstu
  - eol=znak definicja znaku końca wiersza
  - skip=nn opuszczenie nn początkowych wierszy
  - delims=xxx definicja znaków oddzielających tokeny
  - tokens=ttt lista tokenów, które zostaną przypisane do zmiennych

Przykład:

```
for /f “tokens=2” %%i in ('net use ^| find “:” ^| find /i “
   \\Osrv\doc”') do echo %%i
```

Unix - Powłoki

- Sh - Bourne shell, oryginalna powłoka uniksowa
- csh - C shell, o składni podobnej do C
- ksh - Korn shell, powłoka o możliwościach powłoki C, ale zgodna z powłoką Bourne`a
- bash - Bourne Again shell - freewarowy odpowiednik powłoki Korna
- tcsh - TC shell, rozbudowana powłoka C

Powłoka domyślna

- Linux - bash
- Mac OS X - bash lub tcsh
- Solaris - ksh
- HP-UX - ksh
- System V Unix - ksh
- QNX 6 - ksh
- Cygwin (Windows) - bash
- SFU (Windows) - ksh (niekompletny)

Symbole wieloznaczne

- * - dowolny ciąg znaków (także pusty)
- ? - dowolny jeden znak
- [zakres] - ograniczenie zakresu znaków

Przykład:

ls [Aa]*z??a.txt

Uruchamianie poleceń w tle

ls -R [Aa]*z??a.txt > wynik &

Echo bez znaku nowego wiersza

echo -n „Komunikat”

Przykład:

```
echo -n „Twój login name: „
whoami
```

Wynik:

```
Twój login name: robert
```

Unix - zmienne

- sh, ksh,bash ... (zgodne z Bourne shell)
  - NAZWA=WARTOŚĆ
- csh, tcsh ... (zgodne z C shell)
  - set nazwa = wartość
- użycie zmienne
  - $NAZWA
  - ${NAZWA}
- zwalnianie zmiennej
  - unset NAZWA

Unix - interakcja - czytanie

- Czytanie danych podawanych przez użytkownika z klawiatury
  - read NAZWA_ZMIENNEJ

Przykład:

```
echo -n „Podaj hasło: „
read HASLO
```

Unix długie polecenia

Zakończenie wiersza znakiem \

oznacza, że ciąg dalszy polecenia znajduje się w następnym wierszu

Przykład:

```
ls \
-CF \
-l \
/usr/
```

Unix - pętla for

```
for ZMIENNA in lista_elementów
do
    polecenie1
    polecenie2
done
```

Przykład:

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

Unix - pętla for

Przykład:

```
for I in 1 2 3 4 5 6 7 8 9 10
do
        echo -n „...$I”
done
```

Przykład (bash):

```
max=10
for (( i=1; i<=max ; i++)
do
        echo -n „...$I”
done
```

Unix - instrukcja warunkowa if

```
if (warunek) then
   polecenie 1
   polecenie 2
else
   polecenie3
   polecenie4
fi
```

Unix - instrukcja warunkowa if

Przykład:

```
if (ls *.txt > /dev/null) then
    echo „Twierdzenie \” Pliki .txt istnieją \” to prawda”
else
    echo „Twierdzenie \” Pliki .txt istnieją \” to nieprawda”
fi
```

Unix - strumienie standardowe

Z każdym programem skojarzone są trzy standardowe strumienie (pliki):

- stdin - strumień wejściowy (klawiatura)
- stdout - strumień wyjściowy (ekran)
- stderr - strumień diagnostyczny (ekran)

Unix - przekierowanie strumienia

- Przykład 1:

  ls /fred > /dev/null

  ls: /fred: No such file or directory

- Przykład 2:

  ls /fred > /dev/null 2>dev/null

Unix - Instrukcja test

- Porównywanie liczb:

test $x -eq $y - prawda jeśli x=y

test $x -ne $y - prawda jeśli x<>y

test $x -gt $y - prawda jeśli x>y

test $x -ge $y - prawda jeśli x>=y

test $x -lt $y - prawda jeśli x<y

test $x -le $y - prawda jeśli x<=y

Unix - Instrukcja test

- Porównywanie tekstów

test „$s1” = „$s2” - prawda jeśli ciągi identyczne

test „$s1” != „$s2” - prawda jeśli ciągi różne

test $s1 - prawda jeśli ciąg niepusty

test $s1 -z - prawda jeśli długość ciągu=0

test $s1 -n - prawda jeśli długość ciągu>0

Unix - Instrukcja test

- Testowanie plików

test -d nazwa_pliku - prawa jeśli istnieje i jest katalogiem

test -e nazwa_pliku - prawda jeśli istnieje

test -f nazwa_pliku - prawda jeśli istnieje i jest plikiem

test -r nazwa_pliku - prawda jeśli istnieje i można czytać

test -s nazwa_pliku - prawda jeśli istnieje i rozmiar <> 0

test -w nazwa_pliku - prawda jeśli istnieje i można zapisać

test -x nazwa_pliku - prawda jeśli istnieje i można uruchomić

Unix - Instrukcja test

- Operatory logiczne
  - ! - negacja
  - -a - iloczyn (AND)
  - -o - suma (OR)

Przykłady:

```
if (test ! $x -eq $y) then
if (test $x -ne $y -a $x -lt $y) then
```

Unix - instrukcja wybory case

```
Case ZMIENNA in
wartość1)
    polecenie1
    polecenie2
;;
wartość2)
    polecenie3
    polecenie4
;;
*)
    polecenie5
    polecenie6
;;
esac
```

Unix - petla while

```
while [test]
do
   polecenie1
   polecenie2
done
```

Unix - petla until

```
until [test]
do
   polecenie1
   polecenie2
done
```

Unix - argumenty

- $0 - nazwa skryptu
- $1 do $9 - kolejne argumenty
- $# - liczba argumentów
- $* - wszystkie argumenty

Unix - pierwszy wiersz skryptu

#!/pełna/ścieżka/do/interpretera

Przykłady:

#!/bin/sh

#!/bin/bash

Alternatywne języki skryptowe

- Perl
- Python
- Tcl
- java
- VB