Narzędzia informatyczne w językoznawstwie

Wiersz poleceń - Potoki i pliki wsadowe

Marcin Junczys-Dowmunt
junczys@amu.edu.pl

Zakład Logiki Stosowanej
http://www.logic.amu.edu.pl

14. listopada 2007

Polecenia działające na plikach tekstowych

- `echo` – Wyświetla komunikat wpisany jako argument
  np. `echo To jest komunikat`

Polecenia działające na plikach tekstowych

- `echo` – Wyświetla komunikat wpisany jako argument
  np. `echo To jest komunikat`
- `type` – Wyświetla zawartość jednego lub wielu plików na `stdout` np. `type *.txt`

Polecenia działające na plikach tekstowych

- `echo` – Wyświetla komunikat wpisany jako argument
  np. `echo To jest komunikat`
- `type` – Wyświetla zawartość jednego lub wielu plików na `stdout` np. `type *.txt`
- `more` – Wyświetla dane po jednym ekranie na raz
  np. `more plik1.txt`

Polecenia działające na plikach tekstowych

- `echo` – Wyświetla komunikat wpisany jako argument
  np. `echo To jest komunikat`
- `type` – Wyświetla zawartość jednego lub wielu plików na `stdout` np. `type *.txt`
- `more` – Wyświetla dane po jednym ekranie na raz
  np. `more plik1.txt`
- `sort` – sortuje wierze w podanym pliku i wyświetla na konsoli
  np. `sort plik1.txt`

Polecenia działające na plikach tekstowych

- `echo` – Wyświetla komunikat wpisany jako argument
  np. `echo To jest komunikat`
- `type` – Wyświetla zawartość jednego lub wielu plików na `stdout` np. `type *.txt`
- `more` – Wyświetla dane po jednym ekranie na raz
  np. `more plik1.txt`
- `sort` – sortuje wierze w podanym pliku i wyświetla na konsoli
  np. `sort plik1.txt`
- `fc` – porównuje dwa pliki i wyświetla różnice między nimi
  np. `fc plik1.txt plik2.txt`

Polecenia działające na plikach tekstowych

- `echo` – Wyświetla komunikat wpisany jako argument
  np. `echo To jest komunikat`
- `type` – Wyświetla zawartość jednego lub wielu plików na `stdout` np. `type *.txt`
- `more` – Wyświetla dane po jednym ekranie na raz
  np. `more plik1.txt`
- `sort` – sortuje wierze w podanym pliku i wyświetla na konsoli
  np. `sort plik1.txt`
- `fc` – porównuje dwa pliki i wyświetla różnice między nimi
  np. `fc plik1.txt plik2.txt`
- `find` – szuka ciągu znaków w pliku lub wielu plikach
  np. `find /N "ciąg" *.txt`

Co to jest potok?

Potok (ang. pipe)

- to jeden z mechanizmów komunikacji międzyprocesowej umożliwiający wymianę danych pomiędzy dwoma procesami

Co to jest potok?

Potok (ang. pipe)

- to jeden z mechanizmów komunikacji międzyprocesowej umożliwiający wymianę danych pomiędzy dwoma procesami
- odbywa się to poprzez połączenie standardowego wyjścia jednego procesu ze standardowym wejściem drugiego

Co to jest potok?

Potok (ang. pipe)

- to jeden z mechanizmów komunikacji międzyprocesowej umożliwiający wymianę danych pomiędzy dwoma procesami
- odbywa się to poprzez połączenie standardowego wyjścia jednego procesu ze standardowym wejściem drugiego
- liczba procesów, które można w ten sposób połączyć nie jest ograniczona

Co to jest potok?

Potok (ang. pipe)

- to jeden z mechanizmów komunikacji międzyprocesowej umożliwiający wymianę danych pomiędzy dwoma procesami
- odbywa się to poprzez połączenie standardowego wyjścia jednego procesu ze standardowym wejściem drugiego
- liczba procesów, które można w ten sposób połączyć nie jest ograniczona

Polecenia łączymy w potok za pomocą znaku `|`, np.

```
type plik1.txt | sort | more
```

Standardowe strumienie

To standardowe kanały komunikacji między komputerem a otoczeniem (zwykle terminalem).

Standardowe strumienie

To standardowe kanały komunikacji między komputerem a otoczeniem (zwykle terminalem).
Trzy podstawowe połączenia I/O noszą nazwy:

- standard input (**stdin**, standardowy strumień wejścia),

Standardowe strumienie

To standardowe kanały komunikacji między komputerem a otoczeniem (zwykle terminalem).
Trzy podstawowe połączenia I/O noszą nazwy:

- standard input (**stdin**, standardowy strumień wejścia),
- standard output (**stdout**, standardowy strumień wyjścia)

Standardowe strumienie

To standardowe kanały komunikacji między komputerem a otoczeniem (zwykle terminalem).
Trzy podstawowe połączenia I/O noszą nazwy:

- standard input (**stdin**, standardowy strumień wejścia),
- standard output (**stdout**, standardowy strumień wyjścia)
- standard error (**stderr**, standardowy strumień błędów).

Standardowy strumień wyjścia (stdout)

Standardowy strumień wyjścia

- Standardowy strumień wyjścia to strumień, do którego program zapisuje dane wynikowe

Standardowy strumień wyjścia (stdout)

Standardowy strumień wyjścia

- Standardowy strumień wyjścia to strumień, do którego program zapisuje dane wynikowe
- Niektóre programy nie zwracają danych wynikowych - na przykład `move` niczego nie wypisuje, jeżeli przeniesienie się powiodło

Standardowy strumień wyjścia (stdout)

Standardowy strumień wyjścia

- Standardowy strumień wyjścia to strumień, do którego program zapisuje dane wynikowe
- Niektóre programy nie zwracają danych wynikowych - na przykład `move` niczego nie wypisuje, jeżeli przeniesienie się powiodło
- Jeżeli strumień nie jest przekierowany dane są wysyłane do terminala, z którego uruchomiono program

Standardowy strumień wyjścia (stdout)

Standardowy strumień wyjścia

- Standardowy strumień wyjścia to strumień, do którego program zapisuje dane wynikowe
- Niektóre programy nie zwracają danych wynikowych - na przykład `move` niczego nie wypisuje, jeżeli przeniesienie się powiodło
- Jeżeli strumień nie jest przekierowany dane są wysyłane do terminala, z którego uruchomiono program

Aby przekierować strumień wyjścia do pliku, należy użyć znaków

- > lub 1> – jeżeli chcemy stworzyć dany plik lub zastąpić plik danymi ze strumienia
- >> lub 1>> – jeżeli chcemy dopisać dane na końcu pliku

Standardowy strumień wejścia (stdin)

Standardowy strumień wejścia

- Standardowy strumień wejścia to dane (zwykle tekst) przekazywane do programu

Standardowy strumień wejścia (stdin)

Standardowy strumień wejścia

- Standardowy strumień wejścia to dane (zwykle tekst) przekazywane do programu
- Nie wszystkie programy wymagają danych wejściowych. Przykładowo, `dir` wykonuje swoją funkcję nie pobierając żadnych danych z `stdin`

Standardowy strumień wejścia (stdin)

Standardowy strumień wejścia

- Standardowy strumień wejścia to dane (zwykle tekst) przekazywane do programu
- Nie wszystkie programy wymagają danych wejściowych. Przykładowo, `dir` wykonuje swoją funkcję nie pobierając żadnych danych z `stdin`
- O ile strumień nie jest przekierowany, dane są pobierane z terminalu (czyli z klawiatury), z którego został uruchomiony program

Standardowy strumień wejścia (stdin)

Standardowy strumień wejścia

- Standardowy strumień wejścia to dane (zwykle tekst) przekazywane do programu
- Nie wszystkie programy wymagają danych wejściowych. Przykładowo, `dir` wykonuje swoją funkcję nie pobierając żadnych danych z `stdin`
- O ile strumień nie jest przekierowany, dane są pobierane z terminalu (czyli z klawiatury), z którego został uruchomiony program

Aby przekierować plik do strumienia wejścia, należy użyć znaku <

Standardowy strumień błędów (stderr)

Standardowy strumień błędów

- Standardowy strumień błędów wykorzystujemy do wyświetlania komunikatów o błędach

Standardowy strumień błędów (stderr)

Standardowy strumień błędów

- Standardowy strumień błędów wykorzystujemy do wyświetlania komunikatów o błędach
- Jest niezależny od strumienia wyjścia

Standardowy strumień błędów (stderr)

Standardowy strumień błędów

- Standardowy strumień błędów wykorzystujemy do wyświetlania komunikatów o błędach
- Jest niezależny od strumienia wyjścia
- Strumienia ma umożliwić zobaczenie błędu nawet wtedy, gdy strumień wyjścia jest przekierowany

Standardowy strumień błędów (stderr)

Standardowy strumień błędów

- Standardowy strumień błędów wykorzystujemy do wyświetlania komunikatów o błędach
- Jest niezależny od strumienia wyjścia
- Strumienia ma umożliwić zobaczenie błędu nawet wtedy, gdy strumień wyjścia jest przekierowany
- Gdy strumienie wyjścia i błędów mają ten sam cel (np. terminal) to są wyświetlane w takiej kolejności, w jakiej wypisuje je program

Standardowy strumień błędów (stderr)

Standardowy strumień błędów

- Standardowy strumień błędów wykorzystujemy do wyświetlania komunikatów o błędach
- Jest niezależny od strumienia wyjścia
- Strumienia ma umożliwić zobaczenie błędu nawet wtedy, gdy strumień wyjścia jest przekierowany
- Gdy strumienie wyjścia i błędów mają ten sam cel (np. terminal) to są wyświetlane w takiej kolejności, w jakiej wypisuje je program

Aby przekierować strumień błędu do pliku, należy użyć znaku 2> lub odpowiednio 2>>

Przykład (stdout)

- `echo To jest plik testowy > plik1.txt`
- `echo A to kolejny wiersz >> plik1.txt`
- `echo A to juz trzeci wiersz >> plik1.txt`
- `more plik1.txt`
- `type plik1.txt | sort | more`

Przykład (stdout)

- `echo To jest plik testowy > plik1.txt`
- `echo A to kolejny wiersz >> plik1.txt`
- `echo A to juz trzeci wiersz >> plik1.txt`
- `more plik1.txt`
- `type plik1.txt | sort | more`

- `sort < plik1.txt`
- `type plik1.txt > plik2.txt`

Przykład (stdout)

- `echo To jest plik testowy > plik1.txt`
- `echo A to kolejny wiersz >> plik1.txt`
- `echo A to juz trzeci wiersz >> plik1.txt`
- `more plik1.txt`
- `type plik1.txt | sort | more`

- `sort < plik1.txt`
- `type plik1.txt > plik2.txt`

- `type * > all.txt`
- `type * > all.txt 2> errors.txt`

Urządzenia specjalne `con` i `nul`

con

Urządzenie `con` jest symbolem konsoli

- Jeśli czytamy z `con`, to czytami z klawiatury
  `type con >plik.txt` (Należy naciśnąc `Ctrl+Z` i następnie Enter, żeby przerwać wczytywanie z klawiatury)
- Możemy też zapisywać informacje do `con`, ale jest to często zachowaniem standardowym poleceń

Urządzenia specjalne con i nul

con

Urządzenie con jest symbolem konsoli

- Jeśli czytamy z con, to czytami z klawiatury
  `type con >plik.txt` (Należy naciśnąc Ctrl+Z i następnie Enter, żeby przerwać wczytywanie z klawiatury)
- Możemy też zapisywać informacje do con, ale jest to często zachowaniem standardowym poleceń

nul

Wszystkie dane wysłane do nul po prostu znikają, stąd nazwa *czarna dziura*

- "Wycisza" strumienie wyjściowe, np. wyjście błędów
  `dir jakaśbzdura 2>nul`
- Można również "czytać" informacje z nul
  `type nul`

Przykładowy potok

```
perl splitter.pl -crp german_big.txt -lem lemDE.txt
<german.txt 2>nul | perl lemmatize.pl -lem lemDE.txt
| perl cleanpos.pl -mapping german.map | perl
subst_segments.pl -segs segments.txt -map subst.map
>german_lemmatized.txt
```

Przykładowy potok

```
perl splitter.pl -crp german_big.txt -lem lemDE.txt
<german.txt 2>nul | perl lemmatize.pl -lem lemDE.txt
| perl cleanpos.pl -mapping german.map | perl
subst_segments.pl -segs segments.txt -map subst.map
>german_lemmatized.txt

type reconstructed*.txt | perl find_phrases.pl -list
pattern.txt -mode long | perl subst_by_normal.pl
-normalpl polish_normalized.txt -normalen
english_normalized.txt | perl unique_cps.pl | perl
pairs_to_db.pl
```

Programy wsadowe

Program wsadowy

- to w systemach MS-DOS lub Windows plik tekstowy zawierający serię poleceń, które ma wykonać interpreter komend (np. kasowanie, kopiowanie, uruchamianie)

Programy wsadowe

Program wsadowy

- to w systemach MS-DOS lub Windows plik tekstowy zawierający serię poleceń, które ma wykonać interpreter komend (np. kasowanie, kopiowanie, uruchamianie)
- Kiedy program wsadowy zostanie uruchomiony, interpreter czyta plik i uruchamia kolejno zapisane w nim programy.

Programy wsadowe

Program wsadowy

- to w systemach MS-DOS lub Windows plik tekstowy zawierający serię poleceń, które ma wykonać interpreter komend (np. kasowanie, kopiowanie, uruchamianie)
- Kiedy program wsadowy zostanie uruchomiony, interpreter czyta plik i uruchamia kolejno zapisane w nim programy.
- Programy wsadowe systemu MS-DOS posiadają rozszerzenia .BAT lub .CMD

Programy wsadowe

Program wsadowy

- to w systemach MS-DOS lub Windows plik tekstowy zawierający serię poleceń, które ma wykonać interpreter komend (np. kasowanie, kopiowanie, uruchamianie)
- Kiedy program wsadowy zostanie uruchomiony, interpreter czyta plik i uruchamia kolejno zapisane w nim programy.
- Programy wsadowe systemu MS-DOS posiadają rozszerzenia .BAT lub .CMD
- Umożliwiają automatyzację często powtarzających się lub złożonych poleceń

Prosty przykład

Tworzymy plik tekstowy o nazwie `windows.bat` i wpisujemy:

```
@echo off
echo Uwaga, zaraż wyświetlę zawartość C:\Windows
dir /B C:\Windows | sort /R | more
echo Katalog został wyświetlony
```

Zmienne i parametry

Zmienne istotnie zwiększają funkcjonalność każdego języka skryptowego

Proszę sprawdzić działanie pliku wsadowego o następującej treści:

```
@echo off
set var=test 1 2 3
echo Wartość zmiennej to "%var%"
```

Zmienne i parametry

Zmienne istotnie zwiększają funkcjonalność każdego języka skryptowego

Proszę sprawdzić działanie pliku wsadowego o następującej treści:

```
@echo off
set var=test 1 2 3
echo Wartość zmiennej to "%var%"
```

Następnie zmodyfikować do tej postaci:

```
@echo off
set var=%1
echo Wartość zmiennej to "%var%"
```

Wykonać plik wsadowy w nast. sposób: `test.bat mały_test`

Sprawdzanie warunków

Polecenie `if` pozwala na sprawdzanie warunków logicznych na poziomie wiersza poleceń. Jednak najbardziej przydatne jest w plikach wsadowych

- `if [not] ciąg1==ciąg2 (polecenie) [else (polecenie)]`
- `if [not] exist plik (polecenie) [else (polecenie)]`

Sprawdzanie warunków

Polecenie `if` pozwala na sprawdzanie warunków logicznych na poziomie wiersza poleceń. Jednak najbardziej przydatne jest w plikach wsadowych

- `if [not] ciąg1==ciąg2 (polecenie) [else (polecenie)]`
- `if [not] exist plik (polecenie) [else (polecenie)]`

Przykład:

```
@echo off
if "%1"=="" (set kat=konsola) else (set kat=%1)
if not exist %kat% (md %kat%) else (dir %kat%)
```

Skoki i etykiety

Polecenie `goto etykieta` pozwala na przeskok miejsca w pliku wsadowym, gdzie znajduje się etykieta.

Przykład:

```
@echo off
if "%1"=="" goto error
echo Podałeś wartość "%1"
echo Uratowałeś świat
goto koniec

:error
echo Błąd! Nie podałeś strasznie ważnego parametru!
echo Więc teraz komputer wybuchnie

:koniec
echo Przeskoczyłem na koniec
```

Pętle

Polecenie `for` służy do powtarzania jednego polecenia dla każdego pliku z pewnego zbioru plików lub dla każdej wartości z pewnego zbioru wartości.

```
for %[%]i in (zbiór) do polecenie
```

Pętle

Polecenie `for` służy do powtarzania jednego polecenia dla każdego pliku z pewnego zbioru plików lub dla każdej wartości z pewnego zbioru wartości.

```
for %[%]i in (zbiór) do polecenie
```

Uwaga

Jeśli korzystamy z pętli z poziomu wiersza poleceń zmienną oznaczamy standardowo np. `%i`, w pliku wsadowym natomiast `%%i`

Pętle

Polecenie `for` służy do powtarzania jednego polecenia dla każdego pliku z pewnego zbioru plików lub dla każdej wartości z pewnego zbioru wartości.

```
for %[%]i in (zbiór) do polecenie
```

Uwaga

Jeśli korzystamy z pętli z poziomu wiersza poleceń zmienną oznaczamy standardowo np. `%i`, w pliku wsadowym natomiast `%%i`

Przykłady (w pliku wsadowym):

```
for %%i in (*.txt *.html)
copy %%i %%i.bak
for %%i in (a b c d e f) echo Wypisz literę "%%i"
for /L %%i (1,2,13) echo Plik nr %%1 > plik%%1.txt
```

Ostatni przykład - składamy wszystko w całość

```
set USER=%1%
set N=%2%
if "%USER%"=="" goto error
if "%N%"=="" goto error

cd "C:\Documents and settings\%USER%\Pulpit"

if exist konsola (del /S /Q konsola\*) else md konsola

for /L %%i in (1 1 %N%) do echo Plik %%i 1>konsola/plik%%i.txt

goto koniec

:error
echo Nie podałes uzytkownika lub liczby plików

:koniec
```