Strumienie i składanie strumieni

Paweł Józiak

Wydział Matematyki i Informatyki Politechniki Warszawskiej

3 XI 2021

Plan zajęć

Plan zajęć

Strumienie

- *Strumień* (*stream*) to jednokierunkowy kanał komunikacyjny przesyłu uporządkowanych bajtów.
- *Strumieni* używa się do:
  - czytania z oraz pisania do plików (`fopen()` tworzy strumień i przyporządkowuje go do treści pliku),
  - interakcji z użytkownikiem via klawiatura a wyświetlaniem (*TTY*),
  - komunikacji między procesami (tak zwane *składanie strumieni*),
- Proces jest nieświadomy czy jego strumień wejściowy jest związnay z:
  - plikiem,
  - urządzeniem wejścia-wyjścia,
  - innym procesem.

Strumienie standardowe

- Procesy obsługują 3 predefiniowane strumienie standardowe. Są to:
  - `stdin` (`0`) – standardowe wejście,
  - `stdout` (`1`) – standardowe wyjście,
  - `stderr` (`2`) – standardowy błąd.
- Procesy zwykle nie czytają bezpośrednio z klawiatury. Raczej: czytają z `stdin` , które jest domyślnie dowiązane z klawiaturą.
- Procesy zwykle nie piszą bezpośrednio na ekran. Raczej: piszą do `stdout` oraz `stderr` , które są domyślnie powiązane z ekranem.
- Używa się `stdout` do wyświetlania zwykłego wyjścia, zaś `stderr` do pisania komunikatów związanych z błędami.
- Ponieważ oba są domyślnie powiązane z ekranem, można je łatwo pomylić, zwłaszcza na początkowym etapie.
  ✍ *Łatwo pomylić komunikaty `stdout` i `stderr`*

Zwykłe komunikaty i komunikaty błędu

```
$ date # wyświetla komunikat do  stdout
$ cd nieistniejacy # wyświetla komunikat błędu do  stderr
```

Przekierowanie strumienia

- Strumienie standardowe mogą być przekierowane za pomocą operatorów < , > oraz >> .

Przekierowanie strumieni

```
$ ls > plik # nadpisze plik wyjściem komendy ls
$ ls >> plik # dopisze do pliku wyjście komendy ls
$ somesetup < plik # zaczyta wejście z pliku
$ ls >& plik # przekieruje wyjście oraz błędy do pliku,
nadpisując go
$ ls &> plik # to samo, co wcześniej
$ ls &>> plik # przekieruje wyjście oraz błędy do pliku,
kontynuując go
```

Przekierowywanie strumieni w Bashu

- Bash, oraz każda inna powłoka zgodna z sh, pozwala używać numerycznych identyfikatorów strumieni. Operator `2>` może być używany do przekierowania strumienia standardowego błędu, `2>&1` – dla sklejenia strumieniu standardowego błędu i standardowego wyjścia. Ten ostatni wzorzec bywa często wykorzystywany w skryptach basha.
  ✍ *Poniższe komendy są równoważne:*

Przekierowanie stdout i stderr

```
$ ls >& plik
$ ls > plik 2&>1
```

Przekierowywanie strumieni do */dev/null*

- `/dev/null` jest plikiem specjalnym, który działa jak *czarna dziura*.
- Wyjście komendy może być ignorowane przez przekierowanie go do `/dev/null` .
- Ten wzorzec jest często używany do uzyskania efektu *silent mode* skryptów powłoki.
  ✍ *Komendy w trybie cichym:*

Ignorowanie strumeni standardowego wyjścia i błędu

```
$ date >/dev/null # ignorowanie stdout
$ cd nieistniejacy_folder 2>/dev/null # ignorowanie stderr
$ date >/dev/null 2>&1 # ignorowanie obu
```

Przykłady przekierowania strumieni

Czyszczenie pliku historii

```
$ echo -n > ~/.bash_history
```

Zmiana ustawień historii

```
$ echo "HISTSIZE=4096" >> ~/.bashrc
$ echo ""HISTFILESIZE=4096" >> ~/.bashrc
```

Wyłączenie *write*

```
$ echo "mesg n" >> ~/.bashrc
```

Przygotowanie pliku różnicowego

```
$ diff plik1 plik2 > plik_roznicowy
```

Ignorowanie wiadomości *Permission denied*

```
$ find /etc -mtime -30 2>/dev/null
```

Filtry

- Wiele komend procesuje treści plików i przyjmuje nazwy plików jako argumenty.
- Otwieranie pliku wiąże się z utworzeniem strumienia i przypisaniem do niego treści pliku.
- Takie strumienie operują w tożsamy sposób co `stdin`.
- Jeśli nazwa pliku nie jest podana, `stdin` mógłby być użyty do podania treści do procesowania.
- Komendy, które procesują treści podane via `stdin` jeśli nie wskażemy nazwy pliku w argumencie, nazywamy **filtrami**.

Składanie strumieni

- `stdout` jednej komendy może posłużyć jako wejście na `stdin` drugiej komendy. Takie przekierowanie uzyskać można za pomocą operatora `|` .
- Jeśli druga komenda jest filtrem, procesuje `stdout` pierwszej komendy.
- Taką technikę nazywamy **składaniem strumieni**.
- Składanie strumieni pozwala na konstruowanie skomplikowanych instrukcji procesowania danych za pomocą łatwych narzędzi.
- Można przekierować połączony `stdout` i `stderr` do `stdin` kolejnej komendy, za pomocą operatora `|&` – rzadko stosowane.
  ✍ *Zmienne środowiskowe są domyślnie nieposortowane,* `wc -l` *wyświetla liczbę linii pliku/wejścia*

Składanie

```
$ env | sort
$ ls | wc -l
```

Natychmiastowe wywołania

- wywołanie komendy jak `$(cmd)` albo `` `cmd` `` pozwala na zamianę takiego literału na zawartość strumienia wyjściowego w miejscu wywołania; z pomięciem nowych linii.
  ✍ *Komenda `date` wyświetla (domyślnie: bieżącą) datę; `touch` tworzy pusty plik o zadanej nazwie, jeśli nie istnieje. Zwróć uwagę, że podwójny cudzysłów jest konieczny w pierwszym przykładzie dla zachowania znaczenia spacji:*

Natychmiastowe wywołanie

```
$ date
  pon, 25 paź 2021, 17:15:02 CEST
$ touch "`date`"
$ date +%d.%m.%Y
  25.10.2021
$ touch $(date +%d.%m.%Y)
```

Natychmiastowe wywołania – przykłady

Liczenie zmiennych

```
$ echo "Liczba zmiennych: $(set | wc -l)"
```

Szukanie plików modułów jądra Linuxa

```
$ find /lib/modules/$(uname -r)/kernel -name "*.ko.zst"
```

A jakby ktoś zapomniał:

```
$ echo "Nazywam się $(id -un)"
```

Dynamiczna pętla

```
$ for FILE in `ls`; do ... # uwaga na pliki ze spacjami w nazwie!
```

Plan zajęć

Przeglądarki plików

- Wszystkie programy do przeglądania plików: `more` , `less` , `head` i `tail` mogą być używane jako filtry.
  ✍ *Nie każdy TTY pozwala na przesuwanie widoku w górę:*

Listowanie zmiennych Powłoki

```
$ set | less
```

✍ *Wyświetlanie siódmej linii pliku*

Siódmy to ostatni z pierwszych siedmiu

```
$ head -n7 /etc/passwd | tail -n1
```

- `cat` również może być używany jako filtr, ale ponieważ jedynie kopiuje `stdin` do `stdout` , może nie mieć to dużego sensu.

Zliczanie słów: *wc*

Syntaks *wc*

```
$ wc [-l|-w|-c] plik
```

- Wypisuje liczbę linii (znaków końca linii, EOL), słów (sekwencji znaków nie-białych przedzielonych ciągami znaków białych) oraz znaków (bajtów) w *pliku*.
- Zwróć uwagę, że filtry nie są świadome nazwy pliku.
- Wynik może być ograniczony do linii (`-l`), słów (`-w`) bądź znaków ASCII (`-c`).
- W implementacji GNU, dostępna jest także opcja `-m` , która zlicza faktyczne znaki (por. następny slajd).
- Wiele komend generuje wynik per-rekord w nowych liniach, więc filtr `wc -l` jest szczególnie użyteczny przy zliczaniu tego wyniku.

wc – przykłady

Trick na zamaskowanie nazwy pliku

```
$ wc -l /etc/passwd
$ echo "Liczba wpisów: $(cat /etc/passwd | wc -l)"
```

Zliczanie liczby...

```
$ set | wc -l # zmiennych
$ env | wc -l # zmiennych środowiskowych
$ ls -A | wc -l # plików w bieżącym folderze
$ pacman -Q | wc -l # pakietów zainstalowanych w Arch Linux
$ getent passwd | wc -l # użytkowników lokalnych i zdalnych
```

Znaki a bajty

```
$ echo 'a' | wc -c # 2: również znak EOL.
$ echo 'a' | wc -m # 2: więc jaka różnica?
$ echo 'ą' | wc -c # 3: znak spoza ASCII, 2 bajty
$ echo 'ą' | wc -m # 2: tak to rozumiemy
```

Sortowanie linii (*sort*)

- Sortuje linie pliku bądź strumienia.
- Wszystkie opcje komendy `sort` działającej na plikach, działają także na strumieniach.

  ✍ *`env` domyślnie nie wyświetla zmiennych środowiskowych w sposób posortowany*

Sortowanie zmiennych środowiskowych

```
$ env | sort
```

Sortowanie użytkowników po identyfikatorze numerycznym

```
$ getent passwd | sort -nk3 -t:
```

Filtrowanie znaków spoza ASCII (*strings*)

- Drukuje na wyjście jedynie znaki ASCII (małe), które strumień wejściowy zawiera.
- Pozwala na wyciągnięcie z binarnego pliku jego zawartości tekstowej.
- Na plikach tekstowych działa tak samo, jak `cat`

Porównanie

```
$ ls --version
$ strings /usr/bin/ls # generuje w wyniku m.in. treść opcji
help i version
```

Parser wyrażeń regularnych (*generic regular expression parser*) – *grep*

Syntaks komendy *grep*

```
$ grep [opcje] wzorzec plik
```

- `grep` wyświetla tylko linie, w których odnaleziono dopasowanie do wzorca. Wśród najpopularniejszych opcji można wyróżnić:
  - `-i` ignorowanie wielkości liter,
  - `-r` (albo `-R`) rekursywne przeszukiwanie katalogu,
  - `-v` odwrócenie selekcji (pokaż tylko linie, w których nie ma trafień),
  - `-c` zamiast wyświetlać trafienia, zlicz je.
- `grep` czasem jest w stanie znaleźć (ale już nie wyświetlić) trafienie w pliku binarnym (np. pliku skompresowanym). W takich sytuacjach warto użyć `zgrep` .

grep – przykłady

Szukanie wzorca w pliku binarnym

```
$ strings /bin/bash | grep License
```

Szukanie plików konfiguracyjnych zawierających fragment adresu IP

```
$  grep -R 194.29.178 /etc 2>/dev/null
```

Ilu użytkowników używa powłoki innej niż nasza?

```
$ getent passwd | grep -cv "$SHELL"
```

Plan zajęć

Wyrażenia regularne POSIX

- Wzorce do przeszukiwania dowolnego tekstu.
- Kluczowy koncept teorii automatów skończonych.
- U fundamentów wielu języków programowania, m.in.
  - awk,
  - perl,
  - ruby.
- Używany w silnikach przeszukiwania m.in. w `less` (`man`), `vim` etc.
- Syntaks ustandaryzowany w ramach POSIX: BRE (*Basic Regular Expressions*) i ERE (*Extended Regular Expressions*), jednak istnieją również rozszerzenia (np. PCRE – *Perl Compatible Regular Expressions*).

Przykłady wyrażeń regularnych

- `alibaba` – odnalezienie dokładnie frazy `alibaba`,
- `ali(baba|gator)` – jedno z dwóch, `alibaba` bądź `aligator`,
- `[A-Z][a-z]*ski$` – nazwiska kończące się na -ski,
- `(ba|c|da|fi|k|tc|z)?sh` – jedna ze standardowych Powłók,
- `[(ba)c(da)(fi)k(tc)z]?sh` – to samo, ale inaczej,
- `[ \t]+` – jeden bądź więcej znaków białych, poza nową linią,
- `[ \t\n]+` – jeden bądź więcej znaków białych,
- `\s+` – to samo, co wcześniej
- `[-+]?[0-9]+[.]?[0-9]*` – liczba rzeczywista ze znakiem.
- `[-+]?[0-9]+[.]?[0-9]*(?=\s*(zł|PLN))` – liczba reprezentująca kwotę w złotówkach.
- `(?<=dnia|dn\.)\s*(\d{2}\s*(-|\.)){2}\d{4}` – data w formacie polskim

Syntaks wyrażeń regularnych ERE/PCRE cz. 1

- `x` – dokładnie znak x;
- `.` – dowolny pojedynczy znak (joker – jak ? w bashu);
- `^` – początek tekstu (dopasowanie 0-znakowe);
- `$` – koniec tekstu (dopasowanie 0-znakowe);
- `\x` – znak specjalny (np. `\t` : tabulacja, `\.` : znak kropki, `\^` : znak daszka, `\$` : znak dolara);
- `[` ab... `]` – dowolny znak spośród wskazanych (a, b itd);
- `[` a-z `]` – dowolny znak, którego kod ASCII jest w przedziale pomiędzy ASCII(a) a ASCII(z), włącznie;
- `[^` ab... `]` – dowolny znak za wyjątkiem wskazanych a, b, ...;
- `(` <r> `)` – grupa znaków w wyrażeniu regularnym;
- <r1> `|` <r2> – alternatywa: <r1> bądź <r2>;
- <r> `?` – opcjonalne wystąpienie <r>, tj. 0 bądź 1 raz;
- <r> `+` – wielokrotne wystąpienie <r>, tj. 1 bądź więcej razy;
- <r> `*` – opcjonalne wielokrotne wystąpienie <r>, tj. 0 bądź więcej razy;

Syntaks wyrażeń regularnych ERE/PCRE cz. 2

- `\s` – dowolny znak biały;
- `\S` – dowolny znak nie będący białym (negacja `\s`);
- `\d` – dowolna cyfra;
- `\D` – dowolny znak nie będący cyfrą;
- `\w` – dowolny znak *słowa*: mała bądź duża litera, cyfra oraz podkreślnik _;
- `\W` – dowolny znak nie będący znakiem *słowa*;
- `<r>{n,m}` – wystąpienie `<r>` między *n* a *m*-razy, włącznie;
- `<r>{n,}` – wystąpienie `<r>` co najmniej *n*-razy;
- `<r>{n}` – wystąpienie `<r>` dokładnie *n*-razy;
- `\b` – brzeg słowa (przejście między znakiem typu \w a \W bądź odwrotnie), dopasowanie 0-znakowe;

Syntaks wyrażeń regularnych ERE/PCRE cz. 3

- <r1> (?= <r2>) – dopasowanie <r1> ale tylko jeśli następuje po nim <r2>;
- <r1> (?! <r2>) – dopasowanie <r1> ale tylko jeśli NIE następuje po nim <r2>;
- (?<= <r1>) <r2> – dopasowanie <r2> ale tylko jeśli następuje po <r1>;
- (?<! <r1>) <r2> – dopasowanie <r2> ale tylko jeśli NIE następuje po <r1>;
- <r> *? – dopasowanie dowolną, ale możliwie małą liczbę razy (być może 0), wyrażenia <r>;
- <r> +? – dopasowanie dowolną, ale możliwie małą liczbę razy (co najmniej 1), wyrażenia <r>;

Uwagi do syntaksu wyrażeń regularnych i *grepa*

- `[0-9]+` nie oznacza ciągu powtórzeń tej samej cyfry, a raczej: niepusty ciąg jakichkolwiek cyfr.
- W pojedynczych nawiasach można określić kilka zakresów, np. `[A-Za-zĄĆĘŁÓŻŹąćęłóżź]` .
- Kropkę można reprezentować na dwa sposoby: `\.` albo `[.]` .
- `grep -E` pozwala używać wyrażeń regularnych (ERE) jako wzorców.
- `grep -P` pozwala używać wyrażeń regularnych (PCRE) jako wzorców
- W niektórych implementacjach `grep` , znak `\t` nie jest interpretowany jako tabulacja. Zamiast `grep -E` użyj wówczas `grep -P` .
- Wyrażenia regularne powinno się pisać w apostrofach, by nie były interpretowane przez Powłokę.

Wyrażenia regularne – przykłady

- Różne wyrażenia regularne szukające trafień na tym samym tekście:

Tekst przeszukiwany

```
"Ganimedes", powiedział, "jest największym naturalnym
satelitą w Układzie Słonecznym".
```

".+"

```
"Ganimedes", powiedział, "jest największym naturalnym
satelitą w Układzie Słonecznym".
```

".+?"

```
"Ganimedes", powiedział, "jest największym naturalnym
satelitą w Układzie Słonecznym".
```

(?<!^)".+?"

```
"Ganimedes ", powiedział, " jest największym naturalnym
satelitą w Układzie Słonecznym".
```