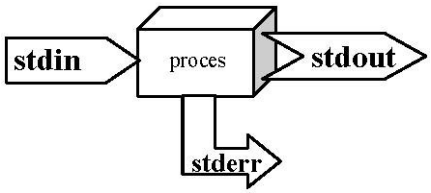
8. Strumienie i filtry w systemach Linux

8.1. Strumienie

W Linuxie z każdym procesem związane są tzw. strumienie lub potoki. Z każdym procesem związane są zwykle trzy strumienie: **stdin** - standardowy strumień wejściowy, zwykle związany z klawiaturą - z niego pobierane są znaki do obróbki przez proces (np. komendy dla powłoki), **stdout** - standardowy strumień wyjściowy, zwykle związany z ekranem - ten strumień reprezentuje wszystkie dane wyprowadzane (wyświetlane) przez program, **stderr** - standardowy strumień błędów, również zwykle związany z ekranem - na ten strumień kierowane są wszystkie komunikaty o błędach. Dzięki zastosowaniu strumieni, poszczególne procesy nie są na stałe związane z klawiaturą czy ekranem, tylko z odpowiednim strumieniem. To powłoka decyduje o tym gdzie kierować dane z poszczególnych strumieni. Dzięki temu łatwo można przekierować standardowe związanie strumieni.



Graficzna prezentacja standardowych strumieni danych

8.2. Przekierowania do plików

Aby przekierować standardowy strumień wyjściowy (**stdout**) np. do pliku, wystarczy po treści komendy użyć znaku "**>**" i podać nazwę pliku wyjściowego. Najprościej prześledzić to na przykładzie.

Komenda:

```
ls -l /etc
```

wyświetli zawartość katalogu /etc na ekran (dokładniej - do stdout, który związany jest domyślnie z ekranem). Zapis:

```
ls -l /etc > /tmp/etc.lst
```

spowoduje zapis w pliku /tmp/etc.lst listy plików w katalogu /etc. Przyjrzyjmy co się stanie w chwili wystąpienie błędów aplikacji z której przekierowujemy dane:

```
ls -l /nie_istniejacy_katalog > /tmp/cos
```

Komenda ta spowoduje utworzenie pustego pliku /tmp/cos i wyświetlenie na ekranie komunikatu:

```
ls /nie_istniejacy_katalog: No such file or directory
```

Komunikat ten trafił na ekran pomimo przekierowania **stdout**, dlatego, że jest komunikatem błędu. Komunikaty takie są wysyłanie do standardowego strumienia błędów, (**stderr**), a nie do stdout. Strumień ten również można przekierować. Dokonuje się tego podobnie jak w przypadku stdout - poprzez dodanie znaków "**2>**" i nazwy pliku po treści polecenia. Zatem:

```
ls –l /nie_istniejacy_katalog 2> /tmp/cos
```

spowoduje wpisanie komunikatu:

```
ls /nie_istniejacy_katalog: No such file or directory
```

do pliku /tmp/cos (czyli zawartość stderr), natomiast na ekran żaden komunikat nie będzie wyprowadzony.

Wszystkie dotychczasowe przekierowania powodowały utracenie dotychczasowej zawartości plików wynikowych. Stosując "**>>**" zamiast "**>**" oraz "**2>>**" zamiast "**2>**" zawartość odpowiedniego strumienia zostanie dopisana do istniejącego pliku, zatem dotychczasowa zawartość pliku zostanie zachowana.

8.3. Przekierowania z plików

Czasami przydatne jest przekierowanie standardowego strumienia wejściowego (**stdin**), na przykład przy automatyzacji pracy poleceń czy skryptów. Wówczas proces z przekierowanym **stdin** będzie pobierał znaki wejściowe z pliku, zamiast z klawiatury. Przekierowanie to uzyskuje się poprzez zastosowanie operatora "**<**" i nazwy pliku wejściowego. Na przykład, jeśli zawartość pliku /tmp/in będzie następująca:

```
ls –l /etc
echo Gotowe!
```

wówczas wywołanie polecenia:

```
bash < /tmp/in
```

spowoduje wylistowanie zawartości katalogu /etc/ oraz wyświetlenie napisu "Gotowe".

Przekierowania wszystkich trzech strumieni można ze sobą łączyć. Nawiązując do poprzedniego przykładu, poprawne jest wywołanie:

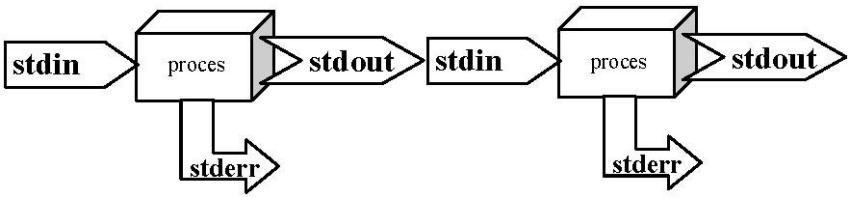
```
bash < /tmp/in >> /tmp/wynik 2> /dev/null
```

Polecenie to uruchomi kopię powłoki bash, wczyta z pliku /tmp/in komendy, wykona je, wynik pracy dołączy do pliku /tmp/wynik, a komunikaty o błędach zostaną zignorowane.

8.4. Przekierowania do aplikacji

Istnieje również możliwość przekierowania strumienia wyjściowego jednego procesu na strumień wejściowy procesu drugiego. Operacja ta nazywana jest **potokiem**. Możliwość ta jest bardzo często wykorzystywana w codziennej pracy użytkownika systemu Linux. Przekierowanie takie realizowane jest przez podanie znaku "|" na końcu treści polecenia pierwszego (tzn. tego, którego **stdout** ma być przekierowany) oraz wpisanie treści drugiego polecenia (tzn. tego, do

którego strumień ma trafić na **stdin**).



Graficzna prezentacja potoku

Realizacja praktyczna jest dość prosta. Na przykład:

```
ls -l /etc | lpr
```

spowoduje wygenerowanie listy plików z katalogu /etc i przekazanie jej na strumień **stdin** komendy "lpr" (komenda lpr stanowi interfejs linuxowego systemu wydruku). Przekierowania takie można łączyć w dłuższe sekcje, na przykład:

```
ls -R / | sort | uniq | less
```

8.5. Łączenie strumieni

Można przekierować obydwa strumienie jednocześnie, łącząc obydwie składnie. Na przykład:

```
ls -l /katalog > /tmp/wynik 2> /tmp/bledy
```

spowoduje utworzenie pliku /tmp/wynik z wynikiem działania powyższej komendy oraz pliku /tmp/bledy z treścią ewentualnych błędów. W przypadkach, kiedy należy przekierować zarówno **stdout** jak i **stderr** do tego samego pliku, należy posłużyć się operatorem "**&>**", na przykład:

```
ls -l /katalog &> /tmp/wynik
```

Po wykonaniu powyższego polecenia, zawartość **stdout** jak i **stderr** zostanie zapisana w pliku /tmp/wynik, a na ekranie nie pojawią się żadne komunikaty. Składnię tą można wykorzystać do całkowitej utraty wyniku zastosowanej komendy:

```
komenda &> /dev/null
```

8.6. Przekierowania a urządzenia

Ogromne korzyści można czerpać z połączenia mechanizmu strumieni z unixową reprezentacją urządzeń. Ponieważ systemy wywodzące się od Unixa przedstawiają urządzenia jako pliki (zazwyczaj w katalogu /dev) istnieje możliwość przekierowania danych do urządzeń np.:

```
cat tekst_do_wydruku > /dev/prn
```

spowoduje wydruk pliku (bez jakiejkolwiek interpretacji!).

Czasami przydatne jest przekierowanie **stdout** lub/i **stderr** do tzw. urządzenia pustego (/dev/null).

Wówczas cokolwiek pojawi się w strumieniu wyjściowym nie zostanie wyświetlone ani nigdzie zapisane. Np.:

```
find / –name 'costam' 2> /dev/null
```

wyświetli odnalezione pliki, zaś zignoruje informacje o braku praw dostępu. Jest to przydatne, gdy wynik działania programu jest niepożądany - na przykład w przypadku skryptów automatyzujących działanie serwera, wykonywanych wiele razy na dobę.

Systemu Linux dysponują także urządzeniami logicznymi, które mogą dostarczyć istotnych informacji dla skryptów, np.: /dev/random, /dev/date itp. Istnieje także możliwość bezpośredniego dostępu do „surowych" danych na dyskach oraz w pamięci. Dostęp do nich ograniczony jest wyłącznie do administratora, gdyż możliwość czytania tych danych może być zagrożeniem dla bezpieczeństwa systemu, zaś zapis może spowodować nieodwracalne uszkodzenia systemu.

8.7. Proste filtry

cat

Polecenie **cat** służy do wysłania wybranego pliku (lub kilku plików) na standardowe wyjście. Stanowi dobre narzędzie do rozpoczęcia przetwarzania strumieniowego. Może także służyć do sklejania grupy plików:

```
cat plik1 plik2 plik3 > suma_plikow
```

Ponieważ cat uruchomione bez parametrów pobiera dane ze standardowego wejścia może także służyć do wprowadzania danych z klawiatury np.:

```
cat > nowy_plik
```

spowoduje utworzenie nowego pliku w wypełnieniu go danymi wprowadzonymi z klawiatury.

head, tail

Polecenie **head** i **tail** pozwalają na wyświetlanie części pliku: odpowiednio początku i końca. Np.:

```
head 10 plik
```

wyświetli pierwszych 10 linii pliku.

Polecenie tail może spełnia szczególną funkcję po wywołaniu z parametrem '-f'. Wyświetla ono wówczas koniec pliku i oczekuje na nowe dane. Może, więc służyć jako

„monitor” pliku modyfikowanego przez inną aplikację (pracującą w tle lub na innej konsoli). Np.:

```
wget –t 0 –r l5 –o out.txt –P wp http://www.wp.pl &
tail –f out.txt
```

spowoduje pobranie portalu www.wp.pl. Pobieranie będzie realizowane w tle, a dzięki tail w dowolnej chwili można sprawdzić aktualny stan działania programu wget. Działanie tail -f można przerwać kombinacją Ctrl-C nie przerywając działania programu wget.

Posprzątajmy później po sobie :-) :

```
killall wget
rm –r wp
rm out.txt
```

more, less

Polecenie **more** i **less** pozwalają na łatwiejsze przeglądanie strumienia wyjściowego. W przypadku dużej ilości danych konsola systemu zostaje „przewinięta” i część danych zostaje utraconych. Istnieje wprawdzie możliwość cofnięcia tekstu na konsoli (shift+pgup lub suwak w xterm) jednak bufor danych jest także ograniczony. Aby móc swobodnie czytać dane zwracane przez program wywołujemy polecenie:

```
komenda | more
```

Po każdym zapełnieniu ekranu **more** zatrzyma wyświetlanie danych i będzie oczekiwał na naciśnięcie enter (przewijanie po linii) lub spacji (przewijanie po stronach). Polecenie **less** jest wygodniejsze, gdyż pozwala na swobodne przewijanie danych. Pracę z danymi wyświetlonymi przez **less** lub **more** możemy zakończyć naciskając klawisz ‘q’.

sort, uniq

Komenda **sort** służy do sortowania (domyślnie: alfabetycznego) linijek tekstu stanowiących dane wejściowe. Gdy się ją wywoła z argumentami będącymi nazwami plików, danymi do sortowania będzie zawartość tychże; w przypadku wywołania bez argumentów (nie będących opcjami, za pomocą, których można zadać bardziej złożone kryteria sortowania), komenda sort oczekuje, że dane do przetworzenia pojawią się w standardowym strumieniu wejściowym. W obu tych przypadkach, wynik sortowania pojawi się na **stdout**. Przykład:

```
cat /etc/passwd | sort
```

zwróci posortowaną listę użytkowników systemu.

Uzupełnieniem komendy sort jest komenda **uniq**. Powoduje pominięcie wierszy powtarzających się. Np.:

```
cat /etc/passwd | sort | uniq
```

Nowe wersje polecenia **sort** mają możliwość usuwania powtarzających się linii, dzięki czemu komenda **uniq** traci na znaczeniu.

tr

Polecenie to służy do usuwania lub zastępowania znaków. Kopiuje znaki ze standardowego wejścia na standardowe wyjście, zastępując po drodze lub usuwając niektóre z nich.

Opcje:
-c – (*ang. complement*) zamienia wszystkie znaki, oprócz tych, które występują w pierwszym łańcuchu (dopełnienie zbioru znaków o kodach ASCII 0 - 255)
-d – kasuje z tekstu wejściowego znaki podane w pierwszym łańcuchu
-s – jeżeli znak zawarty w drugim łańcuchu wystąpi w tekście wyjściowym kilka razy pod rząd, wielokrotność jest usuwana (wpisywany jest tylko jeden taki znak)

W nawiasach klamrowych można podać zakresy znaków, można też użyć zapisu [a*n], co oznacza n powtórzeń znaku a.

Przykłady:

```
tr ',' '\n' < dane > wynik
```

Polecenie to zastąpi wszystkie przecinki w pliku dane znakami końca linii, a wynik działania polecenia zostanie umieszczony w pliku wynik.

```
cat zapiski | tr asdkpz '[.*6]'
```

Polecenie to zastąpi litery a, s, d, k, p, z znakami kropki.

```
cat zapiski | tr –c a '[.*]'
```

Polecenie to zastąpi wszystkie znaki oprócz znaku 'a' znakami kropki.

grep

Przy przekierowaniach często używa się komendy **grep**. Komenda ta wyświetla linie pasujące (lub nie) do określonego wzorca. **grep** jest niezwykle rozbudowaną komendą, lecz zwykle używa się kilku jego podstawowych właściwości.

Uproszczona składnia:

```
grep [–v] WZORZEC [PLIK(I)]
```

gdzie:
-v – oznacza negację wzorca (czyli wzorzec nie może wystąpić)
WZORZEC – to wzór informacji do wyszukania,
PLIK(I) – lista plików do kontroli. W przypadku nie podania nazw plików, "grep" pracować będzie na stdin.

Przykłady wykorzystania:

```
ls –l | grep student
```

spowoduje wyświetlenie tylko tych linii, w których znajduje się słowo "student" .

```
cat plik.c | grep include
```

Powyższe polecenie wyświetli wszystkie linie pliku plik.c, zawierające ciąg "include"

Wzorzec programu grep stanowi wyrażenie regularne. Wyrażenia regularne są to wyrażenia wzorcowe tworzone za pomocą liter i cyfr w połączeniu ze znakami specjalnymi, które działają podobnie do operatorów. Czynią one łatwiejszymi odnajdowanie i filtrowanie informacji w plikach.

Najważniejsze operatory wyrażeń regularnych

Znak	**Opis**
.	Dopasuj dowolny znak
$	Dopasuj poprzedzające wyrażenie do końca wiersza
^	Dopasuj występujące po operatorze wyrażenie do początku wiersza
*	Dopasuj zero lub więcej wystąpień znaku poprzedzającego operator
\	Oznacza pominięcie specjalnego znaczenia znaku np.: *
[ ]	Dopasuj dowolny znak ujęty w nawiasy. np.: [abc]
[-]	Dopasuj dowolny znak z przedziału. np.: [0-9] – wszystkie cyfry; [a-z] – wszystkie małe litery; [0-9a-zA-Z] – wszystkie litery i cyfry
[^]	Dopasuj znak, który nie znajduje się w nawiasach.

Przykłady:

`grep 'Ala' plik` – znajduje wyraz Ala
`grep 'A.a' plik` – znajduje wyrazy takie jak Ala, Aga, Ara, A+a i inne
`grep 'A[lg]a' plik` – znajduje TYLKO wyrazy Ala lub Aga
`grep '^Ala' plik` – znajduje linie zaczynające się na "Ala"
`grep 'Go*gle' plik` – znajduje Gogle, Google, Gooogle itd.
`grep '[0–9][0–9]*'` – znajduje dowolny ciąg cyfr

Z wyrażeń regularnych korzystają także inne programy, np.: **ed**, **sed**, **awk** i inne.

Cześć programów (np. sed, awk) potrzebuje, aby wyrażenie regularne ujęte było w znaki "/" np.: /abc[0-9]/, inne programy (grep) przyjmują wyrażenia regularne bez dodatkowych znaków np.: *grep abc[0-9] plik*. Należy jednak pamiętać, że część stosowanych znaków może zostać zinterpretowanych przez shell. Z tego powodu bezpieczniej jest użyć cytowania.

8.8. Zaawansowane filtry

sed (edytor strumieniowy)

Edytor **sed** jest to nieinteraktywny edytor używany w skryptach do filtrowania informacji w pliku. **sed** w przeciwieństwie do innych edytorów nie modyfikuje pliku, lecz wysyła wynik operacji na standardowe wyjście.

Składnia edytora sed:

```
sed [-n] [-e polecenie edycji] [-f skrypt_z_poleceniami]
[plik_wejściowy]
```

gdzie opcje mają następujące znaczenia:
-n – nie wyświetlaj wynikowych wierszy
-e – pobierz polecenie z wiersza poleceń
-f – pobierz polecenia z pliku

Polecenie **sed** ma następującą składnię:

```
[zakres_wierszy] polecenie_edycyjne [argumenty_polecenia]
```

Zakres wierszy można podać określając numer pierwszego i ostatniego wiersza: pierwszy, ostatni. Można stosować także wyrażenia regularne: /wyr_poczatkowe/, /wyr_koncowe/. Wyrażenia regularne należy ujmować w znaki '/'.

Przykłady zakresów:

1,5
/ala/, /kota/
*/abc[0-9]/, /a.*z/*

Najważniejsze polecenia edycyjne:

a\tekst	dopisuje tekst na początku wiersza
b etykieta	przejdź do polecenia zaczynającego się :etykieta
c\tekst	zastępuje wiersze tekstem
d	usuwa wiersze
p	wyświetla wiersz
g	zastępuje bieżący wiersz zawartością bufora
h	kopiuje wiersz do bufora
i\tekst	wstawia tekst przed wybranymi wierszami
rnazwa	wysyła na wyjście plik o danej nazwie
s/wyr_reg/tekst/znaczniki	zastępuje ciąg odpowiadający wyrażeniu regularnemu na

	tekst. Znaczniki: g – zastąp wszystkie wystąpienia wzorca (brak tej opcji spowoduje zastąpienie pierwszego wystąpienia w linii) p – wyświetl wiersz po wykonaniu zastąpienia w – zapisz wiersz do pliku po wykonaniu zastąpienia n – zastąp n-te wystąpienie wzorca symbol & oznacza ciag pasujący do wyrażenia regularnego
t etykieta	przejdź do wiersza z etykietą po wykonaniu zastąpienia
wnazwa	zapisz bieżący wiersz do pliku
y/abc/xyz/	zastąp odpowiednie znaki (porównaj polecenie tr)
=	wypisz numer wiersza na wyjściu
!polecenie	wykonaj polecenie gdy wiersz nie odpowiada wzorcowi
:etykieta	etykieta
{}	grupa poleceń

Przykłady:

`sed –n '20,30p' plik > nowy_plik` – przepisuje wiersze od 20 do 30 do nowego pliku

`sed –n '/^..X/p' plik` – wyświetla wiersze w których na trzecim miejscu znajduje się ‘X’

`cat /usr/doc/bash-2.05b/README | sed '/bug/y/g/s/' | less` – poprawia błędy (zamienia w słowach “bug”: 'g' na 's')

`cat /usr/doc/bash-2.05b/README | sed '1,10s/[Bb]ug/problem/' | less` – poprawia słowa “bug” lub “Bug” na “problem” (w liniach 1-10):

`cat /usr/doc/bash-2.05b/README | sed –e 's/[^ \t\n]*@[a-zA-Z0-9_\-\.]*/[&]/' | less` – ujmuje w nawiasy wszystkie adresy mailowe

awk

awk jest językiem ogólnego przeznaczenia do przeszukiwania wzorców i przetwarzania tekstów. Można za jego pomocą wykonywać różnorodne działania związane z filtrowaniem, przekształcaniem i sporządzaniem raportów.

Polecenie awk ma następującą składnię:

`awk [-Fseparator_pól] 'program' nazwa_pliku`

```
awk [-Fseparator_pól] -f plik_z_programem nazwa_pliku
```

Format polecenia awk:

[wzorzec] [{procedura} ...]

Zarówno wzorzec jak i procedura są opcjonalne.
Wczytany plik automatycznie dzielony jest na pola, na podstawie podanego znaku oddzielającego. Do każdego z pól przypisywany jest identyfikator $1 do $n. Identyfikator $0 oznacza cały wiersz. Przykład:

```
awk -F: '{print $1, $3}' plik
```

Wzorce w awk są rozszerzeniem metody stosowanej w sed.
Przykład:

```
awk '/USA/' geo_data
awk '/^TOTAL/' geo_data
```

Istnieje możliwość wpisania wielu wyrażeń lub zakresów połączonych operatorami logicznymi:

`wzorzec && wzorzec` – operator AND
`wzorzec || wzorzec` – operator OR
`wzorzec, wzorzec` – operator zakresu
`! wzorzec` – operator NOT

Drugim typem wzorców są wyrażenia relacji. Wynik jest drukowany jeżeli spełniony jest warunek relacji. Możliwymi operatorami są:= =, <, <=, >, >=, != oraz ~ i !~ który zwraca prawdę jeżeli lewy operand zawiera (nie zawiera) wyrażenie regularne zapisane w drugim operandzie. Przykład:

```
awk '$1 == "Chapter 1"'
awk '($0 !~ /^S/ && ($1>100) print $0}' plik
```

Procedury awk mogą być zbudowane z następujących poleceń:

if (warunek) {lista_poleceń_1} else {lista_poleceń_2}	Konstrukcja warunkowa
while (warunek) {lista_poleceń_1} else {lista_poleceń_2}	Pętla WHILE
for (wyr_pocz; warunek; wyr_kon) {lista_poleceń}	Pętla FOR
break	Wyjście z pętli
continue	Przejście do następnej iteracji

next	Pominięcie sprawdzania następnego wiersza
exit	Zakończenie programu
print [wyrażenie] [>plik]	Wypisuje podane kolumny
printf format [, wyr] [>plik]	Wypisuje sformatowany tekst
zmienna = wyrażenie	operator przypisania
+, -, *, /, %, ++, --, +=, *=, /=, %=	operatory matematyczne (jak w C)

Polecenia, z których można korzystać we wzorcach:

sin(wyr), cos(wyr)	sinus i cosinus
exp(wyr)	eksponent
index(napis1, napis2)	zwraca pozycję napis1 w napis2
length(wyr)	podaje długość wyr
sprintf(format, wyr)	formatuje tekst
substr(napis, start, dlugość)	zwraca fragment napisu

Zmienne wbudowane:

FS	separator pól
RS	separator rekordów
NR	liczba wczytanych wierszy
NF	liczba pól w bieżącym wierszu
FILENAME	nazwa pliku
OFMT	format wyjściowy dla liczb
OFS	separator pól dla wyjścia
ORS	separator rekordów dla wyjścia

Zadania

1.Przekieruj strumienie wyjściowe kilku komend do pliku, a następnie przejrzyj ich zawartość

2.Uzyskaj w pliku dane na temat błędów popełnionych w wywoływanych komendach

3.Dopisz do pliku już uzyskanego **stdout** i **sdterr** na dwa sposoby

4.Wykonaj przekierowanie wszystkich 3 strumieni

5.Wygeneruj listę wszystkich plików w systemie, posortuj a następnie usuń duplikaty, zapisz to wszystko do pliku

6.Z listy, którą otrzymałeś w poprzednim ćwiczeniu wygeneruj listę plików nagłówkowych (*.h) i zapisz do pliku

7.Sprawdź parametry polecenia sort – czy pozwala ono na sortowanie według innych zasad niż „alfabetycznie”?

8.Wyszukaj w */etc/X11/xorg.conf* wszystkich sekcji „Input”
9.Wyświetl nazwy wszystkich użytkowników systemu (tylko nazwy)
10.Wyświetl nazwy procesów należących do użytkownika root (tylko nazwy)
11. Stwórz w swoim katalogu domowym plik *mojtekst* o treści:

```
hej
moj Linux
jest niezly
a teraz
maly bug
papa
```

11.W pliku *mojtekst* zamień w pierwszych 3 liniach wszystkie wystąpienia „Linux” na „GNU/Linux”
12.W pliku *mojtekst* usuń linie zawierające słowo „bug”.