O'REILLY®

Wiersz poleceń Linuksa

Rozwijanie umiejętności efektywnej pracy

Daniel Barrett

Helion

Tytuł oryginału: Efficient Linux at the Command Line: Boost Your Command-Line Skills

Tłumaczenie: Grzegorz Werner

ISBN: 978-83-283-9627-2

Authorized Polish translation of the English *Efficient Linux at the Command Line*
ISBN 9781098113407 © 2022 Daniel Barrett.

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/wiepol
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

Spis treści

ROZDZIAŁ 1.
Łączenie poleceń

Pracując w Windows, macOS i większości innych systemów operacyjnych, prawdopodobnie spędzasz czas w takich aplikacjach jak przeglądarki internetowe, edytory tekstu, arkusze kalkulacyjne i gry. Typowe aplikacje zawierają liczne funkcje: wszystko, czego zdaniem projektantów mogą potrzebować ich użytkownicy. Tak więc zwykle są samowystarczalne. Nie polegają na innych aplikacjach. Możesz od czasu do czasu kopiować i wklejać dane między aplikacjami, ale są one w dużej mierze niezależne.

Wiersz poleceń Linuksa jest inny. Zamiast dużych aplikacji z mnóstwem funkcji Linux dostarcza tysiące małych programów z bardzo nielicznymi funkcjami. Na przykład polecenie `cat` wyświetla pliki na ekranie — i to wszystko. `ls` wyświetla listę plików w katalogu, `mv` zmienia nazwy plików i tak dalej. Każde polecenie ma prosty, dość dobrze zdefiniowany cel.

A jeśli musisz zrobić coś bardziej skomplikowanego? Bez obaw. Linux pozwala łatwo łączyć polecenia tak, aby ich indywidualne funkcje *współdziałały ze sobą*. Taki styl pracy przekłada się na zupełnie inny sposób myślenia o przetwarzaniu danych. Zamiast zastanawiać się: „Którą aplikację mam uruchomić?”, aby uzyskać jakiś rezultat, zadajesz sobie pytanie: „Które polecenia powinienem połączyć?”.

W tym rozdziale dowiesz się, jak rozmieszczać i uruchamiać polecenia w różnych kombinacjach w celu osiągnięcia pożądanych wyników. Aby uprościć sprawę, przedstawię tylko sześć poleceń Linuksa i ich najbardziej podstawowe zastosowania, żebyś mógł się skupić na bardziej złożonej i interesującej kwestii — łączeniu ich — bez konieczności przyswajania sobie nadmiernej ilości informacji. To trochę jak nauka gotowania przy użyciu sześciu składników lub nauka stolarstwa za pomocą młotka i piły. (W rozdziale 5. dodam więcej poleceń do Twojego linuksowego przybornika).

Będziesz łączyć polecenia za pomocą *potoków*, funkcji Linuksa, która przekazuje wyjście jednego polecenia na wejście drugiego. Po wprowadzeniu każdego polecenia (`wc`, `head`, `cut`, `grep`, `sort` i `uniq`) od razu zademonstruję jego użycie z potokami. Niektóre przykłady znajdą praktyczne zastosowanie podczas codziennej pracy z Linuksem, podczas gdy inne posłużą tylko do zademonstrowania jakiejś ważnej funkcji.

Wejście, wyjście i potoki

Większość poleceń Linuksa czyta dane wejściowe z klawiatury, wypisuje dane wyjściowe na ekranie albo robi jedno i drugie. W Linuksie to czytanie i pisanie ma wymyślne nazwy:

stdin („standard in" — standardowe wyjście)

Strumień danych wejściowych, które Linux odczytuje z Twojej klawiatury. Kiedy wpisujesz dowolne polecenie za znakiem zachęty, dostarczasz dane do `stdin`.

stdout („standard out" — standardowe wyjście)

Strumień danych wyjściowych, które Linux wypisuje na Twoim ekranie. Kiedy wykonujesz polecenie `ls` w celu wyświetlenia nazw plików, wyniki pojawiają się na `stdout`.

A teraz clou programu. Możesz połączyć standardowe wyjście jednego polecenia ze standardowym wejściem drugiego, tak aby pierwsze dostarczało dane drugiemu. Na początek wydajmy znajome polecenie `ls -l` w celu wyświetlenia dużego katalogu, takiego jak */bin*, w długim formacie:

```
$ ls -l /bin
total 12104
-rwxr-xr-x 1 root root 1113504 cze 6  2019 bash
-rwxr-xr-x 1 root root  170456 wrz 21 2019 bsd-csh
-rwxr-xr-x 1 root root   34888 lip 4  2019 bunzip2
-rwxr-xr-x 1 root root 2062296 wrz 18 2020 busybox
-rwxr-xr-x 1 root root   34888 lip 4  2019 bzcat
⋮
-rwxr-xr-x 1 root root    5047 kwi 27 2017 znew
```

Ten katalog zawiera znacznie więcej plików, niż Twój ekran ma wierszy, więc wyniki szybko przewijają się przez ekran i znikają. Szkoda, że `ls` nie może wyświetlać informacji ekran po ekranie, zatrzymując się, dopóki nie naciśniesz klawisza, żeby kontynuować. Ale chwileczkę: funkcję tę oferuje inne polecenie Linuksa. `less` wyświetla pliki ekran po ekranie:

```
$ less mojplik
```

Możesz połączyć te dwa polecenia, ponieważ `ls` wypisuje dane na `stdout`, a `less` może czytać ze `stdin`. Użyj potoku, aby przekazać wyjście `ls` na wejście `less`:

```
$ ls -l /bin | less
```

To połączone polecenie wyświetla zawartość ekranu ekran po ekranie. Pionowa kreska (`|`) między poleceniami jest linuksowym symbolem potoku[1]. Łączy on wyjście `stdout` pierwszego polecenia z wejściem `stdin` następnego. Każdy wiersz polecenia, który zawiera symbole potoku, będziemy nazywać *rurociągiem* (ang. *pipeline*).

Polecenia zwykle nie są „świadome", że stanowią część rurociągu. `ls` uważa, że pisze na ekranie, choć tak naprawdę jego dane wyjściowe są przekierowywane do `less`. A `less` myśli, że czyta z klawiatury, podczas gdy w rzeczywistości otrzymuje dane od `ls`.

[1] Na standardowej klawiaturze symbol potoku znajduje się na tym samym klawiszu co ukośnik odwrotny (\), zwykle między klawiszami *Enter* i *Backspace* albo między lewym klawiszem *Shift* a *Z*.

Co to jest polecenie?

Słowo *polecenie* ma w Linuksie trzy różne znaczenia, jak pokazano na rysunku 1.1:

Program

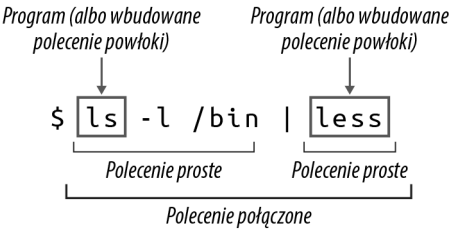
Program wykonywalny, nazwany i uruchamiany jednym słowem, takim jak `ls`, albo podobna funkcja wbudowana w powłokę (zwana *wbudowanym poleceniem powłoki*)[2], na przykład `cd`.

Polecenie proste

Nazwa programu (lub wbudowanego polecenia powłoki), po której opcjonalnie następują argumenty, na przykład `ls -l /bin`.

Polecenie połączone

Kilka prostych poleceń traktowanych jako całość, na przykład rurociąg `ls -l /bin | less`.



Rysunek 1.1. Mianem „polecenia" określa się programy, polecenia proste i polecenia połączone

W niniejszej książce używam słowa *polecenie* we wszystkich tych znaczeniach. To, które z nich mam na myśli, zwykle wynika z kontekstu, ale jeśli będzie inaczej, posłużę się jednym z precyzyjniejszych terminów.

Sześć poleceń na dobry początek

Potoki są kluczowym narzędziem w przyborniku linuksowego eksperta. Zacznijmy zatem budować Twoje umiejętności od niewielkiego zbioru poleceń Linuksa. W ten sposób bez względu na to, jakie polecenia napotkasz w przyszłości, będziesz umiał je łączyć.

Każde z sześciu poleceń — `wc`, `head`, `cut`, `grep`, `sort` i `uniq` — ma liczne opcje i tryby działania, które teraz w dużej mierze pominę, aby skupić się na potokach. Aby dowiedzieć się więcej o jakimkolwiek z nich, wykonaj polecenie `man` w celu wyświetlenia pełnej dokumentacji. Na przykład:

```
$ man wc
```

Aby zademonstrować sześć naszych poleceń w akcji, będę używał pliku o nazwie *zwierzeta.txt*, który zawiera informacje o kilku książkach wydawnictwa O'Reilly, pokazanego na listingu 1.1.

[2] W standardzie POSIX tę formę polecenia określa się mianem *udogodnienia* (ang. *utility*).

Listing 1.1. Zawartość pliku zwierzeta.txt

```
pyton     Python. Leksykon kieszonkowy   2014   Lutz, Mark
sowa      Wyrażenia regularne            2005   Friedl, Jeffrey
lama      Perl. Wprowadzenie             2006   Schwartz, Randal
krab      Nauka algorytmów               2022   Heineman, George
suhak     Wydajność Javy                 2020   Oaks, Scott
krogulec  Raspberry Pi. Receptury        2020   Monk, Simon
dzik      React od podstaw               2021   Porcello, Eve
```

Każdy wiersz zawiera cztery fakty na temat książki O'Reilly, oddzielone znakami tabulacji: zwierzę na przedniej okładce, tytuł książki, rok wydania oraz nazwisko i imię pierwszego autora.

Polecenie 1.: wc

Polecenie `wc` wyświetla liczbę wierszy, słów i znaków w pliku:

```
$ wc zwierzeta.txt
7 45 319 zwierzeta.txt
```

`wc` informuje, że plik *zwierzeta.txt* ma 7 wierszy, 45 słów i 319 znaków.

Jeśli sam policzysz znaki, łącznie ze spacjami i znakami tabulacji, przekonasz się, że jest ich tylko 312, ale `wc` dolicza niewidoczny znak nowego wiersza na końcu każdej linijki.

Opcje `-l`, `-w` oraz `-c` nakazują poleceniu `wc` wyświetlić odpowiednio tylko liczbę wierszy, słów i znaków:

```
$ wc -l zwierzeta.txt
7 zwierzeta.txt
$ wc -w zwierzeta.txt
45 zwierzeta.txt
$ wc -c zwierzeta.txt
319 zwierzeta.txt
```

Liczenie jest tak przydatnym, uniwersalnym zadaniem, że twórcy `wc` zaprojektowali polecenie tak, aby mogło pracować z potokami. Jeśli pominiesz nazwę pliku, będzie czytać dane z `stdin` i zapisywać je na `stdout`. Użyjmy polecenia `ls`, aby wyświetlić zawartość bieżącego katalogu, i przekierujmy jego wyjście do `wc` w celu policzenia wierszy. Ten rurociąg odpowiada na pytanie: „Ile plików jest widocznych w moim bieżącym katalogu?".

```
$ ls -1
mojplik
mojplik2
test.py
zwierzeta.txt
$ ls -1 | wc -l
4
```

Opcja `-1`, która nakazuje poleceniu `ls` wyświetlić wyniki w jednej kolumnie, nie jest tu absolutnie konieczna. Aby dowiedzieć się, czemu jej użyłem, zajrzyj do ramki „Polecenie ls działa inaczej, kiedy jest przekierowane" dalej w tym rozdziale.

`wc` to pierwsze polecenie omawiane w tym rozdziale, więc jesteśmy nieco ograniczeni, jeśli chodzi o to, co możemy robić z potokami. Dla zabawy spróbujmy przekierować wyjście polecenia `wc` do niego samego, co pokaże, że to samo polecenie może występować w rurociągu więcej niż jeden raz.

Poniższe połączone polecenie informuje, że liczba słów na wyjściu wc wynosi cztery (trzy liczby całkowite i jedna nazwa pliku):

```
$ wc zwierzeta.txt
  7  51 327 zwierzeta.txt
$ wc zwierzeta.txt | wc -w
4
```

Dlaczego miałbyś na tym poprzestać? Dodaj trzecie polecenie wc do rurociągu i zlicz wiersze, słowa oraz znaki w łańcuchu „4”:

```
$ wc zwierzeta.txt | wc -w | wc
      1       1       2
```

Wyniki wskazują jeden wiersz (zawierający cyfrę 4), jedno słowo (samą cyfrę 4) oraz dwa znaki. Czemu dwa? Ponieważ wiersz „4” kończy się niewidocznym znakiem nowego wiersza.

Wystarczy już tych niemądrych rurociągów z wc. Im więcej znasz poleceń, tym praktyczniejsze stają się potoki.

Polecenie ls działa inaczej, kiedy jest przekierowane

W przeciwieństwie do praktycznie wszystkich innych poleceń Linuksa, ls rozpoznaje, czy jego wyjściem stdout jest ekran, czy też polecenie zostało przekierowane (do potoku lub gdzie indziej). Powodem jest przyjazność dla użytkownika. Kiedy wyjściem stdout jest ekran, ls rozmieszcza wyniki w wielu kolumnach, żeby były bardziej czytelne:

```
$ ls /bin
bash       dir      kmod     networkctl     red      tar
bsd-csh    dmesg    less     nisdomainname  rm       tempfile
⋮
```

Kiedy jednak wyjście stdout jest przekierowane, ls wyświetla tylko jedną kolumnę. Zademonstruję to, przekierowując wyjście ls do polecenia, które po prostu powiela swoje dane wejściowe, takiego jak cat[3]:

```
$ ls /bin | cat
bash
bsd-csh
bunzip2
busybox
⋮
```

Ten sposób działania może prowadzić do dziwnych wyników, jak w poniższym przykładzie:

```
$ ls
mojplik  mojplik2  test.py  zwierzeta.txt
$ ls | wc -l
4
```

Pierwsze polecenie ls wyświetla wszystkie nazwy plików w jednym wierszu, ale drugie informuje, że ls utworzyło cztery wiersze. Jeśli nie znasz tej specyfiki ls, rozbieżność może być dezorientująca.

Polecenie ls ma opcje, które zmieniają jego domyślne działanie. Możesz wymusić wyświetlanie wyników w jednej kolumnie za pomocą opcji -1 albo w wielu kolumnach przy użyciu opcji -C.

[3] W zależności od Twojej konfiguracji polecenie ls może również używać innych funkcji formatowania, takich jak kolor, kiedy wyświetla dane na ekranie, ale nie, kiedy jest przekierowane.

Polecenie 2.: head

Polecenie `head` wyświetla kilka pierwszych wierszy pliku. Wypisz trzy pierwsze wiersze pliku *zwierzeta.txt* za pomocą polecenia `head` z opcją `-n`:

```
$ head -n3 zwierzeta.txt
pyton    Python. Leksykon kieszonkowy   2014   Lutz, Mark
sowa     Wyrażenia regularne            2005   Friedl, Jeffrey
lama     Perl. Wprowadzenie             2006   Schwartz, Randal
```

Jeśli zażądasz więcej wierszy, niż znajduje się w pliku, `head` wyświetli cały plik (jak `cat`). Jeśli pominiesz opcję `-n`, `head` domyślnie wyświetli 10 wierszy (`-n10`).

Samo w sobie polecenie `head` przydaje się do zaglądania na początek pliku, kiedy nie interesuje Cię reszta jego zawartości. Jest szybkie i efektywne, nawet w przypadku bardzo dużych plików, ponieważ nie musi odczytywać całego pliku. Ponadto `head` wypisuje wyniki na `stdout`, więc jest przydatne w rurociągach. Sprawdźmy liczbę słów w trzech pierwszych wierszach pliku *zwierzeta.txt*:

```
$ head -n3 zwierzeta.txt | wc -w
19
```

Polecenie `head` może również czytać dane z `stdin`. Często używa się go do ograniczenia danych wyjściowych innego polecenia, kiedy nie chcesz oglądać ich w całości, jak w przypadku długiego listingu katalogu. Wyświetlmy na przykład nazwy pierwszych pięciu plików w katalogu */bin*:

```
$ ls /bin | head -n5
bash
bsd-csh
bunzip2
busybox
bzcat
```

Polecenie 3.: cut

Polecenie `cut` wyświetla jedną lub więcej kolumn pliku. Wypiszmy na przykład wszystkie tytuły książek z pliku *zwierzeta.txt*, które znajdują się w drugiej kolumnie:

```
$ cut -f2 zwierzeta.txt
Python. Leksykon kieszonkowy
Wyrażenia regularne
Perl. Wprowadzenie
Nauka algorytmów
Wydajność Javy
Raspberry Pi. Receptury
React od podstaw
```

Polecenie `cut` pozwala zdefiniować na dwa sposoby, co jest „kolumną”. Pierwszym jest cięcie pliku według pola (`-f`), kiedy dane wejściowe składają się z łańcuchów (pól) oddzielonych pojedynczym znakiem tabulacji. Tak się składa, że dokładnie taki format ma plik *zwierzeta.txt*. Poprzednie polecenie wyświetla drugie pole każdego wiersza dzięki opcji `-f2`.

Aby skrócić wyniki, przekieruj je do polecenia `head` i wyświetl tylko trzy pierwsze wiersze:

```
$ cut -f2 zwierzeta.txt | head -n3
Python. Leksykon kieszonkowy
Wyrażenia regularne
Perl. Wprowadzenie
```

Możesz też wyciąć wiele pól albo poprzez oddzielenie ich numerów przecinkami:

```
$ cut -f1,3 zwierzeta.txt | head -n3
pyton   2014
sowa    2005
lama    2006
```

albo poprzez podanie zakresu liczbowego:

```
$ cut -f2-4 zwierzeta.txt | head -n3
Python. Leksykon kieszonkowy  2014   Lutz, Mark
Wyrażenia regularne           2005   Friedl, Jeffrey
Perl. Wprowadzenie            2006   Schwartz, Randal
```

Drugim sposobem definiowania „kolumny" do wycięcia jest określenie pozycji znaku za pomocą opcji `-c`. Wypiszmy trzy pierwsze znaki z każdego wiersza pliku, co możemy określić albo z wykorzystaniem przecinków (`1,2,3`), albo za pomocą zakresu (`1-3`):

```
$ cut -c1-3 zwierzeta.txt
pyt
sow
lam
kra
suh
kro
dzi
```

Teraz, kiedy znasz już podstawowe funkcje, spróbuj zrobić coś bardziej praktycznego z wykorzystaniem polecenia `cut` i potoków. Wyobraź sobie, że plik *zwierzeta.txt* liczy tysiące wierszy, a Ty chcesz wyodrębnić tylko nazwiska autorów. Najpierw wyizoluj czwarte pole, które zawiera nazwisko i imię:

```
$ cut -f4 zwierzeta.txt
Lutz, Mark
Friedl, Jeffrey
Schwartz, Randal
⋮
```

Następnie ponownie przekieruj wyniki do `cut`, używając opcji `-d` (od „delimiter" — ogranicznik), aby zmienić separator ze znaku tabulacji na przecinek i wyodrębnić samo nazwisko:

```
$ cut -f4 zwierzeta.txt | cut -d, -f1
Lutz
Friedl
Schwartz
⋮
```

Oszczędź czas dzięki historii i edytowaniu poleceń

Czy wielokrotnie wpisujesz te same polecenia? Zamiast tego naciśnij kilkakrotnie klawisz strzałki w górę, aby przewinąć listę poleceń, które wydałeś wcześniej (tę funkcję powłoki określa się mianem *historii poleceń*). Kiedy dotrzesz do żądanego polecenia, naciśnij *Enter*, aby wykonać je natychmiast, albo najpierw je zmodyfikuj, używając klawiszy strzałki w lewo i w prawo w celu umieszczenia kursora w odpowiednim miejscu oraz klawisza *Backspace* w celu usunięcia znaków (ta funkcja nosi nazwę *edytowania wiersza polecenia*).

W rozdziale 3. omówię znacznie bardziej zaawansowane funkcje historii i edytowania poleceń.

Polecenie 4.: grep

grep to niezwykle zaawansowane polecenie, ale na razie ukryję większość jego możliwości i powiem po prostu, że wyświetla ono wiersze, które pasują do danego łańcucha (więcej szczegółów znajdziesz w rozdziale 5.). Na przykład poniższe polecenie wyświetla wiersze z pliku *zwierzeta.txt*, które zawierają łańcuch Receptury:

```
$ grep Receptury zwierzeta.txt
krogulec Raspberry Pi. Receptury      2020   Monk, Simon
```

Możesz też wypisać wiersze, które *nie pasują* do danego łańcucha, używając opcji -v. Zauważ, że wiersz ze słowem „Receptury" jest nieobecny:

```
$ grep -v Receptury zwierzeta.txt
pyton    Python. Leksykon kieszonkowy  2014   Lutz, Mark
sowa     Wyrażenia regularne           2005   Friedl, Jeffrey
lama     Perl. Wprowadzenie            2006   Schwartz, Randal
krab     Nauka algorytmów              2022   Heineman, George
suhak    Wydajność Javy                2020   Oaks, Scott
dzik     React od podstaw              2021   Porcello, Eve
```

Ogólnie rzecz biorąc, polecenie grep przydaje się do wyszukiwania tekstu w zbiorze plików. Poniższe polecenie wypisuje wiersze, które zawierają łańcuch Perl, w plikach, których nazwy kończą się na .txt:

```
$ grep Perl *.txt
zwierzeta.txt:lama    Perl. Wprowadzenie    2006   Schwartz, Randal
esej.txt:naprawdę uwielbiam język programowania Perl, który
esej.txt:języki takie jak Perl, Python, PHP i Ruby w równym
```

W tym przypadku polecenie grep znalazło trzy pasujące wiersze, jeden w pliku *zwierzeta.txt* i dwa w pliku *esej.txt*. Polecenie grep czyta z stdin i pisze na stdout, więc działa świetnie w rurociągach. Przypuśćmy, że chcesz się dowiedzieć, ile jest podkatalogów w dużym katalogu */usr/lib*. Nie ma jednego polecenia Linuksa, które odpowiadałoby na to pytanie, więc skonstruuj rurociąg. Zacznij od polecenia ls -l:

```
$ ls -l /usr/lib
drwxr-xr-x   2 root root   4096 lut 27  2019 accountsservice
drwxr-xr-x   3 root root   4096 lut 27  2019 aisleriot
drwxr-xr-x   2 root root   4096 lut 27  2019 apg
-rwxr-xr-x   1 root root   3283 cze 20  2016 command-not-found
-rw-r--r--   1 root root    298 lut 20  2019 os-release
⋮
```

Zauważ, że ls -l oznacza katalogi literą d na początku wiersza. Użyj polecenia cut, aby wyizolować pierwszą kolumnę, która może, ale nie musi zawierać litery d:

```
$ ls -l /usr/lib | cut -c1 d
d
d
d
-
-
⋮
```

Następnie użyj polecenia grep, aby zachować tylko wiersze z literą d:

```
$ ls -l /usr/lib | cut -c1 | grep d
d
d
d
⋮
```

Wreszcie, zlicz wiersze poleceniem wc, a rurociąg złożony z czterech poleceń udzieli poszukiwanej odpowiedzi — */usr/lib* zawiera 138 podkatalogów:

```
$ ls -l /usr/lib | cut -c1 | grep d | wc -l
138
```

Polecenie 5.: sort

Polecenie sort układa wiersze pliku w porządku rosnącym (domyślnie):

```
$ sort zwierzeta.txt
dzik     React od podstaw             2021   Porcello, Eve
krab     Nauka algorytmów             2022   Heineman, George
krogulec Raspberry Pi. Receptury      2020   Monk, Simon
lama     Perl. Wprowadzenie           2006   Schwartz, Randal
pyton    Python. Leksykon kieszonkowy  2014   Lutz, Mark
sowa     Wyrażenia regularne          2005   Friedl, Jeffrey
suhak    Wydajność Javy               2020   Oaks, Scott
```

albo malejącym (z opcją -r):

```
$ sort -r zwierzeta.txt
suhak    Wydajność Javy               2020   Oaks, Scott
sowa     Wyrażenia regularne          2005   Friedl, Jeffrey
pyton    Python. Leksykon kieszonkowy  2014   Lutz, Mark
lama     Perl. Wprowadzenie           2006   Schwartz, Randal
krogulec Raspberry Pi. Receptury      2020   Monk, Simon
krab     Nauka algorytmów             2022   Heineman, George
dzik     React od podstaw             2021   Porcello, Eve
```

sort może porządkować wiersze alfabetycznie (domyślnie) albo numerycznie (z opcją -n). Zademonstruję to za pomocą rurociągów, które wycinają z pliku *zwierzeta.txt* trzecie pole, czyli rok wydania:

```
$ cut -f3 zwierzeta.txt                    Nieposortowane
2014
2005
2006
2022
2020
2020
2021
$ cut -f3 zwierzeta.txt | sort -n          Rosnąco
2005
2006
2014
2020
2020
2021
2022
$ cut -f3 zwierzeta.txt | sort -nr         Malejąco
2022
2021
```

```
2020
2020
2014
2006
2005
```

Aby znaleźć najnowszy rok wydania książki w pliku *zwierzeta.txt*, przekieruj wyjście `sort` na wejście `head` i wypisz tylko pierwszy wiersz:

```
$ cut -f3 zwierzeta.txt | sort -nr | head -n1
2022
```

Wartości maksymalne i minimalne

Polecenia `sort` i `head` są bardzo użyteczne podczas pracy z danymi liczbowymi, kiedy w każdym wierszu znajduje się jedna wartość. Możesz wyświetlić wartość maksymalną poprzez przekierowanie danych do polecenia:

```
... | sort -nr | head -n1
```

oraz wartość minimalną za pomocą polecenia:

```
... | sort -n | head -n1
```

W kolejnym przykładzie pomajstrujmy przy pliku */etc/passwd*, w którym znajduje się lista użytkowników mogących wykonywać procesy w systemie[4]. Wygeneruj listę wszystkich użytkowników w porządku alfabetycznym. W pierwszych pięciu wierszach zobaczysz coś w rodzaju:

```
$ head -n5 /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
kowalska:x:1000:1000:Anna Kowalska,,,:/home/kowalska:/bin/bash
nowak:x:1001:1001:Bartosz Nowak,,,:/home/nowak:/bin/bash
```

Każdy wiersz składa się z łańcuchów oddzielonych dwukropkami, a pierwszym łańcuchem jest nazwa użytkownika, więc możesz wyizolować nazwy użytkowników za pomocą polecenia `cut`:

```
$ head -n5 /etc/passwd | cut -d: -f1
root
daemon
bin
kowalska
nowak
```

i posortować je:

```
$ head -n5 /etc/passwd | cut -d: -f1 | sort
bin
daemon
kowalska
nowak
root
```

Jeśli wolałbyś wyświetlić posortowaną listę wszystkich nazw użytkowników, a nie tylko pierwszych pięciu, zamiast `head` użyj `cat`:

```
$ cat /etc/passwd | cut -d: -f1 | sort
```

[4] W niektórych systemach linuksowych informacje o użytkownikach są przechowywane gdzie indziej.

Aby sprawdzić, czy dany użytkownik ma konto w Twoim systemie, dopasuj jego nazwę użytkownika za pomocą polecenia `grep`. Wynik pusty oznacza, że nie ma takiego konta:

```
$ cut -d: -f1 /etc/passwd | grep -w nowak
nowak
$ cut -d: -f1 /etc/passwd | grep -w brukiew          (brak wyników)
```

Opcja `-w` nakazuje poleceniu `grep` dopasowywać tylko całe słowa, a nie ich części, na wypadek gdyby w Twoim systemie była nazwa użytkownika zawierająca łańcuch „nowak", na przykład `kasianowak2`.

Polecenie 6.: uniq

Polecenie `uniq` wykrywa powtarzające się przyległe wiersze w pliku i domyślnie usuwa powtórzenia. Zademonstruję je na przykładzie prostego pliku, który zawiera wielkie litery:

```
$ cat litery
A
A
A
B
B
A
C
C
C
C
$ uniq litery
A
B
A
C
```

Zauważ, że polecenie `uniq` zredukowało trzy pierwsze wiersze z `A` do jednego, ale pozostawiło ostatnie `A` na miejscu, ponieważ *nie sąsiadowało* ono z pierwszymi trzema.

Możesz też zliczyć wystąpienia za pomocą opcji `-c`:

```
$ uniq -c litery
3 A
2 B
1 A
4 C
```

Przyznam, że kiedy pierwszy raz napotkałem polecenie `uniq`, nie wydawało mi się szczególnie użyteczne, ale szybko stało się jednym z moich ulubionych. Przypuśćmy, że masz rozdzielony znakami tabulacji plik z końcowymi ocenami z kursu uniwersyteckiego w skali od 6 (najlepsza) do 1 (najgorsza):

```
$ cat oceny
5    Gerda
4    Celina
6    Krystyna
6    Sara
4    Hanna
5    Leszek
5    Eligiusz
```

```
5    Emilia
6    Oliwia
3    Norbert
1    Anna
```

Chcesz wypisać ocenę, która występuje najczęściej (w przypadku remisu wystarczy wypisać jedną). Najpierw wyizoluj oceny za pomocą polecenia `cut` i posortuj je:

```
$ cut -f1 oceny | sort
1
3
4
4
5
5
5
5
6
6
6
```

W dalszej kolejności użyj `uniq`, aby zliczyć jednakowe sąsiadujące wiersze:

```
$ cut -f1 oceny | sort | uniq -c
      1 1
      1 3
      2 4
      4 5
      3 6
```

Następnie posortuj wiersze liczbowo w odwrotnej kolejności, aby przenieść najczęściej występującą ocenę do górnego wiersza:

```
$ cut -f1 oceny | sort | uniq -c | sort -nr
      4 5
      3 6
      2 4
      1 3
      1 1
```

i użyj polecenia `head`, aby zachować tylko pierwszy wiersz:

```
$ cut -f1 oceny | sort | uniq -c | sort -nr | head -n1
      4 5
```

Na koniec, ponieważ interesuje Cię tylko najczęściej występująca ocena, a nie konkretna liczba jej wystąpień, wyodrębnij ocenę poleceniem `cut`:

```
$ cut -f1 oceny | sort | uniq -c | sort -nr | head -n1 | cut -c9
5
```

Oto odpowiedź uzyskana za pomocą rurociągu złożonego z sześciu poleceń — jak dotychczas naszego najdłuższego. Budowanie rurociągu krok po kroku nie jest wyłącznie ćwiczeniem edukacyjnym. Właśnie tak pracują linuksowi eksperci. Technice tej poświęcony jest rozdział 8.

Wykrywanie zduplikowanych plików

Połączmy to, czego nauczyłeś się do tej pory, z większym przykładem. Przypuśćmy, że jesteś w katalogu pełnym plików JPEG i chcesz dowiedzieć się, czy niektóre z nich są duplikatami:

```
$ ls
obraz001.jpg  obraz005.jpg  obraz009.jpg  obraz013.jpg  obraz017.jpg
obraz002.jpg  obraz006.jpg  obraz010.jpg  obraz014.jpg  obraz018.jpg
⋮
```

Możesz uzyskać odpowiedź za pomocą rurociągu. Potrzebne będzie inne polecenie, `md5sum`, które bada zawartość pliku i oblicza 32-znakowy łańcuch zwany *sumą kontrolną*:

```
$ md5sum obraz001.jpg
146b163929b6533f02e91bdf21cb9563  obraz001.jpg
```

Z przyczyn matematycznych istnieje bardzo, bardzo duże prawdopodobieństwo, że suma kontrolna danego pliku będzie niepowtarzalna. Jeśli dwa pliki mają tę samą sumę kontrolną, to niemal na pewno są duplikatami. Poniższe polecenie `md5sum` wskazuje, że pierwszy i trzeci plik są duplikatami:

```
$ md5sum obraz001.jpg obraz002.jpg obraz003.jpg
146b163929b6533f02e91bdf21cb9563  obraz001.jpg
63da88b3ddde0843c94269638dfa6958  obraz002.jpg
146b163929b6533f02e91bdf21cb9563  obraz003.jpg
```

Zduplikowane sumy kontrolne łatwo wykryć „na oko”, kiedy są tylko trzy pliki, ale gdybyś miał ich trzy tysiące? Z pomocą przychodzą potoki. Oblicz wszystkie sumy kontrolne, użyj polecenia `cut`, aby wyodrębnić pierwsze 32 znaki każdego wiersza, i posortuj wiersze, żeby wszystkie duplikaty sąsiadowały ze sobą:

```
$ md5sum *.jpg | cut -c1-32 | sort
1258012d57050ef6005739d0e6f6a257
146b163929b6533f02e91bdf21cb9563
146b163929b6533f02e91bdf21cb9563
17f339ed03733f402f74cf386209aeb3
⋮
```

Teraz dodaj polecenie `uniq`, aby policzyć powtórzone wiersze:

```
$ md5sum *.jpg | cut -c1-32 | sort | uniq -c
      1 1258012d57050ef6005739d0e6f6a257
      2 146b163929b6533f02e91bdf21cb9563
      1 17f339ed03733f402f74cf386209aeb3
      ⋮
```

Jeśli nie ma żadnych duplikatów, wszystkie sumy utworzone przez `uniq` będą równe 1. Posortuj wyniki liczbowo od najwyższych do najniższych, a wszystkie sumy większe niż 1 pojawią się na początku listingu:

```
$ md5sum *.jpg | cut -c1-32 | sort | uniq -c | sort -nr
      3 f6464ed766daca87ba407aede21c8fcc
      2 c7978522c58425f6af3f095ef1de1cd5
      2 146b163929b6533f02e91bdf21cb9563
      1 d8ad913044a51408ec1ed8a204ea9502
      ⋮
```

Teraz usuńmy wiersze, które nie są zduplikowane. Ich sumy kontrolne poprzedza sześć spacji, cyfra 1 oraz pojedyncza spacja. Użyjemy polecenia `grep -v` w celu usunięcia tych wierszy[5]:

```
$ md5sum *.jpg | cut -c1-32 | sort | uniq -c | sort -nr | grep -v "      1 "
      3 f6464ed766daca87ba407aede21c8fcc
      2 c7978522c58425f6af3f095ef1de1cd5
      2 146b163929b6533f02e91bdf21cb9563
```

W ten sposób nareszcie otrzymujesz listę zduplikowanych sum kontrolnych posortowaną według liczby wystąpień, utworzoną przez przepiękny rurociąg złożony z sześciu poleceń. Jeśli nie wyświetli on żadnych wyników, oznacza to, że nie ma zduplikowanych plików.

Polecenie to byłoby jeszcze bardziej przydatne, gdyby wyświetlało nazwy zduplikowanych plików, ale ta operacja wymaga funkcji, których dotychczas nie omówiliśmy (poznasz je w podpunkcie „Ulepszanie detektora zduplikowanych plików" w rozdziale 5.). Tymczasem możesz zidentyfikować pliki, które mają określoną sumę kontrolną, wyszukując je poleceniem `grep`:

```
$ md5sum *.jpg | grep 146b163929b6533f02e91bdf21cb9563
146b163929b6533f02e91bdf21cb9563  obraz001.jpg
146b163929b6533f02e91bdf21cb9563  obraz003.jpg
```

a następnie „czyszcząc" wyniki poleceniem `cut`:

```
$ md5sum *.jpg | grep 146b163929b6533f02e91bdf21cb9563 | cut -c35-
obraz001.jpg
obraz003.jpg
```

Podsumowanie

Poznałeś potęgę `stdin`, `stdout` i potoków. Zmieniają one garstkę poleceń w zbiór modularnych narzędzi, dowodząc, że całość jest czymś więcej niż tylko sumą części. *Każde* polecenie, które czyta z `stdin` albo pisze na `stdout`, może być częścią rurociągu[6]. W miarę poznawania kolejnych poleceń będziesz mógł wykorzystać ogólne koncepcje przedstawione w tym rozdziale, aby tworzyć własne użyteczne kombinacje.

[5] Z technicznego punktu widzenia w tym rurociągu nie trzeba używać ostatniego polecenia `sort -nr`, aby wyodrębnić duplikaty, ponieważ polecenie `grep` usunie wszystkie wiersze, które nie są zduplikowane.

[6] Niektóre polecenia nie używają `stdin`/`stdout`, więc nie mogą czytać z potoków ani do nich zapisywać. Przykładem mogą być polecenia `mv` i `rm`. W rurociągach można jednak używać tych poleceń w inny sposób; przykłady znajdziesz w rozdziale 8.

Skorowidz

L

M

N

O

P

R

S

Ś

T

U

V

W

Z

Wiersz poleceń Linuksa: szybciej, inteligentniej, wydajniej!

Linux od dawna jest chętnie używanym systemem operacyjnym, na jego bazie powstało wiele dystrybucji odpowiadających zróżnicowanym potrzebom. Na pierwszy rzut oka może się wydawać, że większość zadań administracyjnych w Linuksie najwygodniej wykonywać za pomocą myszy. Wiersz poleceń jest o wiele trudniejszym interfejsem: na ekranie wyświetla się znak zachęty, a system oczekuje na wpisanie poprawnie zbudowanego polecenia. Taki sposób pracy jednak pozwala na uzyskanie dużo wyższej efektywności niż interfejs graficzny.

Tę książkę docenią administratorzy systemów, projektanci, inżynierowie aplikacji i entuzjaści Linuksa. Dzięki niej dowiesz się, jak tworzyć i uruchamiać złożone polecenia rozwiązujące rzeczywiste problemy, przetwarzające i pobierające informacje, a także automatyzujące zadania, które dotychczas trzeba było wykonywać ręcznie. Poznasz procesy zachodzące w komputerze po wykonaniu polecenia w wierszu poleceń i opanujesz kilkanaście różnych technik uruchamiania poleceń. Poszczególne zagadnienia zilustrowano przejrzystymi przykładami kodu. W książce znalazły się również bezcenne informacje o metodach uzyskiwania najwyższej skuteczności i efektywności działania. Nie zabrakło praktycznych wskazówek, instrukcji i trików ułatwiających takie zadania jak zarządzanie hasłami, łatwe nawigowanie po systemie plików czy przetwarzanie danych internetowych bez odrywania rąk od klawiatury.

Najciekawsze zagadnienia:

- polecenia, dzięki którym szybciej wykonasz czasochłonne zadania
- łatwa nawigacja po systemie plików Linuksa
- budowa zaawansowanych poleceń z prostszych elementów
- przekształcanie plików tekstowych
- analiza danych w plikach tekstowych
- korzystanie z funkcji zwykle aktywowanych za pomocą myszy

Daniel Barrett od ponad 30 lat zajmuje się Linuksem i powiązanymi technologiami, jest autorem licznych książek na ten temat. Obecnie pracuje w firmie Google. W przeszłości bywał inżynierem oprogramowania, administratorem systemu, wykładowcą uniwersyteckim, projektantem witryn internetowych, wokalistą i humorystą.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-283-9627-2
9 788328 396272
Cena: 59,00 zł