Wiersz poleceń Linuksa
dla początkujacych

Tłumaczenie i opracowanie na podstawie samouczka ze strony
`https://ubuntu.com/tutorials/command-line-for-beginners`

Andrzej Zbrzezny

Październik 2023

1 Przegląd

Wiersz poleceń systemu Linux jest tekstowym interfejsem do naszego komputera. Często określany jako powłoka, terminal, konsola, zachęta lub różne inne nazwy, może sprawiać wrażenie skomplikowanego i mylącego w użyciu. Jednak możliwość kopiowania i wklejania poleceń ze strony internetowej, w połączeniu z mocą i elastycznością, jaką oferuje wiersz poleceń, oznacza, że korzystanie z niego może być niezbędne, gdy próbujemy podążać za instrukcjami online.

Ten samouczek przedstawia trochę historii wiersza poleceń, a następnie przeprowadza przez kilka praktycznych ćwiczeń, aby można było zapoznać się z kilkoma podstawowymi poleceniami i koncepcjami.

Czego się nauczymy

- Trochę historii wiersza poleceń
- Jak uzyskać dostęp do wiersza poleceń z własnego komputera
- Jak wykonać kilka podstawowych manipulacji na plikach
- Kilka innych przydatnych poleceń
- Jak łączyć polecenia razem, aby stworzyć potężniejsze narzędzia
- Najlepszy sposób na wykorzystanie uprawnień administratora

Czego będziemy potrzebować

- Komputera z systemem Ubuntu lub inną wersją Linuksa
- Chęci do nauki 

Każdy system Linux zawiera wiersz poleceń tego czy innego rodzaju. Ten samouczek zawiera kilka konkretnych kroków dla systemu Ubuntu, ale większość treści powinna działać niezależnie od dystrybucji Linuksa.

2 Krótka lekcja historii

W latach powstawania przemysłu komputerowego jeden z wczesnych systemów operacyjnych nosił nazwę Unix. Został on zaprojektowany do pracy jako system wielostanowiskowy na komputerach mainframe, z których użytkownicy łączyli się zdalnie za pomocą indywidualnych terminali. Terminale te były dość podstawowe jak na współczesne standardy: tylko klawiatura i ekran, bez możliwości lokalnego uruchamiania programów. Zamiast tego wysyłały tylko naciśnięcia klawiszy do serwera i wyświetlały na ekranie wszelkie otrzymane dane. Nie było myszki, wymyślnych grafik, ani nawet możliwości wyboru koloru. Wszystko było wysyłane jako tekst i odbierane jako tekst. Oczywiście, każdy program działający na mainframe musiał produkować tekst jako wyjście i akceptować tekst jako wejście.

W porównaniu z grafiką, tekst jest bardzo mało wymagający. Nawet na maszynach z lat siedemdziesiątych, obsługujących setki terminali przez bardzo powolne (jak na dzisiejsze standardy) połączenia sieciowe, użytkownicy nadal mogli szybko i sprawnie wchodzić w interakcje z programami. Polecenia były również bardzo krótkie, aby zredukować liczbę potrzebnych naciśnięć klawiszy, co jeszcze bardziej przyspieszało korzystanie z terminala. Ta szybkość i wydajność jest jednym z powodów, dla których ten interfejs tekstowy jest nadal szeroko stosowany dzisiaj.

Kiedy użytkownik logował się do mainframe'a Unixa przez terminal, nadal musiał wykonywać zadania związane z zarządzaniem plikami, które teraz można wykonywać za pomocą myszy i kilku okien. Niezależnie od tego, czy tworzyli pliki, zmieniali ich nazwy, umieszczali je w podkatalogach czy przenosili na dysku, użytkownicy w latach 70. mogli robić wszystko całkowicie za pomocą interfejsu tekstowego.

Każde z tych zadań wymagało własnego programu lub polecenia: jednego do zmiany katalogów (`cd`), drugiego do wyświetlenia ich zawartości (`ls`), trzeciego do zmiany nazwy lub przeniesienia plików (`mv`) itd. Aby skoordynować wykonywanie każdego z tych programów, użytkownik łączył się z jednym programem głównym, który mógł być następnie użyty do uruchomienia każdego z pozostałych. Poprzez opakowywanie poleceń użytkownika, ten program „`shell`", jak go nazywano, mógł zapewnić wspólne możliwości dla każdego z nich, takie jak możliwość przekazywania danych z jednego polecenia bezpośrednio do drugiego, lub używania

specjalnych znaków wieloznacznych do pracy z wieloma podobnie nazwanymi plikami na raz. Użytkownicy mogli nawet pisać prosty kod (zwany „skryptami powłoki”), który mógł być użyty do zautomatyzowania długich serii poleceń powłoki w celu ułatwienia skomplikowanych zadań. Oryginalny program powłoki Uniksa nazywał się po prostu `sh`, ale został rozszerzony i zastąpiony na przestrzeni lat, więc w nowoczesnym systemie Linux najprawdopodobniej będziemy używać powłoki o nazwie `bash`. Nie musimy się zbytnio przejmować tym, jakiej powłoki używamy, ponieważ wszystkie treści zawarte w tym samouczku będą działać na prawie wszystkich.

Linux jest swego rodzaju spadkobiercą Uniksa. Główna część Linuksa została zaprojektowana tak, by zachowywać się podobnie do systemu Uniksowego, dzięki czemu większość starych powłok i innych programów tekstowych działa na nim całkiem dobrze. W teorii można by nawet podłączyć jeden z tych starych terminali z lat 70-tych do nowoczesnego Linuksa i uzyskać dostęp do powłoki za jego pośrednictwem. Jednak w dzisiejszych czasach o wiele bardziej powszechne jest używanie terminala programowego: ten sam stary uniksowy interfejs tekstowy, ale działający w oknie obok programów graficznych. Zobaczymy, jak można to zrobić samemu.

3 Terminal

W tym i następnych rozdziałach zakładamy, że polecenia wykonuje użytkownik `bob` na komputerze o nazwie `lion`. Po uruchomieniu terminala zobaczymy znak zachęty:

```
[bob@lion ~]$
```

Wykonamy teraz nasze pierwsze polecenie. Klikamy myszką w okno, aby upewnić się, że to właśnie tam będą wykonywane naciśnięcia klawiszy, a następnie wpisujemy poniższe polecenie, wszystkie małymi literami, przed naciśnięciem klawisza <Enter> lub `<Return>`, aby je uruchomić. Powinniśmy zobaczyć wypisaną ścieżkę do bieżącego katalogu:

```
[bob@lion ~]$ pwd
/home/bob/
[bob@lion ~]$
```

Jest kilka rzeczy, które należy zrozumieć, zanim przejdziemy do szczegółów tego, co to polecenie faktycznie zrobiło. Pierwszą z nich jest to, że kiedy wpiszemy polecenie, pojawia się ono w tej samej linii co znak zachęty. Ten tekst jest tam, aby powiedzieć, że komputer jest gotowy do przyjęcia polecenia, jest to sposób komputera na podpowiadanie. W rzeczywistości jest to zwykle określane jako znak zachęty, a czasami można zobaczyć instrukcje, które mówią „przywołać znak zachęty”, „otworzyć znak zachęty”, „w znaku zachęty bash” lub podobne. To wszystko są po prostu różne sposoby proszenia o otwarcie terminala, by dostać się do powłoki.

Innym sposobem patrzenia na znak zachęty jest powiedzenie, że jest to linia w terminalu, do której wpisujemy polecenia. Innymi słowy: *wiersz poleceń*. Ponownie, jeżeli widzimy wzmiankę

o „wierszu poleceń”, włączając w to tytuł tego samouczka, jest to po prostu inny sposób mówienia o powłoce uruchomionej w terminalu.

Drugą rzeczą do zrozumienia jest to, że kiedy uruchamiamy polecenie, każde wyjście, jakie ono wytworzy, będzie zazwyczaj wypisane bezpośrednio w terminalu, a po jego zakończeniu zostanie nam pokazany kolejny znak zachęty. Niektóre polecenia mogą wypisać dużo tekstu, inne będą działać cicho i nie wypisują nic. Nie bądźmy zaniepokojeni, jeżeli uruchomimy polecenie i natychmiast pojawi się kolejny znak zachęty, ponieważ zwykle oznacza to, że polecenie się powiodło. Jeżeli przypomnimy sobie powolne połączenia sieciowe terminali z lat siedemdziesiątych, to zrozumiemy, że ówcześni programiści zdecydowali, iż jeżeli wszystko idzie dobrze, mogą zaoszczędzić kilka cennych bajtów transferu danych, nie mówiąc nic.

Znaczenie wielkości liter

Należy zachować szczególną ostrożność z wielkością liter podczas wpisywania w wierszu poleceń. Wpisanie `PWD` zamiast `pwd` spowoduje błąd, ale czasami zła wielkość liter może spowodować, że polecenie zostanie uruchomione, ale nie zrobi tego, czego oczekiwaliśmy. Przyjrzymy się tej sprawie nieco bardziej w następnym rozdziale, ale na razie upewnijmy się, że wszystkie poniższe linie wpisujemy dokładnie w taki sposób, jaki jest pokazany.

4 Tworzenie katalogów i plików

W tym rozdziale stworzymy kilka prawdziwych plików do pracy. Aby uniknąć przypadkowego uszkodzenia któregoś z naszych prawdziwych plików, zaczniemy od utworzenia nowego katalogu, z dala od naszego katalogu domowego, który będzie bezpieczniejszym środowiskiem do eksperymentowania:

```
[bob@lion ~]$ mkdir /tmp/tutorial
[bob@lion ~]$ cd /tmp/tutorial
```

Zauważmy, że użyta została *ścieżka bezwzględna*, aby upewnić się, że utworzymy katalog tutorial wewnątrz katalogu `/tmp`. Bez ukośnika na początku polecenie `mkdir` próbowałoby znaleźć katalog `tmp` wewnątrz bieżącego katalogu roboczego, a następnie spróbować utworzyć w nim katalog `tutorial`. Jeżeli nie znalazłoby katalogu `tmp`, to polecenie zakończyłoby się niepowodzeniem.

Teraz, gdy jesteśmy już bezpiecznie wewnątrz naszego obszaru testowego (sprawdźmy dwukrotnie za pomocą `pwd`, jeżeli nie jesteśmy pewni), stwórzmy kilka podkatalogów:

```
[bob@lion tutorial]$ mkdir dir1 dir2 dir3
```

Do tej pory widzieliśmy tylko polecenia, które działają samodzielnie (`cd`, `pwd`) lub które mają po sobie jeden element (`cd /etc`, `cd /usr`). Ale tym razem dodaliśmy trzy elementy po

poleceniu `mkdir`. Te elementy są określane jako argumenty, a różne polecenia mogą przyjmować różną liczbę argumentów. Polecenie `mkdir` oczekuje co najmniej jednego argumentu, natomiast polecenie `cd` może pracować z zerem lub jednym, ale nie więcej. Zobaczmy co się stanie gdy spróbujemy przekazać niewłaściwą liczbę argumentów do polecenia:

```
[bob@lion tutorial]$ mkdir
mkdir: brakujący argument
Napisz mkdir --help dla uzyskania informacji.
[bob@lion tutorial]$ cd /etc/ /usr
bash: cd: za dużo argumentów
[bob@lion tutorial]$
```

Wracając do naszych nowych katalogów. Powyższe polecenie spowoduje utworzenie trzech nowych podkatalogów wewnątrz naszego katalogu. Przyjrzyjmy się im za pomocą polecenia `ls` (list):

```
[bob@lion tutorial]$ ls
dir1 dir2 dir3
[bob@lion tutorial]$
```

Zauważmy, że polecenie `mkdir` utworzyło wszystkie katalogi w jednym katalogu. Nie utworzyło ono `dir3` wewnątrz `dir2` wewnątrz `dir1`, ani żadnej innej zagnieżdżonej struktury. Ale czasami przydaje się możliwość zrobienia dokładnie tego, a `mkdir` ma na to sposób:

```
[bob@lion tutorial]$ mkdir -p dir4/dir5/dir6
[bob@lion tutorial]$ ls
dir1 dir2 dir3 dir4
[bob@lion tutorial]$
```

Tym razem zobaczymy, że tylko `dir4` został dodany do listy, ponieważ `dir5` znajduje się wewnątrz niego, a `dir6` wewnątrz `dir5`. Później zainstalujemy przydatne narzędzie do wizualizacji struktury, ale mamy już wystarczającą wiedzę, aby ją potwierdzić:

```
[bob@lion tutorial]$ cd dir4
[bob@lion dir4]$ ls
dir5
[bob@lion dir4]$ cd dir5
[bob@lion dir5]$ ls
dir6
[bob@lion dir5]$ cd ../..
[bob@lion tutorial]$
```

Użyte „`-p`” w poleceniu `mkdir` nazywane jest opcją lub przełącznikiem (w tym przypadku oznacza „utwórz również katalogi nadrzędne”). Opcje są używane do modyfikowania sposobu

działania polecenia, pozwalając pojedynczemu poleceniu zachowywać się na wiele różnych sposobów. Niestety, z powodu dziwactw historii i ludzkiej natury, opcje mogą przybierać różne formy w różnych poleceniach. Często zobaczymy je jako pojedyncze znaki poprzedzone myślnikiem (jak w tym przypadku), lub jako dłuższe słowa poprzedzone dwoma myślnikami. Forma pojedynczego znaku pozwala na łączenie wielu opcji, choć nie wszystkie polecenia to akceptują. A żeby jeszcze bardziej zagmatwać sprawę, niektóre komendy w ogóle nie określają wyraźnie swoich opcji, to czy coś jest opcją czy nie, jest podyktowane wyłącznie kolejnością argumentów! Nie musimy się martwić o wszystkie możliwości, wystarczy wiedzieć, że opcje istnieją i mogą mieć kilka różnych form. Przykładowo, wszystkie poniższe oznaczają dokładnie to samo:

```
# Nie wpisuj poniższych poleceń.
# Są tu tylko w celach demonstracyjnych.
[bob@lion tutorial]$ mkdir --parents --verbose dir4/dir5
[bob@lion tutorial]$ mkdir -p --verbose dir4/dir5
[bob@lion tutorial]$ mkdir -p -v dir4/dir5
[bob@lion tutorial]$ mkdir -pv dir4/dir5
```

Teraz wiemy, jak utworzyć wiele katalogów po prostu przekazując je jako argumenty separujące do polecenia `mkdir`. Ale przypuśćmy, że chcemy utworzyć katalog ze spacją w nazwie. Spróbujmy to zrobić:

```
[bob@lion tutorial]$ mkdir other dir
[bob@lion tutorial]$ ls
dir  dir1  dir2  dir3  dir4  other
[bob@lion tutorial]$
```

Prawdopodobnie nie trzeba było nawet wpisywać tego polecenia, aby domyślić się, co się stanie: dwa nowe katalogi, jeden o nazwie `other`, a drugi o nazwie `dir`. Jeżeli chcemy pracować ze spacjami w nazwach katalogów lub plików, musimy od nich „uciec”. Ucieczka jest terminem informatycznym, który odnosi się do używania specjalnych kodów, aby powiedzieć komputerowi, żeby traktował określone znaki inaczej niż normalnie. Wprowadźmy następujące polecenia, aby wypróbować różne sposoby tworzenia katalogów ze spacjami w nazwie:

```
[bob@lion tutorial]$ mkdir "dir 1"
[bob@lion tutorial]$ mkdir 'dir 2'
[bob@lion tutorial]$ mkdir dir\ 3
[bob@lion tutorial]$ mkdir "dir 4" "dir 5"
[bob@lion tutorial]$ mkdir -p "dir 6"/"dir 7"
[bob@lion tutorial]$ ls
dir 'dir 1' 'dir 2' 'dir 3' 'dir 4' 'dir 5' 'dir 6' dir1 dir2 dir3 dir4 other
```

Chociaż wiersz poleceń może być używany do pracy z plikami i katalogami ze spacjami w ich nazwach, konieczność ucieczki od nich za pomocą cudzysłowów lub odwrotnych uko-

śników czyni rzeczy nieco trudniejszymi. Często można rozpoznać osobę, która często korzysta z wiersza poleceń po tworzonych przez nią nazwach plików: będą w nich zazwyczaj używane litery i cyfry, a ponadto podkreślniki (_) lub myślniki (-) zamiast spacji.

Tworzenie plików przy użyciu przekierowania

Nasz katalog demonstracyjny zaczyna wyglądać na pełen katalogów, ale brakuje w nim plików. Naprawimy to, przekierowując wyjście z polecenia tak, aby zamiast na ekran, trafiło do nowego pliku. Najpierw przypomnijmy sobie, co aktualnie znajduje się w katalogu. W tym celu wykonamy polecenie `ls`, ale z opcją `-l`. Opcja ta włącza długi format, wyświetlając uprawnienia, liczbę twardych dowiązań, właściciela, grupę, rozmiar, datę i czas ostatniej modyfikacji i nazwę pliku. Jeżeli data modyfikacji jest starsza niż 6 miesięcy, to czas jest zastępowany rokiem.

```
[bob@lion tutorial]$ ls -l
razem 0
drwxr-xr-x 2 bob users 40 10-24 12:39  other
drwxr-xr-x 2 bob users 40 10-24 12:39  dir
drwxr-xr-x 2 bob users 40 10-24 12:40 'dir 1'
drwxr-xr-x 2 bob users 40 10-24 12:40 'dir 2'
drwxr-xr-x 2 bob users 40 10-24 12:40 'dir 3'
drwxr-xr-x 2 bob users 40 10-24 12:40 'dir 4'
drwxr-xr-x 2 bob users 40 10-24 12:40 'dir 5'
drwxr-xr-x 3 bob users 60 10-24 12:41 'dir 6'
drwxr-xr-x 2 bob users 40 10-24 12:39  dir1
drwxr-xr-x 2 bob users 40 10-24 12:39  dir2
drwxr-xr-x 2 bob users 40 10-24 12:39  dir3
drwxr-xr-x 2 bob users 40 10-24 12:39  dir4
[bob@lion tutorial]$
```

Załóżmy, że chcemy zapisać wyjście tego polecenia w postaci pliku tekstowego, który możemy przeglądać lub dalej nim manipulować. Wszystko co musimy zrobić to dodać znak większości („>”) na końcu naszego wiersza poleceń, a następnie nazwę pliku, do którego chcemy zapisać dane:

```
[bob@lion tutorial]$ ls -l > output.txt
[bob@lion tutorial]$
```

Tym razem nic nie jest wypisywane na ekran, ponieważ wyjście jest przekierowywane do naszego pliku. Jeżeli uruchomimy teraz polecenie `ls`, to zobaczymy, że plik `output.txt` został utworzony. Możemy użyć polecenia `cat`, aby zobaczyć jego zawartość:

```
[bob@lion tutorial]$ cat output.txt
```

Nie jest to dokładnie to, co było wyświetlane na ekranie poprzednio, ale zawiera wszystkie te same dane i jest w bardziej użytecznym formacie do dalszej obróbki. Przyjrzyjmy się teraz kolejnemu poleceniu:

```
[bob@lion tutorial]$ echo "To jest test"
```

Polecenie `echo` po prostu wypisuje swoje argumenty z powrotem (stąd nazwa). Ale połączmy to z przekierowaniem i mamy sposób na łatwe tworzenie małych plików testowych:

```
[bob@lion tutorial]$ echo "To jest test" > test-1.txt
[bob@lion tutorial]$ echo "To jest drugi test" > test-2.txt
[bob@lion tutorial]$ echo "To jest trzeci test" > test-3.txt
[bob@lion tutorial]$ ls
```

Powinniśmy wykonać polecenie `cat` dla każdego z tych plików, aby sprawdzić jego zawartość. Ale polecenie `cat` jest czymś więcej niż tylko przeglądarką plików – jego nazwa pochodzi od słowa „concatenate”, co oznacza „łączyć razem”. Jeżeli podamy poleceniu `cat` więcej niż jedną nazwę pliku, wyświetli ono każdy z nich, jeden po drugim, jako pojedynczy blok tekstu:

```
[bob@lion tutorial]$ cat test-1.txt test-2.txt test-3.txt
```

W przypadku, gdy chcemy przekazać wiele nazw plików do jednego polecenia, istnieje kilka przydatnych skrótów, które mogą zaoszczędzić sporo wpisywania, jeżeli pliki mają podobne nazwy. Znak zapytania (`?`) może być użyty do wskazania „dowolnego pojedynczego znaku” w nazwie pliku. Gwiazdka (`*`) może być użyta do wskazania „zera lub więcej znaków”. Są one czasami określane jako „znaki wieloznaczne”. Następujące polecenia robią to samo:

```
[bob@lion tutorial]$ cat test-1.txt test-2.txt test-3.txt
[bob@lion tutorial]$ cat test-?.txt
[bob@lion tutorial]$ cat test-*
```

Jak można się domyślać, ta możliwość oznacza również konieczność ucieczki od nazw plików zawierających znaki `?` lub `*`. Zazwyczaj lepiej jest unikać interpunkcji w nazwach plików, jeżeli chcemy manipulować nimi z wiersza poleceń.

Jeżeli przyjrzymy się wyjściu polecenia `ls`, to zauważymy, że jedynymi plikami lub katalogami, które zaczynają się od `'t'`, są trzy pliki testowe, które właśnie stworzyliśmy, więc moglibyśmy nawet jeszcze bardziej uprościć to ostatnie polecenie do `cat t*`, czyli „połącz wszystkie pliki, których nazwy zaczynają się od `t` i po których następuje zero lub więcej innych znaków”. Użyjmy tej możliwości do połączenia wszystkich naszych plików w jeden nowy plik, a następnie wyświetlmy go:

```
[bob@lion tutorial]$ cat t* > combined.txt
[bob@lion tutorial]$ cat combined.txt
```

Zastanówmy się, co się stanie, jeżeli uruchomimy te dwa polecenia po raz drugi? Czy komputer będzie narzekał, bo plik już istnieje? Czy doda tekst do pliku, więc będzie on zawierał dwie kopie? A może całkowicie go zastąpi? Spróbujmy i zobaczmy, co się stanie, ale aby uniknąć ponownego wpisywania poleceń, możemy użyć klawiszy strzałek w górę i w dół do poruszania się w historii poleceń, których użyliśmy. Naciśnijmy strzałkę w górę kilka razy, aby dostać się do pierwszego polecania `cat` i naciśnijmy klawisz `<Enter>`, aby uruchomić polecenie, a następnie zróbmy to samo, aby dostać się do drugiego.

Jak widzimy, plik wygląda tak samo. To nie dlatego, że nie został zmieniony, ale dlatego, że powłoka czyści całą zawartość pliku zanim zapisze do niego wyjście naszego polecenia `cat`. Z tego powodu, powinniśmy być bardzo ostrożni podczas używania przekierowań, aby upewnić się, że nie nadpiszemy przypadkowo potrzebnego nam pliku. Jeżeli chcemy dołączyć, a nie zastąpić, zawartość plików, należy użyć dwóch znaków większości:

```
[bob@lion tutorial]$ cat t* >> combined.txt
[bob@lion tutorial]$ echo "I've appended a line!" >> combined.txt
[bob@lion tutorial]$ cat combined.txt
```

Powtórzmy pierwsze polecenie `cat` jeszcze kilka razy, używając strzałki w górę dla wygody, i być może dodajmy kilka innych arbitralnych poleceń `echo`, aż nasz dokument tekstowy będzie tak duży, że nie zmieści się w terminalu, gdy użyjemy polecenia `cat` do jego wyświetlenia. Aby zobaczyć cały plik, musimy teraz użyć innego programu, zwanego „pagerem” (ponieważ wyświetla on plik po jednej „stronie” na raz). Standardowy pager starego typu był nazywany `more`, ponieważ umieszczał linię tekstu na dole każdej strony z napisem `'-More-'`, aby wskazać, że nie przeczytaliśmy jeszcze wszystkiego. W dzisiejszych czasach istnieje znacznie lepszy pager, którego powinniśmy używać zamiast `more`; ponieważ zastępuje on `more`, programiści zdecydowali się nazwać go `less`.

```
[bob@lion tutorial]$ less combined.txt
```

Podczas przeglądania pliku za pomocą `less` możemy używać klawiszy Strzałka w górę, Strzałka w dół, Page Up, Page Down, Home i End do poruszania się po naszym pliku. Wypróbujmy je, aby zobaczyć różnicę między nimi. Kiedy skończymy przeglądać nasz plik, wciśnijmy klawisz `q`, aby wyjść z `less` i wrócić do wiersza poleceń.

Uwaga na temat wielkości liter

Systemy uniksowe rozróżniają wielkość liter, czyli uznają `"A.txt"` i `"a.txt"` za dwa różne pliki. Gdybyśmy uruchomili poniższe polecenia, skończylibyśmy z trzema plikami:

```
[bob@lion tutorial]$ echo "Lower case" > a.txt
[bob@lion tutorial]$ echo "Upper case" > A.TXT
[bob@lion tutorial]$ echo "Mixed case" > A.txt
```

Generalnie powinniśmy starać się unikać tworzenia plików i katalogów, których nazwa różni się tylko przypadkiem. Nie tylko pomoże to uniknąć zamieszania, ale także zapobiegnie problemom podczas pracy z różnymi systemami operacyjnymi. Na przykład system Windows nie rozróżnia wielkości liter, więc potraktowałby wszystkie trzy powyższe nazwy plików jako jeden plik, co mogłoby spowodować utratę danych lub inne problemy.

Może nas kusić, by po prostu nacisnąć klawisz Caps Lock i używać wielkich liter dla wszystkich nazw plików. Ale większość poleceń powłoki to małe litery, więc skończyłoby się na tym, że musielibyśmy często włączać i wyłączać je podczas pisania. Większość doświadczonych użytkowników wiersza poleceń ma tendencję do trzymania się głównie małych liter dla swoich plików i katalogów, dzięki czemu rzadko muszą się martwić o kolizje nazw plików, lub o to, jakiej wielkości użyć dla każdej litery w nazwie.

Dobra praktyka nadawania nazw

Jeżeli weźmiemy pod uwagę zarówno wielkość liter jak i ucieczkę, dobrą zasadą jest używanie w nazwach plików tylko małych liter, cyfr, podkreśleń i myślników. W przypadku plików zazwyczaj występuje również kropka i kilka znaków na końcu, aby wskazać typ pliku (określamy to jako "rozszerzenie pliku"). Ta zasada może wydawać się restrykcyjna, ale jeżeli będziemy używać wiersza poleceń z jakąkolwiek częstotliwością, będziemy zadowoleni, że trzymamy się tego wzorca.

5 Operacje na plikach

Kopiowanie i zmiana nazwy plików

Teraz, gdy mamy już kilka plików, przyjrzyjmy się rodzajom codziennych zadań, które można na nich wykonać. W praktyce nadal będziemy najprawdopodobniej używać programu graficznego, gdy będziemy chcieli przenieść, zmienić nazwę lub usunąć jeden lub dwa pliki, ale wiedza, jak to zrobić za pomocą wiersza poleceń może być przydatna do masowych zmian lub gdy pliki są rozmieszczone w różnych katalogach. Dodatkowo, po drodze dowiemy się kilku innych rzeczy o linii poleceń.

Zacznijmy od umieszczenia naszego pliku `combined.txt` w katalogu `dir1`, używając polecenia `mv` (move):

```
[bob@lion tutorial]$ mv combined.txt dir1
```

Można potwierdzić, że zadanie zostało wykonane poprzez użycie polecenia `ls`, aby zobaczyć, że brakuje go w katalogu roboczym; następnie `cd dir1`, aby przejść do podkatalogu `dir1`; `ls`, aby zobaczyć, że plik `combined.txt` tam jest; a następnie `cd ..`, aby zmienić z powrotem katalog roboczy. Można też zaoszczędzić wiele pisania, przekazując ścieżkę bezpośrednio do polecenia `ls`, aby uzyskać od razu to potwierdzenie, którego szukamy:

```
[bob@lion tutorial]$ ls dir1
```

Załóżmy teraz, że okaże się, że plik nie powinien być w ogóle w `dir1`. Przenieśmy go z powrotem do katalogu roboczego. Moglibyśmy wejść do `dir1`, a następnie użyć polecenia `mv combined.txt ..`, aby powiedzieć: „przenieś plik `combined.txt` do katalogu nadrzędnego". Ale możemy też użyć innego skrótu ścieżki, aby uniknąć zmiany katalogu w ogóle. Mianowicie, dwie kropki (`..`) reprezentują katalog nadrzędny, a pojedyncza kropka (`.`) reprezentuje bieżący katalog roboczy. Ponieważ wiemy, że w `dir1` jest tylko jeden plik, możemy również użyć znaku gwiazdki `*` do dopasowania dowolnej nazwy pliku w tym katalogu, oszczędzając sobie kilku dodatkowych naciśnięć klawiszy. Nasze polecenie przeniesienia pliku z powrotem do katalogu roboczego wygląda więc tak jak poniżej (zauważmy spację przed kropką oraz to, że do polecenia `mv` przekazywane są dwa argumenty):

```
[bob@lion tutorial]$ mv dir1/* .
```

Polecenie `mv` pozwala nam również przenieść więcej niż jeden plik na raz. Jeżeli podamy więcej niż dwa argumenty, ostatni z nich jest traktowany jako katalog docelowy, a pozostałe jako pliki (lub katalogi) do przeniesienia. Użyjmy jednego polecenia do przeniesienia pliku `combined.txt`, wszystkich naszych plików `test-n.txt` oraz katalogu `dir3` do katalogu `dir2`. Dzieje się tu trochę więcej, ale jeżeli przyjrzymy się każdemu argumentowi z osobna, powinniśmy być w stanie zrozumieć, co się tu dzieje:

```
[bob@lion tutorial]$ mv combined.txt test-* dir3 dir2
[bob@lion tutorial]$ ls
[bob@lion tutorial]$ ls dir2
```

Skoro `combined.txt` został przeniesiony do katalogu `dir2`, co się stanie, jeżeli uznamy, że znowu jest w złym miejscu? Zamiast do katalogu `dir2`, powinien zostać umieszczony w katalogu `dir6`, który jest wewnątrz katalogu `dir5`, który z kolei jest w katalogu `dir4`. Z tym, co już wiemy o ścieżkach, to też nie jest problemem:

```
[bob@lion tutorial]$ mv dir2/combined.txt dir4/dir5/dir6
[bob@lion tutorial]$ ls dir2
[bob@lion tutorial]$ ls dir4/dir5/dir6
```

Zauważmy, że nasze polecenie `mv` pozwoliło nam przenieść plik z jednego katalogu do drugiego, mimo że nasz katalog roboczy jest zupełnie inny. Jest to potężna właściwość wiersza poleceń: bez względu na to, gdzie w systemie plików się znajdujemy, wciąż możliwe jest operowanie na plikach i katalogach w zupełnie innych lokalizacjach.

Ponieważ wydaje się, że będziemy często używać (i przenosić) ten plik, być może powinniśmy zachować jego kopię w naszym katalogu roboczym. Podobnie jak polecenie `mv` przenosi pliki, tak polecenie `cp` kopiuje je (ponownie, zauważmy spację przed kropką):

```
[bob@lion tutorial]$ cp dir4/dir5/dir6/combined.txt .
[bob@lion tutorial]$ ls dir4/dir5/dir6
[bob@lion tutorial]$ ls
```

Teraz stwórzmy kolejną kopię pliku, w naszym katalogu roboczym, ale o innej nazwie. Możemy ponownie użyć polecenia `cp`, ale zamiast podawać mu ścieżkę do katalogu jako ostatni argument, podamy mu nową nazwę pliku:

```
[bob@lion tutorial]$ cp combined.txt backup-combined.txt
[bob@lion tutorial]$ ls
```

To jest niezłe, ale być może wybór nazwy kopii zapasowej mógłby być lepszy. Może warto zmienić nazwę tak, aby zawsze pojawiała się obok oryginalnego pliku na posortowanej liście. Tradycyjny wiersz poleceń systemu Unix obsługuje zmianę nazwy tak, jakbyśmy przenosili plik z jednej nazwy na drugą, więc naszym dobrym rozwiązaniem jest polecenie `mv`. W tym przypadku podajemy tylko dwa argumenty: plik, którego nazwę chcemy zmienić, oraz nową nazwę, której chcemy użyć.

```
[bob@lion tutorial]$ mv backup-combined.txt combined-backup.txt
[bob@lion tutorial]$ ls
```

Działa to również na katalogi, dając nam sposób na uporządkowanie tych trudnych ze spacjami w nazwie, które stworzyliśmy wcześniej. Aby uniknąć ponownego wpisywania każdego polecenia po pierwszym, użyjmy strzałki w górę, aby wyświetlić poprzednie polecenie w historii. Możemy wtedy edytować polecenie przed jego uruchomieniem, przesuwając kursor w lewo i w prawo za pomocą klawiszy strzałek i usuwając znak po lewej stronie za pomocą klawisza `<Backspace>` lub ten, na którym znajduje się kursor za pomocą klawisza `<Delete>`. Na koniec wpiszmy nowy znak w to miejsce i naciśnijmy `<Enter>` lub `<Return>`, aby uruchomić polecenie, gdy skończymy. Upewnijmy się, że w każdej z tych linii zmieniamy oba wystąpienia liczby.

```
[bob@lion tutorial]$ mv "dir 1" dir-1
[bob@lion tutorial]$ mv "dir 2" dir-2
[bob@lion tutorial]$ mv "dir 3" dir-3
[bob@lion tutorial]$ mv "dir 4" dir-4
[bob@lion tutorial]$ mv "dir 5" dir-5
[bob@lion tutorial]$ mv "dir 6" dir-6
[bob@lion tutorial]$ ls
```

Usuwanie plików i katalogów

W tej sekcji zaczniemy usuwać pliki i katalogi. Aby mieć absolutną pewność, że nie usuniemy przypadkowo niczego z katalogu domowego, użyjmy polecenia `pwd`, aby ponownie sprawdzić, czy nadal znajdujemy się w katalogu `/tmp/tutorial`, zanim przejdziemy dalej.

Wiemy już jak przenosić, kopiować i zmieniać nazwy plików i katalogów. Biorąc jednak pod uwagę, że są to tylko pliki testowe, być może tak naprawdę nie potrzebujemy trzech różnych kopii `combined.txt`. Posprzątajmy trochę, używając polecenia `rm` (remove):

```
[bob@lion tutorial]$ rm dir4/dir5/dir6/combined.txt combined-backup.txt
```

Być może powinniśmy usunąć również niektóre z tych nadmiarowych katalogów:

```
[bob@lion tutorial]$ rm dir-*
rm: nie można usunąć 'dir-1': Jest katalogiem
rm: nie można usunąć 'dir-2': Jest katalogiem
rm: nie można usunąć 'dir-3': Jest katalogiem
rm: nie można usunąć 'dir-4': Jest katalogiem
rm: nie można usunąć 'dir-5': Jest katalogiem
rm: nie można usunąć 'dir-6': Jest katalogiem
```

Co się tu wydarzyło? Otóż, okazuje się, że polecenie `rm` ma jedno małe zabezpieczenie. Owszem, możemy użyć go do usunięcia każdego pliku w katalogu za pomocą jednego polecenia, przypadkowo usuwając tysiące plików w jednej chwili, bez możliwości ich odzyskania. Ale nie pozwala ono na usunięcie katalogu. Przypuszczalnie pomaga to zapobiec przypadkowemu usunięciu kolejnych tysięcy plików, ale wydaje się to trochę małostkowe dla tak destrukcyjnego polecenia, aby nie pozwolić na usunięcie pustego katalogu. Na szczęście istnieje polecenie `rmdir` (usuń katalog), które wykona tę czynność:

```
[bob@lion tutorial]$ rmdir dir-*
rmdir: nie udało się usunąć 'dir-6': Katalog nie jest pusty
```

Tak więc jest trochę lepiej, ale nadal jest błąd. Jeżeli uruchomimy polecenie `ls`, to zobaczymy, że większość katalogów zniknęła, ale katalog `dir-6` wciąż się tu znajduje. Jak możemy sobie przypomnieć, katalog `dir-6` wciąż ma wewnątrz siebie katalog `dir 7`, a `rmdir` usuwa tylko puste katalogi. Ponownie, jest to małe zabezpieczenie przed przypadkowym usunięciem katalogu pełnego plików, gdy tego nie chcieliśmy.

W tym przypadku jednak faktycznie o to nam idzie. Dodanie opcji do naszych poleceń `rm` lub `rmdir` pozwoli nam wykonać niebezpieczne działania bez pomocy zabezpieczeń. W przypadku polecenia `rmdir` możemy dodać przełącznik `-p`, aby usunąć również katalogi nadrzędne. Pomyślmy o tym jako o odpowiedniku polecenia `mkdir -p`. A więc jeżeli uruchomimy polecenie `rmdir -p dir1/dir2/dir3`, to najpierw usunie ono katalog `dir3`, potem katalog `dir2`, a na końcu usunie katalog `dir1`. Jednak nadal polecenie `rmdir` zachowuje swoją normalną za-

sadę usuwania tylko pustych katalogów. Przykładowo, jeżeli w katalogu `dir1` byłby jakiś plik, to zostałyby usunięte tylko katalogi `dir3` oraz `dir2`.

Bardziej powszechnym podejściem, gdy jesteśmy naprawdę pewni, że chcemy usunąć cały katalog i wszystko, co się w nim znajduje, jest powiedzenie poleceniu `rm` za pomocą przełącznika `-r`, aby działało rekurencyjnie. W takim przypadku zostaną usunie zarówno katalogi, jak również pliki. Mając to na uwadze, zobaczmy polecenie, które pozwoli pozbyć się tego kłopotliwego katalogu `dir-6` oraz znajdującego się w nim podkatalogu:

```
[bob@lion tutorial]$ rm -r dir-6
[bob@lion tutorial]$ ls
```

Pamiętajmy, że chociaż polecenie `rm -r` jest szybkie i wygodne, to jest również niebezpieczne. Najbezpieczniej jest jawnie usunąć pliki, aby wyczyścić katalog, następnie przejść do katalogu nadrzędnego poprzez polecenie `cd ..`, po czym usunąć katalog poleceniem `rmdir`.

Ważne ostrzeżenie

W przeciwieństwie do interfejsów graficznych, polecenie `rm` nie przenosi plików do katalogu o nazwie `trash` lub podobnego. Zamiast tego usuwa je całkowicie i nieodwracalnie. Musimy być bardzo ostrożni z argumentami, których używamy z poleceniem `rm`, aby upewnić się, że usuwamy tylko te pliki, które naprawdę zamierzamy usunąć.

Powinniśmy zachować szczególną ostrożność podczas używania znaków wieloznacznych, ponieważ łatwo jest przypadkowo usunąć więcej plików niż zamierzaliśmy. Błędny znak spacji w naszym poleceniu może je całkowicie zmienić: polecenie `rm t*` oznacza „usuń wszystkie pliki zaczynające się od t”, podczas gdy polecenie `rm t *` oznacza „usuń plik t, jak również każdy plik, którego nazwa składa się z zera lub więcej znaków”, co oznacza wszystko w katalogu, za wyjątkiem plików i katalogów *ukrytych*. Pliki i katalogi ukryte to te, które posiadają na początku swojej nazwy kropkę.

Jeżeli nie jesteśmy pewni, użyjmy opcji `-i` (praca interaktywna), która spowoduje wyświetlenie monitu o potwierdzenie usunięcia każdego pliku. Po wyświetleniu monitu należy wcisnąć klawisz `Y`, aby usunąć plik, klawisz `N`, aby zachować plik, albo użyć kombinacji `<Ctrl>+C`, aby całkowicie zatrzymać operację.

6 Siła wiersza poleceń

Dzisiejsze komputery i telefony mają takie możliwości graficzne i dźwiękowe, jakich użytkownicy terminali z lat 70. nie mogli sobie nawet wyobrazić. A jednak tekst wciąż dominuje jako środek do organizowania i kategoryzowania plików. Niezależnie od tego, czy jest to sama nazwa pliku, współrzędne GPS osadzone w zdjęciach zrobionych telefonem, czy metadane zapisane w pliku audio, tekst wciąż odgrywa istotną rolę w każdym aspekcie informatyki. Na szczęście

wiersz poleceń Linuksa zawiera kilka potężnych narzędzi do manipulowania zawartością tekstu oraz sposoby łączenia tych narzędzi w celu stworzenia czegoś jeszcze bardziej wydajnego.

Zacznijmy od prostego pytania. Ile linii jest w naszym pliku `combined.txt`? Polecenie `wc` (word count) może nam to powiedzieć, używając przełącznika `-l`, aby wskazać, że chcemy zobaczyć tylko liczbę linii (polecenie `wc` może również liczyć znaki oraz, jak sama nazwa wskazuje, słowa):

```
[bob@lion ~]$ wc -l combined.txt
```

Analogicznie, gdybyśmy chcieli dowiedzieć się, ile plików i katalogów znajduje się w naszym katalogu domowym, a następnie posprzątać po sobie, można to zrobić w następujący sposób:

```
[bob@lion ~]$ ls ~ file-list.txt
[bob@lion ~]$ wc -l file-list.txt
[bob@lion ~]$ rm file-list.txt
```

Ta metoda działa, ale tworzenie pliku tymczasowego do przechowywania danych wyjściowych z polecenia `ls` tylko po to, by usunąć go dwie linie później, wydaje się pewną przesadą. Na szczęście wiersz poleceń Uniksa zapewnia skrót, który pozwala uniknąć konieczności tworzenia pliku tymczasowego, poprzez pobieranie wyjścia z jednego polecenia (określanego jako standardowe wyjście lub STDOUT) i podawanie go bezpośrednio jako wejście do innego polecenia (standardowe wejście lub STDIN). Jest to tak, jakbyśmy podłączyli rurę pomiędzy wyjściem jednego polecenia a wejściem następnego, do tego stopnia, że proces ten jest określany jako przesyłanie danych z jednego polecenia do drugiego. Oto jak przekazać wyjście naszego polecenia `ls` do polecenia `wc`:

```
[bob@lion ~]$ ls ~ | wc -l
```

Zauważmy, że nie jest tworzony żaden plik tymczasowy, ani nie jest potrzebna nazwa pliku. Potoki działają całkowicie w pamięci, a większość uniksowych narzędzi wiersza poleceń będzie oczekiwać, że otrzymają dane wejściowe z potoku, jeżeli nie podamy pliku, na którym mają pracować. Patrząc na powyższą linię, widzimy, że są to dwa polecenia, `ls` (wylistuj zawartość katalogu domowego) i `wc -l` (policz linie), oddzielone znakiem pionowej kreski (`|`). Ten proces łączenia jednego polecenia z drugim jest tak powszechnie stosowany, że sam znak jest często nazywany znakiem potoku, tak więc jeżeli widzimy ten zwrot, to wiemy, że oznacza on właśnie pionową kreskę.

Zwróćmy uwagę, że spacje wokół znaku potoku nie są istotne (zostały użyte dla przejrzystości), ale poniższe polecenie działa równie dobrze, tym razem dla poinformowania nas, ile pozycji znajduje się w katalogu `/etc`:

```
[bob@lion ~]$ ls /etc|wc -l
```

To całkiem sporo plików. Gdybyśmy chcieli je wszystkie wylistować, z pewnością wypełniłyby więcej niż jeden ekran. Jak stwierdziliśmy wcześniej, gdy polecenie produkuje dużo danych wyjściowych, lepiej jest polecenia `less` do ich wyświetlenia, i ta wskazówka nadal obowiązuje, gdy używamy potoku (pamiętajmy o naciśnięciu klawisza `q`, aby wyjść):

```
[bob@lion ~]$ ls /etc | less
```

Wracając do naszych plików, wiemy jak uzyskać liczbę linii w pliku `combined.txt`, ale biorąc pod uwagę, że został on utworzony przez wielokrotne łączenie tych samych plików, zastanawiamy się ile jest w nim unikalnych inii? Unix ma polecenie `uniq`, które wyświetli tylko unikalne linie w pliku. Musimy więc wyłuskać zawartość pliku i przesłać ją do polecenia `uniq`. Ale wszystko, czego chcemy, to liczba linii, więc musimy również użyć polecenia `wc`. Na szczęście wiersz poleceń nie ogranicza nas do pojedynczego potoku na raz, więc możemy kontynuować łańcuch tylu poleceń, ilu potrzebujemy:

```
[bob@lion ~]$ cat combined.txt | uniq | wc -l
```

Powyższe polecenie prawdopodobnie spowodowało uzyskanie liczby, która jest dość bliska całkowitej liczbie linii w pliku, jeżeli nie dokładnie taka sama. Z pewnością nie może to być właściwy wynik. Zmieńmy ostatni potok, aby zobaczyć wyjście polecenia dla lepszego wyobrażenia o tym, co się stało. Jeżeli nasz plik jest bardzo długi, możemy chcieć go przesłać do polecenia `less`, aby ułatwić kontrolę tego co się stało:

```
[bob@lion ~]$ cat combined.txt | uniq | less
```

Okazuje się, że bardzo niewiele, jeżeli w ogóle, naszych zduplikowanych linii jest usuwanych. Aby zrozumieć dlaczego, musimy przyjrzeć się dokumentacji polecenia `uniq`. Większość narzędzi wiersza poleceń posiada krótką (a czasem nie tak krótką) instrukcję obsługi, dostępną poprzez polecenie `man` (manual). Dane wyjściowe są automatycznie przekazywane do naszego pagera, którym zazwyczaj jest polecenie `less`, więc możemy poruszać się tam i z powrotem po danych wyjściowych, a następnie nacisnąć klawisz `q`, aby skończyć:

```
[bob@lion ~]$ man uniq
```

Ponieważ ten typ dokumentacji jest dostępny poprzez polecenie `man`, będziemy go nazywać `"man page"`, jak w „sprawdź stronę man po więcej szczegółów”. Format stron man jest często zwięzły, traktujmy je więc bardziej jako szybki przegląd polecenia niż pełny tutorial. Często są one bardzo techniczne, ale zazwyczaj możemy pominąć większość treści i po prostu poszukać szczegółów dotyczących opcji lub argumentu, którego używamy.

Strona man polecenia `uniq` jest typowym przykładem tego, że zaczyna się od krótkiego, jednolinijkowego opisu polecenia, przechodzi do streszczenia jak go używać, a następnie następuje szczegółowy opis każdej opcji lub parametru. Ale chociaż strony man są nieocenione, mogą być również nieprzeniknione. Najlepiej używać ich, gdy potrzebujemy przypomnienia o konkretnym przełączniku lub argumencie, a nie jako ogólnego źródła wiedzy o używaniu wiersza poleceń.

Niemniej, pierwsza linia sekcji DESCRIPTION dla `man uniq` odpowiada na pytanie, dlaczego nie usunięto zduplikowanych linii: działa to tylko na sąsiednich, zgodnych liniach.

Pytanie brzmi więc, jak zmienić układ linii w naszym pliku, aby zduplikowane wpisy znajdowały się w sąsiednich liniach. Gdybyśmy posortowali zawartość pliku alfabetycznie, załatwiłoby to sprawę. Unix oferuje polecenie `sort`, które właśnie to robi. Szybkie sprawdzenie `man sort` pokazuje, że możemy przekazać nazwę pliku bezpośrednio do polecenia, więc zobaczmy co zrobi ono z naszym plikiem:

```
[bob@lion ~]$ sort combined.txt | less
```

Powinniśmy być w stanie zobaczyć, że linie zostały uporządkowane i teraz nadają się do przesłania bezpośrednio do polecenia `uniq`. Możemy wreszcie zakończyć nasze zadanie zliczania unikalnych linii w pliku:

```
[bob@lion ~]$ sort combined.txt | uniq | wc -l
```

Jak widzimy, możliwość przesyłania danych z jednego polecenia do drugiego, budując długie łańcuchy do manipulowania danymi, jest potężnym narzędziem, jak również zmniejsza potrzebę plików tymczasowych i oszczędza nam wiele pisania. Z tego powodu zobaczymy je dość często używane w wierszu poleceń. Długi łańcuch poleceń może na początku wyglądać zastraszająco, ale pamiętajmy, że możemy rozbić nawet najdłuższy łańcuch na poszczególne polecenia (i zajrzeć do ich stron man), aby lepiej zrozumieć, co robią.

Manuale

Większość narzędzi wiersza poleceń Linuksa zawiera stronę `man`. Spróbujmy przyjrzeć się pokrótce stronom niektórych poleceń, z którymi już się zetknęliśmy: `man ls`, `man cp`, `man rmdir` i tak dalej. Istnieje nawet strona man dla samego programu `man`, do którego dostęp uzyskuje się oczywiście za pomocą `man man`.

Opcja pomocy

Większość narzędzi wiersza poleceń Linuksa pozwala użyć także opcji `--help`. Spróbujmy użyć tej opcji dla tych poleceń, z którymi już się zetknęliśmy: `man -help`, `mkdir -help` `ls -help`, `cp -help`, i tak dalej.

Polecenia wbudowane

Część poleceń to polecenia *wbudowane* w powłokę. Ich listę można uzyskać wykonując polecenie `help`:

```
[bob@lion ~]$ help
```

7 Wiersz poleceń i superużytkownik

Jednym z dobrych powodów do nauki podstaw wiersza poleceń jest to, że instrukcje dostępne w Internecie często faworyzują użycie poleceń powłoki zamiast interfejsu graficznego. Jeżeli te instrukcje wymagają zmian na komputerze, które wykraczają poza modyfikację kilku plików w katalogu domowym, nieuchronnie będziemy mieli do czynienia z poleceniami, które muszą być uruchamiane z prawami administratora komputera (superużytkownika w języku Uniksa). Zanim zaczniemy uruchamiać dowolne polecenia, które znajdziemy w jakimś ciemnym zakątku internetu, warto zrozumieć, jakie są konsekwencje jego uruchamiania jako administrator i jak rozpoznać te instrukcje, które tego wymagają, abyśmy mogli lepiej ocenić, czy można je bezpiecznie uruchomić, czy nie.

Superużytkownik to, jak sama nazwa wskazuje, użytkownik z super uprawnieniami. W starszych systemach był to prawdziwy użytkownik, z prawdziwą nazwą użytkownika (prawie zawsze „`root`"), który mógł się zalogować jako ten, który miał hasło. Jeżeli idzie o te super uprawnienia: użytkownik `root` może zmodyfikować lub usunąć dowolny plik w dowolnym katalogu w systemie, niezależnie od tego, kto jest jego właścicielem; `root` może przepisać reguły zapory lub uruchomić usługi sieciowe, które mogłyby potencjalnie otworzyć maszynę na atak; `root` może wyłączyć komputer, nawet jeżeli inne osoby nadal go używają. Krótko mówiąc, `root` może zrobić prawie wszystko, z łatwością omijając zabezpieczenia, które są zwykle wprowadzane, aby powstrzymać użytkowników przed przekraczaniem swoich uprawnień.

Oczywiście, osoba zalogowana jako `root` jest tak samo podatna na popełnianie błędów jak każdy inny. Kroniki historii informatyki pełne są opowieści o błędnie wpisanym poleceniu, które usuwa cały system plików lub zabija ważny serwer. Istnieje też możliwość złośliwego ataku: jeżeli użytkownik jest zalogowany jako `root` i odchodzi od biurka, to niezbyt trudno jest niezadowolonemu koledze wejść na jego komputer i dokonać spustoszenia. Pomimo tego, że natura ludzka jest taka, jaka jest, wielu administratorów przez lata było winnych używania konta roota jako swojego głównego lub jedynego konta.

Nie używajmy konta roota. Jeżeli ktoś poprosi nas o włączenie konta `root`, lub zalogowanie się jako root, należy być bardzo podejrzliwym co do jego intencji.

W celu zmniejszenia tych problemów wiele dystrybucji Linuksa zaczęło zachęcać do używania polecenia `su`. Polecenie to jest różnie opisywane jako skrót od "superuser"lub "switch user"i pozwala nam zmienić użytkownika na maszynie bez konieczności wylogowywania się i ponownego wchodzenia. Użyte bez argumentów zakłada, że chcemy zmienić użytkownika na root'a (stąd pierwsza interpretacja nazwy), ale możemy przekazać mu nazwę użytkownika, aby przełączyć się na konkretne konto użytkownika (druga interpretacja). Zachęcając do używania polecenia `su`, chciano przekonać administratorów do spędzania większości czasu na normalnym koncie, przełączania się na konto superużytkownika tylko wtedy, gdy jest to konieczne, a następnie używania polecenia wylogowania (lub skrótu `Ctrl+D`) tak szybko, jak to możliwe, aby powrócić na swoje konto użytkownika.

Poprzez zminimalizowanie czasu spędzonego na logowaniu jako root, użycie polecenia `su` zmniejsza okno możliwości popełnienia katastrofalnego błędu. Pomimo tego, natura ludzka jest taka, jaka jest, wielu administratorów było winnych pozostawienia otwartych długo działających terminali, w których użyli polecenia `su` do przejścia na konto root. Pod tym względem polecenie `su` było tylko małym krokiem naprzód dla bezpieczeństwa.

Nie używajmy su

Jeżeli ktoś prosi nas o użycie polecenia `su`, bądźmy ostrożni. Jeżeli używamy `Ubuntu`, konto root jest domyślnie wyłączone, więc polecenie `su` bez argumentów nie zadziała. Ale i tak nie warto ryzykować, na wypadek gdyby konto zostało włączone bez naszej świadomości. Jeżeli zostaniemy poproszeni o użycie `su` z nazwą użytkownika to (jeżeli mamy hasło) będziemy mieli dostęp do wszystkich plików tego użytkownika i moglibyśmy je przypadkowo usunąć lub zmodyfikować.

Podczas używania polecenia `su` cała nasza sesja terminalowa jest przełączana na innego użytkownika. Polecenia, które nie wymagają dostępu roota, coś tak prozaicznego jak `pwd` lub `ls`, będą uruchamiane pod auspicjami superużytkownika, zwiększając ryzyko, że błąd w programie spowoduje poważne problemy. Co gorsza, jeżeli stracimy kontrolę nad tym, jako który użytkownik aktualnie działamy, możemy wydać polecenie, które jest dość łagodne, gdy jest uruchamiane jako użytkownik, ale które może zniszczyć cały system, jeżeli zostanie uruchomione jako root.

Lepiej całkowicie wyłączyć konto root, a następnie, zamiast pozwalać na długotrwałe sesje terminalowe z niebezpiecznymi uprawnieniami, wymagać od użytkownika specjalnego żądania praw superużytkownika na poziomie poszczególnych poleceń. Kluczem do tego podejścia jest polecenie zwane `sudo` (przełącz użytkownika i wykonaj podane polecenie).

Polecenie `sudo` jest używane do poprzedzania poleceń, które muszą być uruchamiane z uprawnieniami superużytkownika. Plik konfiguracyjny `/etc/sudoers` jest używany do określenia, którzy użytkownicy mogą używać `sudo` i jakie polecenia mogą wykonywać. Podczas uruchamiania takiego polecenia, użytkownik jest proszony o podanie swojego hasła, które jest przechowywane w pamięci podręcznej przez pewien czas (domyślnie 15 minut), więc jeżeli musi uruchomić wiele poleceń na poziomie superużytkownika, nie będzie ciągle proszony o jego wpisywanie.

W systemie `Ubuntu` pierwszy użytkownik utworzony podczas instalacji systemu jest uważany za superużytkownika. Podczas dodawania nowego użytkownika istnieje opcja utworzenia go jako administratora, w którym to przypadku będzie on mógł również uruchamiać polecenia superużytkownika za pomocą polecenia `sudo`.

Zakładając, że jesteśmy w systemie Linux, który używa polecenia `sudo`, a nasze konto jest skonfigurowane jako konto administratora, spróbujmy wykonać następujące czynności, aby zobaczyć, co się stanie, gdy spróbujemy uzyskać dostęp do pliku, który jest uważany za wrażliwy (zawiera zaszyfrowane hasła):

```
[bob@lion ~]$ cat /etc/shadow
cat: /etc/shadow: Brak dostępu
[bob@lion ~]$ sudo cat /etc/shadow
[sudo] hasło użytkownika bob:
```

Jeżeli podamy nasze hasło, powinniśmy zobaczyć zawartość pliku `/etc/shadow`. Teraz wyczyśćmy terminal, wykonując polecenie `clear` albo `reset`, i ponownie uruchommy polecenie `sudo cat/etc/shadow`. Tym razem plik zostanie wyświetlony bez zapytania o hasło, ponieważ nadal jest ono w pamięci podręcznej.

Bądźmy ostrożni z sudo

Jeżeli zostaniemy poinstruowani, aby uruchomić polecenie z `sudo`, upewnijmy się, że rozumiemy, co to polecenie robi, zanim będziemy kontynuować. Uruchomienie z `sudo` daje temu poleceniu wszystkie te same uprawnienia, co superużytkownikowi. Przykładowo, strona wydawcy oprogramowania może poprosić nas o pobranie pliku i zmianę jego uprawnień, a następnie użycie `sudo` do uruchomienia go. Jeżeli nie wiemy dokładnie, co robi ten plik, otwieramy dziurę, przez którą złośliwe oprogramowanie może być potencjalnie zainstalowane w naszym systemie. Polecenie `sudo` może uruchamiać tylko jedno polecenie na raz, ale to polecenie może samo uruchamiać wiele innych. Każde nowe użycie `sudo` traktujmy jako równie niebezpieczne jak zalogowanie się jako `root`.

W przypadku instrukcji dotyczących `Ubuntu`, częstym pojawieniem się `sudo` jest instalacja nowego oprogramowania w naszym systemie za pomocą poleceń `apt` lub `apt-get`. Jeżeli instrukcje wymagają, abyśmy najpierw dodali nowe repozytorium oprogramowania do naszego systemu, używając polecenia `apt-add-repository`, edytując pliki w `/etc/apt`, lub używając „`PPA`” (`Personal Package Archive`), powinniśmy być ostrożni, ponieważ te źródła nie są nadzorowane przez `Canonical`. Często jednak instrukcje wymagają od nas po prostu zainstalowania oprogramowania ze standardowych repozytoriów, co powinno być bezpieczne.

Zainstalujmy nowy program z wiersza poleceń ze standardowych repozytoriów Ubuntu, aby zilustrować to użycie polecenia `sudo`:

```
[bob@lion ~]$ sudo apt install tree
```

Gdy podamy hasło, program `apt` wydrukuje sporo linijek tekstu informujących nas o tym, co robi. Program `tree` jest niewielki, więc pobranie i instalacja dla większości użytkowników nie powinna zająć więcej niż minutę lub dwie. Po powrocie do normalnego wiersza poleceń, program jest zainstalowany i gotowy do użycia. Uruchommy go, aby lepiej zorientować się, jak wygląda nasza kolekcja plików i katalogów:

```
[bob@lion ~]$ cd /tmp/tutorial
[bob@lion tutorial]$ tree
```

Wracając do polecenia, które faktycznie zainstalowało nowy program (`sudo apt install tree`) wygląda ono nieco inaczej niż te, które widzieliśmy do tej pory. W praktyce działa to w następujący sposób:

1. Polecenie `sudo`, użyte bez żadnych opcji, przyjmie, że pierwszy argument jest poleceniem do uruchomienia z uprawnieniami superużytkownika. Wszelkie inne argumenty zostaną przekazane bezpośrednio do nowego polecenia. Przełączniki `sudo` zaczynają się od jednego lub dwóch myślników i muszą bezpośrednio następować po poleceniu `sudo`, więc nie może być niejasności, czy drugi argument w wierszu jest poleceniem czy opcją.

2. Poleceniem w tym przypadku jest `apt`. W przeciwieństwie do innych poleceń, które widzieliśmy, to polecenie nie pracuje bezpośrednio z plikami. Zamiast tego oczekuje, że jego pierwszym argumentem będzie instrukcja do wykonania (`install`), a reszta argumentów będzie się zmieniać w zależności od podanej instrukcji.

3. W tym przypadku polecenie `install` mówi poleceniu `apt`, że pozostała część wiersza poleceń będzie się składać z jednej lub więcej nazw pakietów do zainstalowania z repozytoriów oprogramowania systemu. Zazwyczaj jest to dodanie nowego oprogramowania do systemu, ale pakietami mogą być dowolne zbiory plików, które należy zainstalować w określonych miejscach, takie jak czcionki lub obrazy tła pulpitu.

Ostrzeżenie

Jedną ze sztuczek z `sudo` jest użycie go do uruchomienia polecenia `su`. To da nam powłokę roota, nawet jeżeli konto roota jest wyłączone. Może to być przydatne, gdy musimy uruchomić serię poleceń jako superużytkownik, aby uniknąć konieczności poprzedzania ich wszystkich poleceniem `sudo`, ale otwiera nas na dokładnie ten sam rodzaj problemów, które zostały opisane powyżej dla `su`. Jeżeli stosujemy się do jakichkolwiek instrukcji, które każą nam uruchomić `sudo su`, bądźmy świadomi, że każde polecenie po tym będzie uruchamiane z prawami użytkownika `root`.

Instalowanie nowego oprogramowania

Istnieje wiele różnych sposobów na instalację oprogramowania w systemach Linux. Instalacja bezpośrednio z oficjalnych repozytoriów oprogramowania danej dystrybucji jest najbezpieczniejszą opcją, ale czasami aplikacja lub wersja, którą chcemy zainstalować nie jest dostępna w ten sposób. Podczas instalacji za pomocą innych mechanizmów, upewnijmy się, że pobieramy pliki z oficjalnego źródła danego projektu.

Oznaki, że pliki pochodzą spoza repozytoriów danej dystrybucji obejmują (ale nie są ograniczone tylko do nich) korzystanie z jednego z następujących poleceń: `curl`, `wget`, `pip`, `npm`, `make`, jak również instrukcji, które polecają zmianę uprawnień pliku, aby uczynić go wykonywalnym.

Coraz częściej Ubuntu korzysta ze "snapów", nowego formatu pakietów, który oferuje pewne ulepszenia w zakresie bezpieczeństwa poprzez nieco ściślejsze ograniczenie programów, aby powstrzymać je przed dostępem do tych części systemu, których nie potrzebują. Jednakże niektóre opcje mogą zmniejszyć poziom bezpieczeństwa, więc jeżeli jesteśmy proszeni o uruchomienie `snap install` z argumentami innymi niż nazwa snap'a, warto sprawdzić, co dokładnie dane polecenie próbuje zrobić.

8 Pliki ukryte

Zanim zakończymy ten samouczek warto wspomnieć o ukrytych plikach (i katalogach). Są one powszechnie używane w systemach Linux do przechowywania ustawień i danych konfiguracyjnych, i zazwyczaj są ukryte po prostu po to, aby nie zasłaniały widoku naszych własnych plików. Nie ma nic specjalnego w ukrytym pliku lub katalogu, poza jego nazwą: wystarczy rozpocząć nazwę od kropki (`.`), aby nie był on widoczny.

```
[bob@lion ~]$ cd /tmp/tutorial
[bob@lion tutorial]$ ls
[bob@lion tutorial]$ cp combined.txt .combined.txt
[bob@lion tutorial]$ ls
```

Nadal możemy pracować z ukrytym plikiem, upewniając się, że jego nazwa zawiera na początku kropkę:

```
[bob@lion tutorial]$ cat .combined.txt
[bob@lion tutorial]$ mkdir .hidden
[bob@lion tutorial]$ mv .combined.txt .hidden
[bob@lion tutorial]$ less .hidden/.combined.txt
```

Jeżeli uruchomimy polecenie `ls`, to zobaczymy, że katalog `.hidden` jest, jak można się spodziewać, ukryty. Możemy wypisać jego zawartość używając polecenia `ls .hidden`, ale ponieważ zawiera on tylko jeden plik, który sam w sobie jest ukryty, nie otrzymamy zbyt wiele informacji. Możemy jednak użyć przełącznika `-a` („show all") w poleceniu `ls`, aby pokazać wszystko w katalogu, w tym ukryte pliki i katalogi:

```
[bob@lion tutorial]$ ls
[bob@lion tutorial]$ ls -a
[bob@lion tutorial]$ ls .hidden
[bob@lion tutorial]$ ls -a .hidden
```

Zauważmy, że skróty, których użyliśmy wcześniej, `.` oraz `..`, również pojawiają się tak, jakby były prawdziwymi katalogami.

Jeżeli idzie o nasze ostatnio zainstalowane polecenie `tree`, to działa ono w podobny sposób (z wyjątkiem braku pojawienia się `.` oraz `..`):

```
[bob@lion tutorial]$ tree
[bob@lion tutorial]$ tree -a
```

Wróćmy do naszego katalogu domowego (`cd`) i spróbujmy uruchomić `ls` bez, a następnie z przełącznikiem `-a`. Przekierujmy wyjście przez `wc -l`, aby dać wyraźniejszy obraz tego, jak wiele ukrytych plików i folderów znajduje się w naszym katalogu domowym. Pliki te zazwyczaj przechowują naszą osobistą konfigurację i w ten sposób systemy uniksowe zawsze oferowały możliwość posiadania ustawień na poziomie systemu (zwykle w katalogu `/etc`), które mogą być nadpisane przez poszczególnych użytkowników (dzięki ukrytym plikom w ich katalogach domowych).

Zazwyczaj nie powinniśmy mieć do czynienia z ukrytymi plikami, ale czasami instrukcje mogą wymagać od na wejścia do katalogu `.config`, lub edycji jakiegoś pliku, którego nazwa zaczyna się od kropki. Przynajmniej teraz będziemy rozumieć co się dzieje, nawet jeżeli nie możemy łatwo zobaczyć pliku w swoich narzędziach graficznych.

Sprzątanie

Dotarliśmy do końca tego samouczka, i powinniśmy być teraz z powrotem w naszym katalogu domowym (użyjmy polecenia `pwd`, aby to sprawdzić, i polecenia `cd`, aby przejść tam, jeżeli nie jesteśmy). Dobrze jest zostawić nasz komputer w takim samym stanie, w jakim go zastaliśmy, więc jako ostatni krok usuńmy obszar eksperymentalny, którego używaliśmy wcześniej, a następnie sprawdźmy dwukrotnie, czy rzeczywiście go nie ma:

```
[bob@lion ~]$ rm -r /tmp/tutorial
[bob@lion ~]$ ls /tmp
```

Jako ostatni krok, zamknijmy terminal. Możemy po prostu zamknąć okno, ale lepszą praktyką jest wylogowanie się z powłoki. Możemy użyć albo polecenia `exit`, albo skrótu klawiaturowego `Ctrl-D`. Jeżeli planujemy często korzystać z terminala, to zapamiętanie skrótu `Ctrl-Alt-T` do uruchomienia terminala oraz `Ctrl-D` do jego zamknięcia sprawi, że szybko uznamy terminal za poręcznego asystenta, którego możemy natychmiast przywołać i równie łatwo oddalić.

9 Podsumowanie

Ten tutorial był tylko krótkim wprowadzeniem do linii poleceń Linuksa. Przyjrzeliśmy się kilku popularnym komendom służącym do poruszania się po systemie plików i manipulowania nimi, ale żaden tutorial nie jest w stanie dostarczyć wyczerpującego przewodnika po wszystkich dostępnych komendach. Ważniejsze jest to, że poznaliśmy kluczowe aspekty pracy z powłoką. Zostaliśmy wprowadzeni do szeroko używanej terminologii (i synonimów), którą możemy na-

potkać w sieci, oraz uzyskaliśmy wgląd w niektóre kluczowe części typowego polecenia powłoki. Poznaliśmy ścieżki bezwzględne i względne, argumenty, opcje, strony `man`, polecenie `sudo` i konto `root`, ukryte pliki i wiele innych.

Dzięki tym kluczowym koncepcjom powinniśmy być w stanie nadać więcej sensu wszelkim instrukcjom wiersza poleceń, na które się natkniemy. Nawet jeżeli nie rozumiemy każdego polecenia, powinniśmy przynajmniej mieć pojęcie, gdzie kończy się jedno polecenie, a zaczyna następne. Powinniśmy łatwiej być w stanie powiedzieć, jakimi plikami manipulują, lub jakie inne przełączniki i argumenty są używane. Dzięki odniesieniom do stron `man` możemy nawet być w stanie dowiedzieć się dokładnie, co robi polecenie, lub przynajmniej uzyskać ogólne pojęcie.

Niewiele zostało tu omówione, co mogłoby sprawić, że porzucimy naszego graficznego menedżera plików na rzecz wiersza poleceń, ale manipulacja plikami nie była tak naprawdę głównym celem. Jeżeli jednak jesteśmy zaintrygowani możliwością manipulowania plikami w różnych częściach naszego dysku twardego za pomocą kilku naciśnięć klawiszy, wciąż jest wiele do nauczenia się.

Dalsze lektury

Istnieje wiele tutoriali online i komercyjnie wydanych książek na temat wiersza poleceń, ale jeżeli chcemy zagłębić się w temat dobrym punktem wyjścia może być poniższa książka:

- Wiersz poleceń systemu Linux autorstwa Williama Shottsa

Powodem, dla którego warto polecić tę książkę jest w szczególności to, że została ona wydana na licencji `Creative Commons` i jest dostępna do pobrania za darmo jako plik `PDF`, co czyni ją idealną dla początkujących, którzy nie są pewni, jak bardzo chcą się poświęcić wierszowi poleceń. Jest również dostępna w formie drukowanej, gdyby okazało się, że załapaliśmy się na bakcyla wiersza poleceń i chcemy mieć źródło informacji w wersji papierowej.

10 Zadanie końcowe

1. Przejdź do katalogu ze swoim repozytorium Gita.
2. Utwórz w repozytorium katalog `tutorial` i przejdź do utworzonego katalogu.
3. Ponownie wykonaj polecenia z samouczka.
4. Wróć do katalogu nadrzędnego.
5. Dodaj katalog `tutorial` do śledzenia, zatwierdź zmiany i wyślij na Bitbucket'a.