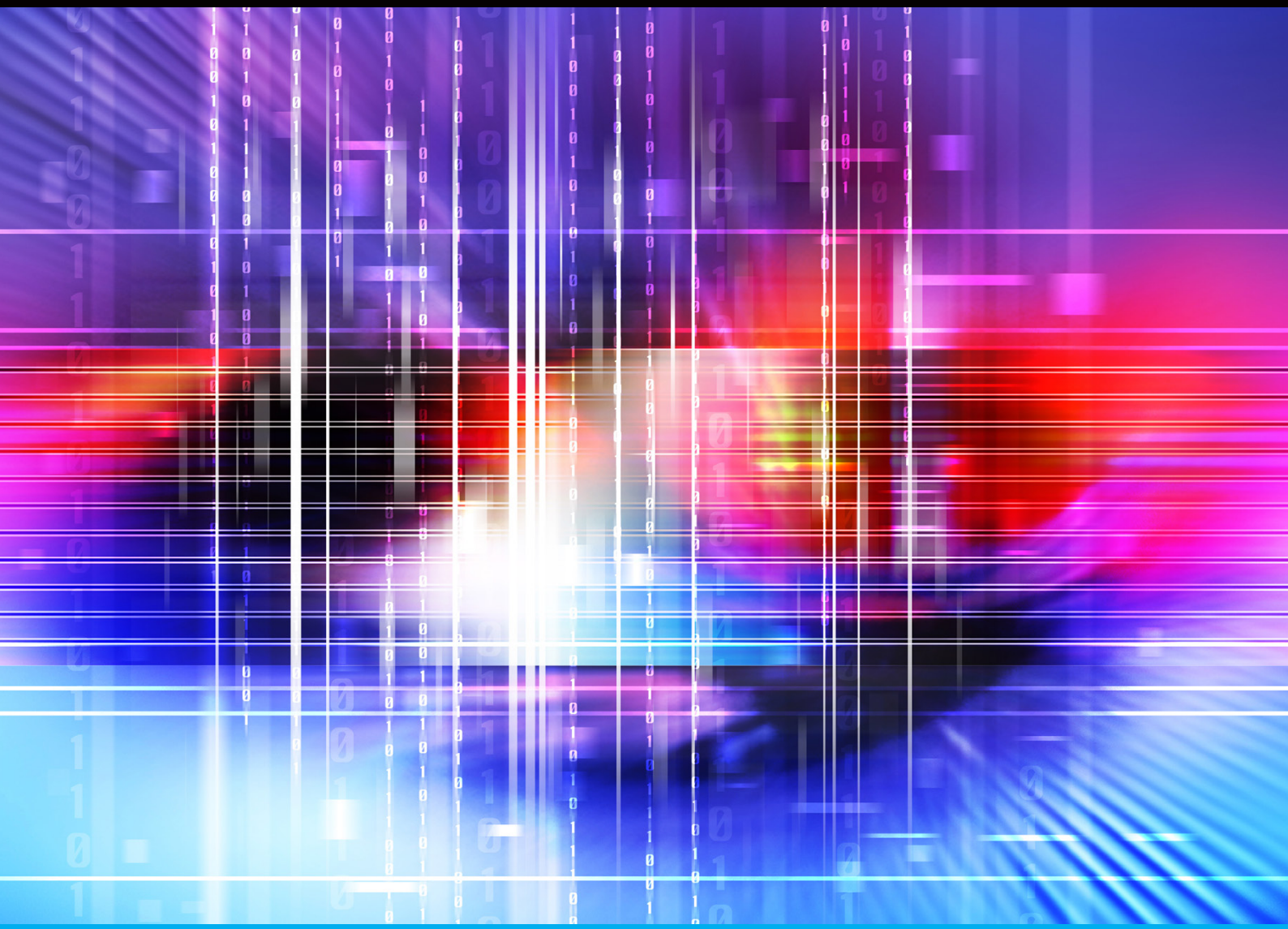
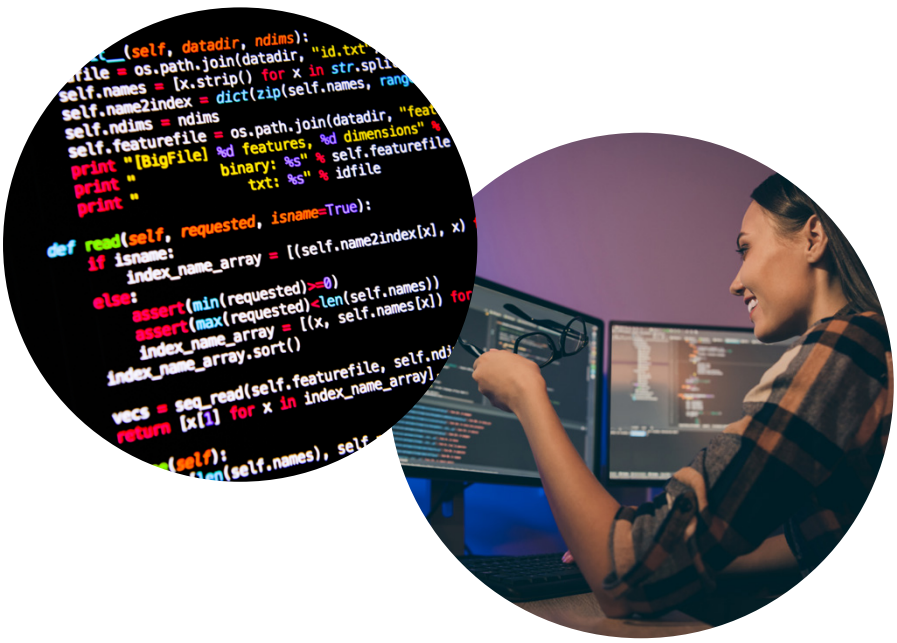
Linux

PRZYDATNE POLECENIA



LINUX MAGAZINE

Spis treści



W trybie tekstowym Linuksa mamy do czynienia z ogromną liczbą niewielkich narzędzi, z których każde zostało zaprojektowane w określonym celu. Są one ściśle wyspecjalizowane, dlatego też zwykle są niewielkie i szybkie; najczęściej też nie wymagają znajomości wielu opcji. Największą produktywność uzyskamy łącząc je w łańcuchy poleceń, by uzyskać żądany rezultat.
Weźmy prosty przykład:

```
ls -al
```

Polecenie wyświetli zawartość danego katalogu, wraz z ukrytymi plikami (*-a*) i w długim formacie (*-l*). Jeżeli znamy angielski, nazwy przełączników będzie nam łatwo zapamiętać: *-a* to skrót od *all* („wszystkie”), zaś *-l* pochodzi od *long* („długi”).
Jest bardzo prawdopodobne, że jeżeli w danym katalogu znajduje się wiele plików, nie uda nam się ujrzeć ich wszystkich, bo ekran przewinie się zbyt szybko. Możemy co prawda użyć kombinacji [Shift] + [PgUp], nie jest to jednak zbyt wygodne rozwiązanie. Dlatego lepiej jest napisać:

```
ls -al | less
```

Polecenie *less* nie tylko pozwoli nam przejrzeć po kolei wszystkie strony wyniku działania *ls*, ale także wrócić do wcześniejszych, a nawet wyszukać interesujące nas fragmenty (naciskamy / a potem wpisujemy interesujący nas ciąg).
Jeżeli z kolei interesują jedynie pliki AVI znajdujące się w bieżącym katalogu, możemy dodać jeszcze jedno polecenie filtrujące, mianowicie *grep*:

```
ls -al  | grep "\.avi" | less
```

W ten sposób połączyliśmy trzy narzędzia: *ls* służące do wyświetlania zawartości katalogów, *grep*, służący do wyszukiwania ciągów znaków i *less* – program do dzielenia na strony wyników działania innych narzędzi.
Każde z tych narzędzie ma wiele różnych opcji. W praktyce jednak nie potrzebujemy ich wszystkich znać na pamięć. W zasadzie nie musimy znać żadnych, bowiem wszystkie wyposażone są w znakomitą pomoc podręczną. Ma ona dwie postaci: krótką i rozbudowaną. Postać krótką uzyskujemy, wywołując polecenie z argumentem *--help*, np.:

```
ls --help
```

Z kolei postać rozbudowaną uzyskujemy poleceniem *man* (od angielskiego *manual*, czyli „podręcznik”):

```
man ls
```

Dlatego też w dalszej części niniejszej książki ograniczymy opisy opcji programów, skupiając się na najbardziej użytecznych. Zainteresowany Czytelnik może bowiem z łatwością sprawdzić pozostałe, korzystając z opisanych wyżej sposobów.

cat

Polecenie *cat* służy do wyświetlania zawartości plików. W najprostszej postaci wywołamy je tak:

```
cat /etc/hosts
```

Powyższa komenda wyświetli po prostu plik z definicjami hostów */etc/hosts*. W tym przypadku *cat* dzieła niemal identycznie jak windowsowe *type*. Jeżeli chcemy wyświetlić też niewidoczne znaki, takie jak koniec wiersza, tabulator itp., użyjemy poniższego zestawu opcji:

```
cat -vet /etc/hosts
```

Ale to nie wszystko. Równie prosto możemy użyć *cata* do tworzenia nowych plików w trybie interaktywnym. W tym celu używamy poniższej konstrukcji:

```
cat > plik.txt
```

Po wydaniu powyższego polecenia przejdziemy do prostego trybu edycji pliku *plik.txt*, który opuścimy kombinacją [Ctrl] + [D] – tę samą, która służy do opuszczania powłoki.
Jeżeli nie chcemy tworzyć pliku od nowa, lecz tylko dopisać coś na jego końcu, użyjemy dwóch trójkątnych prawych nawiasów:

```
cat >> plik.txt
```

Operator >> zadba o to, by pierwotna zawartość nie została nadpisana – cała nowa treść zostanie po prostu dodana na końcu poprzedniej.
Co ciekawe, kombinacja [Ctrl] + [D] nie jest jedyną, którą można zakończyć wpisywanie plików. Zasadniczo możemy użyć jakiegokolwiek ciągu znaków, tyle że musimy go umieścić po operatorze << na końcu wiersza. Najczęściej rolę tego ciągu pełni sekwencja __EOF__ – od angielskiego *End Of File*, czyli „koniec pliku”:

```
cat > plik.txt << __EOF__
```

Polecenie *cat* używane jest też często do łączenia wielu plików. Jeżeli na przykład chcemy połączyć zawartość pliku *plik1.txt* z plikiem *plik2.txt*, użyjemy następującej konstrukcji:

```
cat plik1.txt plik2.txt > plik3.txt
```

Tak jak inne polecenia linuksowe, *cat* idealnie współpracuje z innymi narzędziami, takimi jak wspomniane wcześniej *less* czy *grep*. Jeżeli np. chcemy szybko sprawdzić częstotliwość taktowania zegara w naszych procesorach, użyjemy kombinacji poleceń *cat* i *grep*:

```
cat /proc/cpuinfo | grep Hz
```

Istnieje prosta sztuczka z *cat*, która czasem się przydaje - np. do numerowania wierszy w listingach. Wystarczy dodać przełącznik -n, a zawartośc pliku zostanie wyświetlona wraz z numerami wierszy. Wystarczy teraz przekierować wynik do innego pliku, a otrzymamy ładnie ponumerowany listing:

```
cat -n bez_numerow.php > z_numerami.txt
```

Cat można używać w połączeniu z nazwami urządzeń specjalnych. Przykładowo */dev/urandom* to nazwa specjalnego pliku, stanowiącego interfejs do generatora liczb pseudolosowych. Jeżeli spróbujemy wyświetlić jego zawartość, nasz ekran pokryje się śmieciami:

```
cat /dev/urandom
```

Działanie powyższego polecenia przerywamy więc kombinacją [Ctrl] + [C]. Możemy jednak przekształcić dane wynikowe do czytelnej postaci, filtrując je narzędziem *hexdump* z opcją *-C*:

```
cat /dev/urandom | hexdump -C
```

Z kolei */dev/dsp* to specjalny plik stanowiący interfejs naszej karty dźwiękowej. Innymi słowy, możemy go użyć do nagrywania i odtwarzania. A co stanie się, jeżeli przekierujemy zawartość generatora liczb pseudolosowych na wejście karty muzycznej? Sprawdźmy:

```
cat /dev/urandom > /dev/dsp
```

Nie martwmy się, nie uszkodzimy w ten sposób naszej karty. W głośnikach powinniśmy usłyszeć jednostajny szum (jeżeli nie jest jednostajny... generowane liczby nie są pseudolosowe). Jeżeli mamy podłączony mikrofon, możemy nagrać się poleceniem:

```
cat /dev/dsp > moj_glos
```

...a następnie odsłuchać nagrania poleceniem:

```
cat moj_glos > /dev/dsp
```

Jeżeli mamy mysz skonfigurowaną jako */dev/mouse* (lub np. */dev/psaux*) możemy na bieżąco obserwować surowe dane, kiedy tylko poruszymy urządzeniem:

```
cat /dev/psaux
```

Jeżeli jesteśmy z natury ciekawscy, prędzej czy później będziemy zajrzeć do pamięci systemu. Aby miało to sens, musimy przepuścić strumień danych przez filtr polecenia *hexdump*:

```
cat /dev/mem | hexdump -C | less
```

W analogiczny sam sposób przeszukamy dysk twardy i wszelkie inne urządzenia blokowe. Jeżeli np. kupiliśmy dysk twardy i jesteśmy ciekawi, co na nim przechowywał poprzedni właściciel, możemy - choć oczywiście nie powinniśmy - użyć poniższego polecenia:

```
cat /dev/sda1 | hexdump -C | less
```

cp

Polecenie cp samo w sobie nie jest zbyt interesujące – służy po prostu do kopiowania plików. Zwróćmy jednak uwagę, że można je wykorzystać w kreatywny sposób, wykorzystując możliwości powłoki i pomocniczych narzędzi. Na przykład powłoka Bash rozwija argumenty listy zawarte w nawiasach klamrowych, dopisując je do występującego wcześniej ciągu. Jeżeli więc napiszemy:

```
cp /etc/passwd{,.bak}
```

Polecenie to zostanie zinterpretowane jako:

```
cp /etc/passwd /etc/passwd.bak
```

Jest to przydatne we wszelkiego typu skryptach – możemy łatwo i elegancko manipulować nazwami plików.
Polecenie *tac* działa identycznie jak *cat*, tyle że odwraca kolejność wierszy w pliku.

date

Date służy głównie do wyświetlania i ustawiania daty systemowej, ma też wiele innych, nie mniej ważnych zastosowań.
W podstawowej formie polecenie to wywołujemy bez żadnych argumentów:

```
date
```

Wyświetlona zostanie data w postaci: *nie, 7 mar 2010, 19:24:46 CET*. W praktyce jednak rzadko używamy pełnej formy – najczęściej zależy nam jedynie na bieżącej dacie lub godzinie. Dlatego też używamy łańcuchów formatujących, składających się ze znaku procenta (%) i pojedynczej litery, symbolizującej godzinę (H), minutę (M), sekundę (S), dzień (d), miesiąc (m) czy rok (y). Oczywiście symboli tych jest znacznie więcej (np. N oznacza nanosekundy, zaś B to lokalna nazwa miesiąca), w praktyce jednak najczęściej używane są te wspomniane wcześniej.

Poprzedzone znakiem procenta symbole umieszczamy jako argument po znaku +, np. ciąg 19:31:45 uzyskamy poleceniem:

```
date +%H:%M:%S
```

Jeżeli chcemy wiedzieć, jaka data nastąpi dokładnie za 13 dni, możemy albo zerknąc do kalendarza, albo napisać w terminalu:

```
date --date='13 days'
```

Jeżeli chcemy się dowiedzieć, w jaki dzień tygodnia w tym roku przypadnie wigilia Bożego Narodzenia, użyjemy polecenia:

```
date --date='24 Dec' +%A
```

No dobrze, ale jak ustawimy datę? Służy do tego opcja --set. Jeżeli np. chcemy przestawić zegar o trzy minuty do przodu, napiszemy:

```
date --set='+3 minutes'
```

Następnie synchronizujemy zegar systemowy ze sprzętowym:

```
hwclock --systohc
```

Możemy też podać dokładny czas i datę. Jest to dość uciążliwe, bo musimy pamiętać o formacie, w czym nie pomaga nam dokumentacja. Format składa się wyłącznie z cyfr, kolejno: miesiąc, dzień, godzina, minuta – każda z tych pozycji powinna być liczbą dwucyfrową. Chcąc więc ustawić datę na trzeciego marca, zaś godzinę na dziewiętnastą pięćdziesiąt, napiszemy:

```
date 03071950
hwclock --systohc
```

Jeżeli zaś chcemy dodatkowo zmienić rok, dopisujemy go na końcu (w postaci czterocyfrowej):

```
date 030719502030
hwclock --systohc
```

W skryptach *date* używane jest często do intuicyjnego oznaczania kopii zapasowych plików. Kopię pliku */etc/passwd* z bieżącą datą wykonamy w ten sposób:

```
cp /etc/passwd /etc/passwd.`date +%Y_%d_%m`
```

Możemy tez utworzyć alias lub zmienną, która pozwoli nam łatwo wyświetlić dzisiejszą datę:

```
dzis=$(date "+%d %B %Y")
```

Sprawdźmy, jak to działa:

```
# echo $dzis
08 marzec 2010
```

Jeżeli jako argument *echo* podamy gwiazdkę, polecenie to zachowa się jak *ls* – wyświetli wszystkie pliki w bieżącym katalogu.

dd

Polecenie *dd* używane jest do kopiowania zawartości plików i urządzeń. Za jego pomocą możemy na przykład zrobić pełną kopię partycji i zachować ją w pliku. Ma ono dość nietypową konstrukcję: najpierw podajemy nazwę pliku wejściowego poprzedzoną ciągiem *if=* (od angielskiego *Input File*, czyli „plik wejściowy"), następnie nazwę pliku wyjściowego, poprzedzoną ciągiem *of=* (od angielskiego *Outut File*, czyli „plik wyjściowy"), po czym następują opcjonalne argumenty, takie jak rozmiar bloku (*bs=*, od *Block Size*) czy liczba bloków (*count*).
Jeżeli np. chcemy skopiować partycję */dev/sda3* (trzecia partycja na dysku twardym SCSI lub SATA) do pliku *plik.par*, użyjemy następującej konstrukcji:

```
dd if=/dev/sda of=plik.par
```

Polecenie to może też posłużyć do wykonania kopii głównego rekordu rozruchowego (MBR), czyli pierwszego bloku na dysku twardym, który ma długość 512 bajtów (wymagane są uprawnienia administratora):

```
dd if=/dev/sda of=mbr.par bs=512 count=1
```

Powinniśmy ujrzeć komunikat podobny do poniższego:

```
1+0 przeczytanych recordów
1+0 zapisanych recordów
skopiowane 512 bajtów (512 B), 0,0234193 s, 21,9 kB/s
```

Kopię MBR możemy zachować np. na dysku USB. Jeżeli jakiś złośliwy program (wirus, instalator nowej wersji Windows itp.) uszkodzi nam MBR, będziemy mogli go łatwo odzyskać:

```
dd if= mbr.par of=/dev/sda bs=512 count=1
```

Poręczne polecenie *file* właściwie zidentyfikuje plik po jego zawartości. Jeżeli wydamy polecenie:

```
file mbr.par
```

W rezultacie powinniśmy otrzymać komunikat:

```
mbr.par: x86 boot sector; partition 1: ID=0x83, active, starthead 1, startsector
63, 32049612 sectors; partition 2: ID=0x5, starthead 254, startsector 32049675,
1494045 sectors, code offset 0x63
```

Przed sprzedażą dysku możemy użyć *dd* do wyzerowania wszystkich bitów (urządzeniem /dev/zero) czy też zapełnieniem ich przypadkowymi wartościami:

```
dd if=/dev/urandom of=/dev/sda
```

UWAGA! Powyższe polecenie jest destrukcyjne – nie należy go używać lekkomyślnie. Istnieje też wariant złośliwy: jeżeli wiemy, że przyszły użytkownik dysku będzie chciał zajrzeć do jego poprzedniej zawartości, możemy celowo wprowadzić go w błąd. Jedną z metod jest zastąpienie określonych napisów innymi. Jeżeli chcemy zastąpić wszystkie wystąpienia na dysku twardym imienia „Zosia" napisem „Agata", wydamy następujące polecenie:

```
dd if=/dev/sda | sed ‚s/Zosia/Agata/g' | dd of=/dev/sda
```

W tym przypadku oba imiona mają tę samą długość; drugie jednak może być krótsze – spacje mogą zastąpić brakujące znaki.
Mamy ochotę utworzyć kopię płyty CD? Wystarczy użyć polecenia:

```
dd if=/dev/cdrom of=plik.iso bs=2048 conv=sync,notrunc
```

Plik ISO możemy następnie wypalić ulubionym programem do nagrywania obrazów ISO.
Wreszcie, jeżeli z jakiegoś powodu zależy nam na zapełnieniu całej wolnej przestrzeni dyskowej, użyjemy polecenia:

```
dd if=/dev/urandom of=duży.plik
```

Polecenie *dd* jest też używane do wielu innych rzeczy, takich jak tworzenie pustych plików, które mogą zostać później użyte do innych celów. Dobrym przykładem są pliki wymiany. Brakuje nam RAM-u? Możemy w każdej chwili zwiększyć ilość pamięci, tworząc najpierw pusty plik, np. o rozmiarze 1 GB:

```
dd if=/dev/zero of=/plik.swap bs=1M count=1000
```

Następnie inicjalizujemy go i dodajemy do puli sekwencją poleceń:

```
mkswap /plik.swap
swapon /plik.swap
```

Ilość dostępnej pamięci możemy łatwo sprawdzić poleceniem *free*.
Wreszcie, jeżeli z jakiegoś powodu zależy nam na zamianie wszystkich liter w pliku *plik1.txt* na wielkie, możemy łatwo to zrobić właśnie poleceniem *dd* (rezultat zapiszemy wpliku *plik2.txt*):

```
dd if=plik1.txt of=plik2.txt conv=ucase
```

W podobny sposób użyjemy *dd* do utworzenia dysku w pamięci RAM. Procedura jest prosta: najpierw generujemy pusty plik (np. 10-megabajtowy), korzystając z urządzenia */dev/zero*, a następnie tworzymy na nim system plików poleceniem *mkfs*:

```
dd if=/dev/zero of=/dev/ram1 bs=1M count=10
mkfs.ext3 /dev/ram1
```

Dysk RAM jest gotowy do pracy. Możemy go teraz zamontować:

```
mount /dev/ram1 /tmp
```

Będzie ultra-szybki, nie należy jednak zapominać, że nadal jest to pamięć RAM, której zawartość zostanie wymazana przy zaniku zasilania. Mimo to, jeżeli np. często musimy kompilować złożone projekty, z pewnością docenimy wydajność dysku RAM.
Z kolei pamięć systemową przeszukamy poleceniem:

```
dd if=/dev/mem | hexdump -C | grep "napis"
```

Warto wspomnieć o jeszcze jednym zastosowaniu *dd*. Jeżeli chcemy, by użytkownik nacisnął jeden klawisz ale tylko jeden!), możemy użyć poniższej konstrukcji:

```
a=`dd bs=1 count=1 2>/dev/null`
echo $a
```

W ten sposób pobraliśmy wartość klawisza od użytkownika, przypisaliśmy ją zmiennej $a i wyświetliliśmy na ekranie. Nie da się tego zrobić standardowym poleceniem *read*, które służy do wprowadzania ciągu znaków zakończonych naciśnięciem [Enter].

df

Polecenie df służy do szybkiego sprawdzania zajętości miejsca na wszystkich zamontowanych systemach plików. W najprostszej formie wywołujemy je jako:

```
df
```

Nie jest to jednak zbyt wygodne, dlatego najczęściej używamy przełącznika -h, który sprawi, że wynik działania będzie bardziej czytelne. Porównajmy:

```
# df
```

System plików	bl. 1K B	użyte	dostępne	%uż.	zamont. na
simfs	10485760	1028404	9457356	10%	/
tmpfs	4081056	0	4081056	0%	/lib/init/rw
tmpfs	4081056	0	4081056	0%	/dev/shm

```
# df -h
```

System plików	rozm.	użyte	dostępne	%uż.	zamont. na
simfs	10G	1005M	9,1G	10%	/
tmpfs	3,9G	0	3,9G	0%	/lib/init/rw
tmpfs	3,9G	0	3,9G	0%	/dev/shm

Możemy tez wymusić pokazywanie miejsca w blokach o określonym rozmiarze, np. kilobajtach (K), megabajtach (M), gigabajtach (G) itd., np.:

```
df -B G
```

echo

Polecenie echo służy do wyświetlania tekstu. W najprostszej formie możemy go użyć w poniższy sposób:

```
echo "test"
```

Jeżeli zaś chcemy dopisać do pliku plik1.txt napis test, użyjemy konstrukcji:

```
echo "test" > plik1.txt
```

Jak widzimy polecenie to samo w sobie nie jest zbyt interesujące. Staje się prawdziwie użyteczne w połączeniu z innymi narzędziami. Jednym z nich jest kalkulator bc, który w trybie wsadowym pozwoli nam dokonywać całkiem zasadniczych obliczeń, zaś echo wyświetli je na ekranie, np:

```
echo "123*123" | bc
```

Na koniec jedna uwaga dotycząca przekierowania wyjścia. Tradycja nakazuje umieszczać je na końcu, np.:

```
echo "test" > plik1.txt
```

W rzeczywistości jednak możemy przenieść je na początek:

```
> plik1.txt echo "test"
```

A nawet w środek:

```
echo > plik1.txt "test"
```

Wszystkie trzy polecenia są równoważne. Cudzysłów nie jest obowiązkowy, ale dodawanie go jest dobrym zwyczajem – pomaga unikać problemów z wyrażeniami wielowyrazowymi.

ln

W Linuksie istnieją dwa szczególne typy plików, zwane „dowiązaniami” bądź „odnośnikami” do innych plików: twarde i miękkie (zwane również symbolicznymi). Między nimi istnieje zasadnicza różnica, warto więc dobrze zapamiętać, które są które.
Dowiązania symboliczne to zwykłe odnośniki - same w sobie nie zawierają żadnych danych. Możemy usunąć odnośnik symboliczny, ale plik, na który on wskazuje, pozostanie nienaruszony. Oczywiście możemy tworzyć nieograniczona liczbę odnośników symbolicznych do różnych plików.
Dowiązania twarde mają zupełnie inną naturę: stanowią jak gdyby kopię oryginalnych plików w tym sensie, że po utworzeniu twardego dowiązania do pliku możemy usunąć oryginalny plik, zaś dane nie zostaną utracone, będą one bowiem w dowiązaniu.
Dowiązanie symboliczne tworzymy poleceniem ln z opcją -s. Zwróćmy uwagę, że zawsze najpierw podajemy nazwę oryginalnego pliku, a dopiero na drugim miejscu nazwę dowiązania:

```
ln -s /bin/ls kat
```

W ten sposób utworzyliśmy dowiązanie symboliczne do pliku /bin/ls, które będzie działało dokładnie tak, jak oryginał. Z kolei dowiązanie twarde tworzymy tym samym poleceniem, tyle że bez opcji -s:

```
ln /bin/ls kat
```

Łatwo zorientować się, że dany plik jest dowiązaniem symbolicznym, za pomocą polecenia ls:

```
# ls -l kat
lrwxrwxrwx 1 root root 7 mar  8 09:55 kat -> /bin/ls
```

Zwróćmy uwagę na pierwszy znak wiersza: jest to l, czyli dowiązanie symboliczne (- oznacza zwykły plik, d - katalog, zaś s - gniazdo).
Z kolei dowiązanie twarde wygląda prawie jak zupełnie niezależny plik:

```
# ls -l kat
-rwxr-xr-x 2 root root 92312 kwi  4  2008 kat
```

Jedyna wskazówka dotycząca dowiązania symbolicznego to liczba 2 po polu uprawnień, która informuje nas, że istnieje jedno twarde dowiązanie oprócz oryginalnego pliku. Jednak dopiero dzięki opcji -i polecenia ls możemy zorientować się, że oba pliki dzielą ten sam numer i-węzła, czyli zawierają te same dane:

```
# ls -il kat
2360959 -rwxr-xr-x 2 root root 92312 kwi  4  2008 kat
# ls -il /bin/ls
2360959 -rwxr-xr-x 2 root root 92312 kwi  4  2008 /bin/ls
```

ls

Polecenie ls służy do wyświetlania zawartości katalogów - podobnie zresztą jak dir. Ma ono sporo opcji, jednak w praktyce używa się jedynie kilku z nich.
Jeżeli chcemy wyświetlić zawartość katalogu domowego, napiszemy:

```
ls ~
```

Jeżeli zaś interesuje nas zawartość katalogu nadrzędnego (do obecnego), użyjemy konstrukcji:

```
ls ../
```

Opcje ls można grupować. Przykładowo parametr -a służy do wyświetlania wszystkich plików, również tych ukrytych, -s wyświetli także rozmiary plików, zaś -h użyje formatu łatwego do odczytania przez ludzi:

```
ls -ash
```

Jeżeli często używamy polecenia *ls* z określonym zestawem parametrów, możemy utworzyć alias, czyli rodzaj skrótu. Jeżeli np. zechcemy nadać powyższemu poleceniu nazwę-alias „l", użyjemy konstrukcji:

```
alias l='ls -ash'
```

Odtąd za każdym razem, kiedy napiszemy l, będzie to miało taki sam skutek, jak *ls -ash*.
Często zdarza się, że chcemy wyświetlić wszystkie pliki w danym katalogu w kolejności od najmłodszego do najstarszego (według daty ostatniej modyfikacji). Użyjemy wtedy konstrukcji:

```
ls -alt
```

Kolejność tę możemy odwrócić przełącznikiem *-r*:

```
ls -altr
```

Jeżeli interesuje nas czas utworzenia pliku, a nie modyfikacji, napiszemy:

```
ls -act
```

W taki sam sposób posortujemy pliki według rozmiaru – od największego:

```
ls -alS
```

I od najmniejszego:

```
ls -alSr
```

Z kolei opcja *-R* (od angielskiego *recursive*) wyświetli nie tylko zawartość żądanego katalogu, ale również wszystkich folderów podrzędnych, np. w naszym katalogu domowym:

```
ls -aR ~/
```

Nie pamiętamy, który plik ostatnio edytowaliśmy? Pomoże nam w tym kombinacja poleceń:

```
ls -t | head -1
```

Polecenie ls domyślnie wyświetla pliki w wielu kolumnach. Możemy to zmienić, dodając opcję -1, która nakazuje my wyświetlanie każdego pliku w oddzielnym wierszu, a następnie zapisać powstałą listę w pliku:

```
ls -1 > mojepliki.txt
```

Jeżeli chcemy wyświetlić numeryczne identyfikatory właścicieli plików (użytkowników i grup), użyjemy przełącznika -n:

```
ls -n
```

Jeżeli chcemy wyświetlić same katalogi, napiszemy:

```
ls -ap | grep /
```

Na koniec warto wspomnieć, że od pewnego czasu w Linuksie polecenie *ls* (podobnie jak *ps*) przyjmuje również argument -Z, który wyświetla kontekst bezpieczeństwa SELinuksa dla plików w danym katalogu. Ma to znaczenie jedynie wtedy, kiedy używamy SELinuksa (lub został on domyślnie włączony w używanej przez nas dystrybucji).

mkdir

Polecenie mkdir służy do tworzenia katalogów. W najprostszej postaci wywołamy je jako:

```
mkdir katalog
```

Możemy też użyć go do utworzenia zagnieżdżonej struktury kilku katalogów równocześnie. W tym przypadku użyjemy opcji -p:

```
mkdir -p katalog1/katalog2/katalog3/
```

pwd

Polecenie *pwd* jest bardzo użyteczne – służy do sprawdzania, w którym katalogu dokładnie jesteśmy. Warto jednak wspomnieć o jego dwóch użytecznych opcjach – *-L* i *-P*. Służą one do określenia, czy w nazwie ścieżki mają być – odpowiednio – uwzględniane dowiązania symboliczne (patrz wyżej – polecenie *ln*), czy też nie.
Zróbmy prosty eksperyment: utwórzmy dowiązanie symboliczne */r* wskazujące na katalog */root*:

```
ln -s /root /r
```

Następnie przejdźmy do nowo utworzonego katalogu */r*:

```
cd /r
```

I sprawdźmy, gdzie jesteśmy:

```
# pwd
/r
```

Pwd domyślnie działa jak *pwd -L*. A co w sytuacji, gdy użyjemy przełącznika *-P*?

```
# pwd -P
/root
```

Stąd też użyteczne polecenie, które zawsze przeniesie nas do „prawdziwego” katalogu (z tego, który jest dowiązaniem symbolicznym:

```
cd `pwd -P`
```

Jeżeli interesuje nas katalog roboczy określonego procesu, powinniśmy zainteresować się poleceniem *pwdx* z pakietu *procps*.

sleep

Polecenie *sleep* służy zwykle do wykonania kolejnych poleceń z opóźnieniem. Przyjmuje jeden argument – jest nim liczba sekund, które ma odczekać. W najprostszej formie użyjemy go w ten sposób:

```
sleep 2; echo "Upłynęły już dwie sekundy!"
```

Oczywiście nie musimy się ograniczać do sekund: możemy równie dobrze użyć minut (*m*), godzin (*h*) a nawet dni (*d*).

```
sleep 2d; echo "Upłynęły już dwa dni!"
```

Niektórzy użytkownicy używają *sleep* w roli budzika. Spójrzmy na poniższy przykład:

```
sleep 1h && echo "Czas wysłać raport" &
```

Wiemy, że za godzinę mamy wysłać raport. Wpisujemy więc powyższe polecenie, które zostanie wykonane w tle (znak & na końcu wiersza): polecenie *echo* uruchomione zostanie więc po godzinie, wyświetlając na ekranie odpowiedni napis.

Zwróćmy uwagę na operator && oddzielający oba polecenia. Gdybyśmy użyli średnika, nie otrzymalibyśmy żądanego rezultatu: najpierw wykonane zostałoby polecenie *sleep*, a później polecenie *echo* zostałoby przeniesione w tło, co rzecz jasna w tym przypadku nie ma większego sensu.
Oczywiście używanie *sleep* jako przypominajki jest przykładem dość ekstremalnym – jeżeli faktycznie zależy nam na wykonywaniu określonych zadań w wyznaczonym czasie, powinniśmy zainteresować się pakietem *at*. Zainstaluje on demona *atd* służącego do wykonywania zadań z opóźnieniem i program *at*, który pozwoli nam wykonywać polecenia o określonym czasie.

stty

Polecenie *stty* jest bardzo użyteczne, powinniśmy jednak zachować dużą ostrożność w manipulowaniu nim, ponieważ służy między innymi do zmiany ustawień emulatora terminala. Innymi słowy, możemy czasem zupełnie odciąć się od terminala.

Spójrzmy na prostą sztuczkę, która sprawi, że wpisywane z klawiatury znaki przestaną pojawiać się na terminalu:

```
stty -echo
```

Jak wrócić do poprzedniego stanu? Za pomocą polecenia:

```
stty echo
```

Sztuczka ta jest przydatna wszędzie tam, gdzie pojawianie się znaków wpisywanych z klawiatury jest złym pomysłem, np. przy wprowadzaniu haseł. Załóżmy, że chcemy poprosić użytkownika o wprowadzenie hasła, a następnie je wyświetlić:

```
read $haslo; echo "Twoje hasło to" $haslo
```

Jak możemy łatwo się przekonać, hasło pojawi się na ekranie jeszcze wtedy, kiedy użytkownik będzie je wpisywał. Problem ten ominiemy używając naszej konstrukcji z stty:

```
stty -echo ; read $haslo; echo "Twoje hasło to" $haslo; stty -echo
```

Bieżące ustawienia terminala wyświetlimy poleceniem:

```
stty -a
```

Mogą one np. wyglądać następująco:

```
speed 38400 baud; rows 44; columns 114; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>; eol2 = <undef>;
swtch = <undef>; start = ^Q;
stop = ^S; susp = ^Z; rprnt = ^R; werase = ^W; lnext = ^V; flush = ^O; min = 1;
time = 0;
-parenb -parodd cs8 -hupcl -cstopb cread -clocal -crtscts
-ignbrk brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr icrnl ixon -ixoff -iuclc -ixany
imaxbel -iutf8
opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0 cr0 tab0 bs0 vt0 ff0
isig icanon iexten echo echoe echok -echonl -noflsh -xcase -tostop -echoprt
echoctl echoke
```

Znak ^ oznacza naciśnięcie klawisza [Ctrl]. Tak więc kombinacją przerwania wykonywania programu (*intr*) jest [Ctrl]+[C], do zawieszenia wykonywania (przeniesienia w tło, *susp*) służy kombinacja [Ctrl] + [Z] itd. Wszystkie pozostałe parametry opisane zostały w dokumentacji (*man stty*).

Jeżeli chcemy zachować bieżące ustawienia (by móc je łatwo przywrócić), możemy użyć następującej konstrukcji:

```
stty -g
```

Skondensuje ona listę wyświetlonych wyżej parametrów w krótszej postaci, np.:

```
2502:5:bf:8a3b:3:1c:7f:15:4:0:1:0:11:13:1a:0:12
:f:17:16:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0
```

Cały proces zachowywania i przywracania ustawień może wyglądać następująco:

```
zachowaj=`stty -g`
# modyfikujemy ustawienia do woli, a na koniec:
stty $zachowaj
```

Na koniec warto wspomnieć, że w razie jakichkolwiek problemów z terminalem, możemy zawsze użyć polecenia, które przywróci sensowne wartości

```
stty sane
```

touch

Polecenie *touch* służy zasadniczo do zmiany czasu dostępu lub modyfikacji danego pliku. W praktyce jednak często wykorzystuje się jego domyślne zachowanie, które polega na tym, że jeżeli dany plik nie istnieje, zostanie on utworzony. Tak więc polecenie:

```
touch plik.txt
```

...spowoduje, że jeżeli w bieżącym katalogu nie ma pliku o nazwie *plik.txt*, zostanie on utworzony. Jeżeli faktycznie chcemy zmienić datę pliku, możemy użyć przełącznika *-d* i daty w formacie, który został omówiony przy opisie polecenia *date*, np.:

```
touch --date=”next Thursday” plik.txt
```

W ten sposób nadamy plikowi datę (ostatniej modyfikacji, *mtime*, i użycia, *utime*) w przyszły czwartek. Zwróćmy jednak uwagę, że data utworzenia (*ctime*), a raczej ostatniej zmiany struktury i-węzła, nadal jest niezmieniona. Możemy ją łatwo odkryć poleceniem:

```
ls -l --time=c plik.txt
```

Albo w skrócie:

```
ls -lc plik.txt
```

Niestety, w Linuksie nie istnieją żadne standardowe narzędzia pozwalające zmienić *ctime*. Jeżeli z jakiegoś powodu chcielibyśmy to zrobić, musimy przestawić zegar systemowy, a następnie użyć na pliku polecenia takiego jak *chmod*, po czym przywrócić właściwą datę. Możemy też utworzyć plik w taki sam sposób:

```
date 03110000
touch 03110000 plik.txt
date 03101440
```

Warto też zwrócić uwagę, że *ctime* w rzeczywistości nie jest datą utworzenia pliku, lecz datą zmiany struktury i-węzła danego pliku. W Linuksie nie da się ustalić, kiedy faktycznie dany plik został utworzony, bowiem każda zmiana właściciela czy praw dostępu do pliku zmienia jego *ctime*.

Wszystkie trzy czasy wyświetlimy poleceniem *stat*. Wynik jego działania może być podobny do poniższego:

```
File: 'plik.txt'
Size: 0               Blocks: 0          IO Block: 4096   pusty zwykły plik
Device: edh/237d      Inode: 23076874    Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2010-03-11 00:00:00.000000000 +0100
Modify: 2010-03-11 00:00:00.000000000 +0100
Change: 2010-03-10 14:46:39.000000000 +0100
```

Podsumujmy: czas dostępu (*atime*) modyfikowany jest za każdym razem, kiedy dane należące do pliku są odczytane, czas modyfikacji (*mtime*) zmienia się , kiedy następuje zapis do pliku (bez względu na to, czy dane są zmieniane, czy nie), natomiast czas zmiany struktury (*ctime*) zmienia się, kiedy modyfikacji ulegają dane i-węzła, takie jak np. właściciel pliku czy prawa dostępu do niego.

uname

Polecenie *uname* służy do wyświetlania podstawowych informacji na temat naszego komputera. Wywołane z parametrem -a wyświetli najwięcej informacji:

```
# uname -a
Linux openvz-01-91-205-74-245 2.6.24-9-pve #1 SMP PREEMPT Tue Nov 17
09:34:41 CET 2009 i686 GNU/Linux
```

Wszystkie te informacje możemy uzyskać oddzielnie za pomocą odpowiednich przełączników *uname*. Najpierw mamy nazwę jądra (*-s*), następnie sieciową nazwę systemu (*-n*), potem numer wydania (*-r*) i wersji (*-v*) systemu, po czym następuje nazwa architektury (*-m*) i systemu operacyjnego (*-o*).
Zwróćmy jednak uwagę, że w ten sposób ominięta została nazwa platformy sprzętowej i procesora. Informacje te uzyskamy łatwo poleceniem:

```
cat /proc/cpuinfo
```

nice

Nie wszystkie procesy w systemie są równe. Niektóre uruchamiane są z wyższym, a niektóre z niższym priorytetem. Priorytet ten reprezentują liczby od +19 do -20. Aby nieco pokomplikować sprawę, w kontekście polecenia nice im wyższy priorytet, tym mniejsza liczba. Tak więc zadanie o najwyższym priorytecie będzie miało wartość -20.

Jak obejrzeć bieżące wartości nice dla uruchomionych procesów? W tym celu możemy użyć polecenia ps. Nakażemy mu wyświetlić wszystkie uruchomione procesy z następującymi danymi: identyfikator procesu (PID), stan, wartość nice i polecenie, które uruchomiło dany proces:

```
ps -eo pid,state,nice,args
```

Początek wyniku działania tego polecenia będzie wyglądał mniej więcej tak:

PID	S	NI	COMMAND
1	S	0	init [2]
22	S	0	[init-logger]
333	S	0	/usr/sbin/rsyslogd -c3
348	S	0	/usr/sbin/sshd
/etc/xsp2 --nons			
670	S	0	/usr/lib/postfix/master
705	S	0	qmgr -l -t fifo -u
709	S	0	/usr/sbin/vsftpd
728	S	0	/usr/sbin/cron
750	S	0	/usr/sbin/apache2 -k start

Jak widzimy, trzecia kolumna, oznaczona symbolem NI, prezentuje wartość nice. Jeżeli chcemy zmienić wartość priorytetu procesu, np. żeby przyspieszyć jego działanie, użyjemy następującego polecenia (jako administrator):

```
nice -18 make
```

W tym przypadku polecenie make zostanie uruchomione wysokim priorytetem -18.
W każdym przypadku możemy zmienić priorytet uruchomionego już procesu poleceniem renice. Np. aby zwiększyć do -2 priorytet procesu bash o identyfikatorze 712, użyjemy polecenia:

```
renice -2 -p 712
```

Co ciekawe, możemy w ten sposób zmieniać priorytety wielu procesów, np. grup (-g) czy nawet pojedynczych użytkowników (-u):

```
renice -2 -u root
```

who

Zasadniczo polecenie *who* służy do wyświetlania, kto jest zalogowany na danej maszynie. Przełącznik *-H* doda nagłówki kolumn, dzięki czemu łatwiej będzie się zorientować, co oznaczają poszczególne pola:

```
# who -H
UŻYTKOWNIK TERM       CZAS           KOMENTARZ
root     pts/0        2010-03-12 08:58 (62.213.129.212)
```

Polecenie to oferuje jednak znacznie więcej. Możemy go użyć zamiast *uptime* do określenia czasu ostatniego restartu systemu:

```
# who -b
         system boot  2010-02-14 08:11
```

Z kolei przełącznik *-r* wyświetli bieżący poziom uruchomieniowy. Możemy też wyświetlić wiele innych użytecznych informacji przełącznikiem *-a* (i ponownie *-H* pozwoli nam zorientować się, co jest czym):

```
who -aH
```

Polecenie who zignoruje dowcipne dopiski, takie jak „mum likes” (które w połączeniu z *who* tworzą zapytanie „Kogo kocha mamusia?”):

```
who mum likes
```

Zasadniczo do *who* można dodać dowolne dwa wyrazy, a polecenie nie zgłosi błędu. Bardziej ubogą alternatywą dla *who* jest polecenie *users* – nie przyjmuje ono żadnych argumentów.

pinky

Jeżeli używamy czasem polecenia who, zapewne spodoba nam się jego odmiana o nazwie *pinky*.

dircolors

Domyślnie polecenie *ls* nie oznacza plików i katalogów różnymi kolorami. Jeżeli jednak wywołamy je z opcją *--color*, nasz terminal nabierze życia. Do ustawiania kolorów służy zmienna LS_COLORS.

Listę domyślnych kolorów dla różnych typów plików wyświetlimy poleceniem:

```
dircolors -p
```

Listę tę możemy zapisac w formacie gotowym do wykonania przez powłokę poleceniem:

```
dircolors -b > ~/.kolorki_ls
```

Następnie wczytujemy je:

```
. ~/.kolorki_ls
```

Po czym wywołujemy *ls* z opcją wyświetlania koloru:

```
ls --color
```

Przyjrzyjmy się składni zmiennej *LS_COLORS*. Oto co pokaże nam polecenie *dircolors*:

```
LS_COLORS='no=00:fi=00:di=01;34:ln=01;36:pi=40;33:so=01;35:do=01;35:bd=
40;33;01:cd=40;33;01:or=40;31;01:su=37;41:sg=30;43:tw=30;42:ow=34;42:st=
37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.svgz=01;31:*.arj=01;31:*.taz=01;31:*.
lzh=01;31:*.lzma=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.
bz2=01;31:*.bz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.
jar=01;31:*.rar=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.
rz=01;31:*.jpg=01;35:*.jpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.
pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.
tiff=01;35:*.png=01;35:*.svg=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.
mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.ogm=01;35:*.mp4=01;35:*.
m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.
asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.
gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.aac=00;36:*.
au=00;36:*.flac=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.
mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:';
export LS_COLORS
```

Jak widzimy, składa się ona z pól oddzielonych dwukropkiem. Każde pole składa się z nazwy typu pliku, znaku równości i przypisanych temu plikowi wartości kolorów. Niektóre popularne typy plików to:

no	zwykły tekst (nie plik)
fi	zwykły plik
di	katalog
ln	dowiązanie symboliczne
pi	potok
so	gniazdo uniksowe
bd	urządzenie blokowe
cd	urządzenie znakowe
ex	plik wykonywalny
mi	brakujący plik
or	osierocone dowiązanie symboliczne

Dalej mamy rozszerzenia – wiele z nich ma swój własny kolor. Kolory zaś mogą składać się z jednej lub więcej wartości oddzielonych średnikami, według następującego wzoru:

0	przywrócenie koloru domyślnego
1	rozjaśnienie koloru
4	podkreślenie tekstu
5	migotanie tekstu
30	czarne tło
31	czerwony tekst
32	zielony tekst
33	żółty lub brązowy tekst
34	niebieski tekst
35	purpurowy tekst
36	cyjanowy tekst
37	biały lub szary tekst
40	czarne tło
41	czerwone tło
42	zielone tło
43	żółte lub brązowe tło
44	niebieskie tło
45	purpurowe tło
46	cyjanowe tło
47	białe lub szare tło

Jeżeli więc chcemy, by katalogi były wyświetlane jako czerwony tekst na zielonym tle, w pliku *kolorki_ls* przypiszemy *di* wartość *32;41*.

du

Polecenie *du* pozwala ocenić, ile przestrzeni dyskowej zajmują określone katalogi. Domyślnie *du* wyświetli dane dotyczące zawartości bieżącego katalogu. Najlepiej uruchomić je z przełącznikiem *-h*, wtedy łatwiej będzie odczytać wyświetlane wartości, bowiem dodane zostaną nazwy jednostek (K, M, G). Jeżeli chcemy, by *du* wyświetliło informacje dotyczące plików, a nie tylko katalogów, użyjemy przełącznika *-a*:

```
du -ha
```

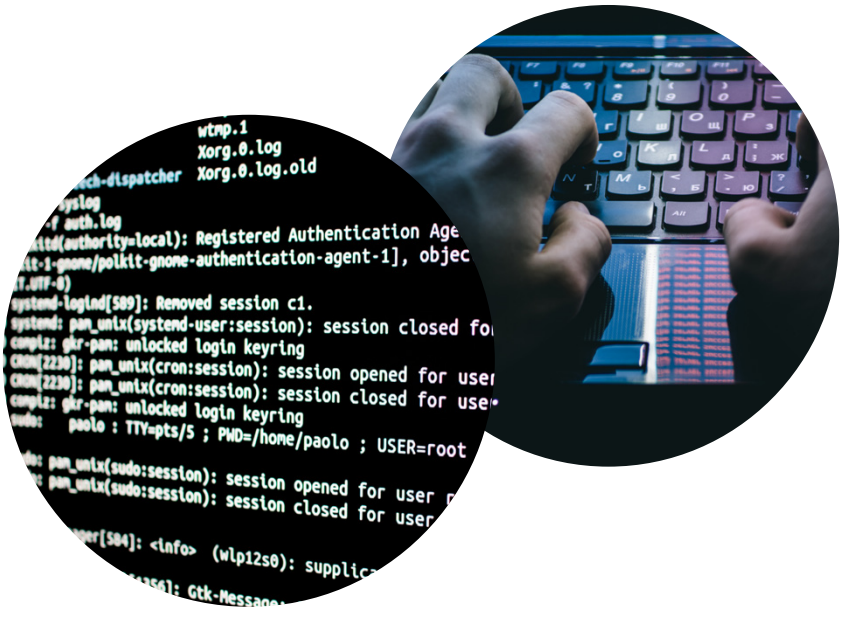
Podsumowanie wyświetlimy opcją *-c*, np.:

```
# du -hc
4,0K    ./.wapi
4,0K    ./.debtags
4,0K    ./.mc/cedit
24K     ./.mc
8,0K    ./.links2
4,0K    ./.aptitude
2,7M    .
2,7M    razem
```

Jeżeli chcemy szybko sprawdzić, ile zajmują wszystkie pliki i katalogi w naszym systemie, użyjemy konstrukcji argumentu *--max-depth=1*. Spowoduje on, że *du* nie będzie wyświetlało zawartości podkatalogów. W rezultacie otrzymamy wynik podobny do poniższego:

```
# du -h --max-depth=1 /
0       /proc
3,7M    /bin
4,0K    /media
4,0K    /srv
163M    /home
2,7M    /root
6,4M    /lib
347M    /var
4,0M    /etc
4,0K    /selinux
```

```
16K     /tmp
4,0K    /boot
4,0K    /opt
3,3M    /sbin
0       /sys
451M    /usr
4,0K    /dev
4,0K    /mnt
981M    /
```



2LM0002