

Procesy w systemie UNIX¹

Proces- ciąg działań podjętych przez CPU komputera pod kontrolą jądra systemu operacyjnego.

Wielozadaniowość- UNIX jest systemem wielozadaniowym, gdyż może "jednocześnie" obsługiwać wiele procesów.

ps- informacja o aktualnie uruchomionych procesach z bieżącego terminala.

1

Parametry:

- *ps -u nazwa_użytkownika* - procesy uruchomione przez konkretnego użytkownika.
- *ps -t numer_terminalu* - procesy uruchomione na danym terminalu.

sleep 20- przerwanie na 20 s komunikacji z shellem.

Sygnały przerywające działanie procesu:

SYGNAŁ	NAZWA	REAKCJA	KLAWISZE
2	int	przerwanie działania procesu	[Del] lub [Ctrl+C]
3	quit	przerwanie procesu wraz z zapamiętaniem jego statusu	[Ctrl+\]
9	kill	bezwzględne przerwanie procesu	
15	terminate	zakończenie działania procesu	

Podstawowe komendy i polecenia w systemie UNIX

cmp- porównywanie plików.

cat- przekierowania strumienia znaków z klawiatury do pliku. Np. *cat > list*. Efektem takiej komendy będzie przekierowanie wpisanych znaków do pliku: list. Za pomocą tej komendy można również przeglądać istniejące pliki.

cd- komenda służąca do zmiany katalogu.

Parametry:

- *cd* - przejście do głównego katalogu użytkownika.
- *cd /* - przejście do katalogu głównego systemu (root).
- *cd ..* - przejście do katalogu nadrzędnego.
- *cd ../../* - przejście o dwa katalogi nadrzędne.
- *cd ~/kantor/strona* - przejście do katalogu strona użytkownika kantor.

¹ http://galaxy.uci.agh.edu.pl/~o_iwona/lab-strona-exe/teksty/unix.html

chmod - polecenie zmieniające prawa dostępu do plików/katalogów.

chmod (-R)	[kto]	[operator]	[prawa]	plik/katalog
	u	+	r	
	g	-	w	
	o (a)	=	x	

Objaśnienia:

-R - dopisujemy, gdy zmieniamy prawa dla katalogu, u (user) - użytkownik, g (group) - grupa, o(a) (all) - wszyscy

+ - dodanie praw, - - odjęcie praw, = - nadanie praw

r (read) - czytanie, w (write) - pisanie, x (execute) - wykonywanie

X - prawo wykonywania tylko wtedy, gdy plik jest katalogiem

cp- polecenie kopiowania. Składnia wygląda tak: *cp ścieżka nazwa pliku gdzie*.

df- informacja na temat ilości miejsca na dysku.

finger - informacja o wszystkich zalogowanych użytkownikach.

finger nick - informacja o użytkowniku np. *finger kantor*. W naszym systemie można również zamiast logina podać nazwisko.

hostname - wyświetlenie nazwy hosta.

komenda|more - polecenie powodujące podzielenie wyników na "ekrany".

ls - wyświetlenie zawartości katalogu, pliki + katalogi.

2

Parametry:

- -l - wyświetlenie plików i katalogów z informacją o prawach dostępu.
- -a - wyświetlenie wszystkich plików i katalogów, również ukrytych.
- -la - wyświetlenie wszystkich plików i katalogów, również ukrytych wraz z prawami dostępu.
- * - wyświetlenie hierarchii katalogów i znajdujących się w nich zbiorach (plikach, katalogach).

mail nazwa konta - wywołanie ekranu inicjującego wysłanie e-maila np. *mail lucas@poczta.fm*. W naszym systemie w celu wysłania e-maila do osoby mającej konto na serwerze student.agh.edu.pl wystarczy komenda mail login np. *mail kantor*.

Zapisujemy używając: *~w nazwa pliku*.

Wychodzimy (nie zapisując, ani nie wysyłając wiadomości) poprzez: *~q*.

w celu wysłania e-mail'a, który znajduje się w pliku należy wpisać: *mail < NAZWA PLIKU*.

Innym wygodnym programem do obsługi poczty jest program pine

man komenda - wywołanie opisu pomocy na temat danej komendy (z ang. *manual*)

mv - polecenie zmiany nazwy/przeniesienia pliku. Składnia wygląda tak: *mv stara nazwa nowa nazwa*

lub *mv nazwapliku nowa lokalizacja*.

passwd - ustawienie lub zmiana hasła.

ps - lista aktywnych procesów.

pwd - aktualna ścieżka, w której znajduje się użytkownik.

mkdir - tworzenie katalogu.

rm - usuwanie pliku/katalogu.

rmdir - usuwanie pustego katalogu.

W sieci wydziałowej (na serwerze krokus) istnieje możliwość wysłania wiadomości do zalogowanego użytkownika
Wpisując: *send "Wiadomość" użytkownik (numer albumu studenta)* Np. *send "Co Ty tu robisz?" s116114*

3

Edytor VI

Edytor VI jest edytorem, który może pracować na każdym typie terminala; nawet na terminalach, które nie umożliwiają realizacji edycji pełnoekranowej - w takim przypadku VI pracuje jako edytor liniowy. VI jest dostarczane wraz z systemem operacyjnym UNIX (LINUX); oznacza to, że na jakimkolwiek sprzęcie UNIX'owym byśmy nie pracowali VI jest zawsze dostępne. Niezależność edytora VI od klawiszy funkcyjnych powoduje, że użytkownik nie napotyka na żadne kłopoty jeżeli kiedyś zmuszony będzie korzystać z jakiejś "egzotycznej" klawiatury.

Organizacja edycji

Edycja tekstu za pomocą edytora VI odbywa się na dwóch płaszczyznach. Płaszczyzna wyższa zwana płaszczyzną komend to ten poziom edycji, na którym znajdujemy się zaraz po uruchomieniu edytora. Na płaszczyźnie tej klawisze literowe mają swoje znaczenia funkcyjne, a nie są po prostu literami. Aby przejść do płaszczyzny niższej zwanej płaszczyzną wprowadzania należy wcisnąć na poziomie komend jedną z liter powodujących przejście do wprowadzania. Klawisze te to *a, i, o, O*, a ich znaczenie zostanie omówione za moment. Powrót z płaszczyzny wprowadzania na płaszczyznę edycji następuje poprzez wcisnięcie klawisza ESC.

a - append (dopisuje tekst za kursorem)

i - insert (dopisuje tekst przed kursorem)

o - open (dopisuje tekst w nowej linii)

Dobrym sposobem przetestowania na której płaszczyźnie jesteśmy jest wcisnięcie klawisza ESC. W przypadku gdybyśmy znajdowali się na płaszczyźnie komend rozlegnie się brzęczyk. Za pomocą VI można edytować jednocześnie wiele plików. Jednakże jednorazowo na ekranie przetwarzany jest tylko jeden. Aby po wywołaniu "VI file1 file2 file3" przejść z edycji pliku file1 do edycji pliku file2 należy z płaszczyzny komend użyć polecenia *:n*. W przypadku gdy

chcemy dołączyć do listy przetwarzanych plików nowy plik używamy komendy :e. Informacje o tym jakie pliki są aktualnie edytowane uzyskamy poprzez komendę :args. Przenoszenie tekstu pomiędzy plikami odbywa się poprzez bufory robocze (może ich być do 26-ciu)

Szybkie poruszanie się po tekście umożliwiają znaczniki tekstu (w jednym pliku do 20 znaczników). Często używaną sekwencję klawiszy możemy zdefiniować jako makro klawiszowe. Sekwencji takich w VI może być dowolna ilość. Podział pliku na zdania, ustępy i sekcje pozwala na bardzo efektywnie przetwarzanie tekstu. Przegląd podstawowych poleceń VI dla początkującego użytkownika:

WYWOŁANIE VI: *VI*

OPUSZCZENIE VI:

Z zapamiętaniem zmian: *ZZ*

Bez zapamiętania zmian: *q!*

ZAPAMIĘTANIE ZMIAN W TRAKCIE EDYCJI: *:w*

WYKONANIE KOMENDY SHELLA: *:!*

STRONICOWE PRZEGLĄDANIE EKRANU:

do przodu: *d*

do tyłu: *u*

PORUSZANIE SIĘ PO PLIKU:

w lewo: *h*

w prawo: *j*

do góry: *k*

na dół: *l*

na początek linii: *0,^*

na koniec linii: *\$*

do linii o numerze #: *#G* (np 1G 20G 250 itd.)

DOPISYWANIE TEKSTU:

na prawo od kursora: *a*

na lewo od kursora: *i*

poniżej linii z kursorem: *o*

powyżej linii z kursorem: *O*

KASOWANIE TEKSTU:

pojedynczego znaku: *x*

pojedynczego słowa: *dw*

całej linii: *dd*

do końca linii: *d\$*

do początku linii: *d0*

POPRAWIANIE TEKSTU:

pojedynczego znuku: *r*
pojedynczego słowa: *cw*
całej linii: *cc*

ODWOŁYWANIE ZMIAN (UNDO):

odwołanie ostatniego polecenia: *u*
wszystkich poleceń: *:e!*

SZUKANIE ŁAŃCUCHA ZNAKÓW:

do przodu: */*
do tyłu: *?*

ZNACZNIKI TEKSTU:

postaw znacznik *#* na bieżącej *m#* (*#* = *a..m* np *ma*, *mb* itd.) pozycji kursora
ustaw kursor na znaczniku *#* (*#* = *a..m* np *'a'*, *'b'* itd.)

ZAPAMIĘTYWANIE TEKSTU DO OBSZARU POMOCNICZEGO (YANK):

pojedynczego słowa: *yw*
całej linii: *yy*
do końca linii: *y\$*

DOPISYWANIE TEKSTU Z OBSZARU POMOCNICZEGO (PUT):

po kursorze: *p*
przed kursorem: *P*

WŁĄCZENIE NUMERACJI LINII: *:set number*

WYŁĄCZENIE NUMERACJI LINII: *:set nonumber*

Podstawowe komendy LINUXa²

Pomoc

man – wyświetla stronę manuala dla polecenia 'program'

- *man program*

info – podobnie jak man, wyświetla stronę pomocy dla polecenia 'program'.

Gdy nie znajdzie strony info, szuka strony man i ją wyświetla.

- *info program, pinfo program*

--help – każdy program ma opcję --help lub -h, która wyświetla krótką pomoc

- *jakis_program --help*

Podstawowe komendy

cat – wypisuje wszystkie podane mu pliki na standardowe wyjście

- *cat plik* – jeśli nie przekierujemy standardowego wyjścia do innego pliku (>, >>) lub programu (|), to wypisze plik na ekran
- *cat plik1 plik2 plik3* – wypisze po kolei zawartość wszystkich plików

tac – wypisuje wszystkie podane mu pliki na standardowe wyjście, ale w odwraca kolejność linii

- *tac plik1 plik2* – wypisze połączone oba pliki, od ostatniej do pierwszej linii

echo – powtarza na standardowym wyjściu słowa podane w argumentach

- *echo costam wypisz, czy echo "costam wypisz"* – wypisze 'costam wypisz' i zakończy znakiem nowej linii
- *echo -n "costam wypisz"* – po wypisaniu argumentów, nie wypisze znaku nowej linii

² <http://www.astrouw.edu.pl/~jskowron/pracownia/komendy/>

wc – liczy linie, słowa, i znaki w pliku

gdy nie podamy argumentu, czyta ze standardowego wejścia

- *cat plik1 plik2 / wc -l* – policzy wszystkie linie z połączonych plików plik1 plik2
- *wc plik* – wypisze linie słowa i znaki oraz nazwę pliku
- *wc -m* – tylko znaki (lub *--chars*)
- *wc -l* – tylko linie (lub *--lines*)
- *wc -w* – tylko słowa (lub *--words*)

less – wygodne i szybkie przeglądanie plików tekstowych

- *less plik* – wyświetla zawartość pliku i pozwala przewijać strony (q-wyjście)

Pliki i katalogi

touch – zmienia czas dostępu i modyfikacji pliku, lub jeśli plik nie istnieje - tworzy go.

- *touch plik*

cp – kopiuje plik

- *cp plik1 plik2* – stworzy ./plik2 identyczny z plik1
- *cp plik3 ../katalog/jakis/* – stworzy plik ../katalog/jakis/plik3
- *cp pom.* podkatalog/* – skopiuje wszystkie pliki zaczynające się na 'pom.' do ./podkatalog/
- *cp plik5 ~/katalog/jakis/pliczek* – stworzy plik ~/katalog/jakis/pliczek

mv – przesuwa plik (tym samym służy również do zmiany nazwy)

- *mv plik1 plik2* – zmieni nazwę pliku z ./plik1 na plik2
- *mv plik3 ../katalog/jakis/* – przesunie plik do ../katalog/jakis/plik3
- *mv plik4 podkatalog/* – przesunie plik ./podkatalog/plik4
- *mv plik5 ~/katalog/jakis/pliczek* – przesunie i zmieni nazwę ~/katalog/jakis/pliczek

rm – kasuje plik

- *rm plik* -
- *rm -r katalog* – kasuje wszystko w katalogu i wszystkie jego podkatalogi (--recursive)
- *rm -f plik* – nie pyta się czy skasować (--force)

mkdir – tworzy katalog

- *mkdir moj_nowy_katalog*
- *mkdir /home/users/ja/voj_nowy_katalog*

rmdir – usuwa pusty katalog

- *rmdir katalog*

ls – listuje katalog

- *ls* – listuje katalog . (*ls .*)
- *ls plik1 plik2 plik3* – listuje tylko wymienione pliki
- *ls *.txt* – wypisze wszystkie pliki o nazwie kończącej się na '.txt'
- *ls katalog1 katalog2* – listuje wymienione katalogi
- *ls -l* – szczegółowa lista
- *ls -a* – wypisuje również ukryte pliki (czyli te których nazwa zaczyna się kropką)
- *ls -R* – listuje katalogi rekursywnie (czyli wyświetla również zawartość podkatalogów)
- *ls -d* – wyświetla tylko nazwy katalogów, tak jak zwyczajnych plików, czyli nie listuje ich zawartości

chmod – zmienia prawa dostępu do pliku

grupy użytkowników: u - user, g - group, o - others, a - all

prawa dostępu: r - read, w - write, x - execute

- *chmod o+r plik* – udziel innym prawo do czytania pliku
- *chmod a-x plik* – zabierz wszystkim prawo do wykonywania pliku
- *chmod g=rw plik* – ustaw prawa do czytania i pisania dla swojej grupy
- *chmod -R go+w katalog* – ustawia prawa wszystkim plikom w katalogu i jego podkatalogach (--recursive)

Przekierowania

> – przekierowanie wyjścia z programu do pliku.

Standardowym wyjściem każdego programu jest ekran (konsola tekstowa) a standardowym wejściem jest klawiatura. Można te wejścia i wyjścia przekierowywać dowolnie.

- *echo "ala ma kota" > plik.txt* – wyjście z programu echo wpisze do pliku plik.txt
- *ls -l > lista.dat* – wypisze liste plików do pliku lista.dat

>> – dołączenie wyjścia z programu na koniec pliku

- *echo "ala ma psa" >> plik.txt* – dopisze "ala ma psa" na koniec pliku plik.txt
- *ls -l > lista.dat* – wypisze liste plików do pliku lista.dat

| – przekierowanie wyjścia jednego programu na wejście drugiego

- *cat plik.txt | wc -l* – cat wypisuje plik.txt na wyjście które przekierowujemy na program liczący wiersze.
- *ls -l | lpr* – program drukujący 'lpr' dostanie na wejście liste plików

- `cat plik.txt | tac | grep "coś" | head > cosie.txt` – wypisanie pliku.txt na program 'tac', który odwaca kolejność wierszy, wynik tego przekierwany na 'grep', który wypisze tylko linie zawierające słowo "coś", wynik tego wysłany na program 'head', który pośle dalej tylko pierwsze 10 wierszy na wyjście, które przekierowaliśmy do pliku cosie.txt.

>! – przekierowanie do pliku. Działa podobnie jak **>**, ale kontynuuje nawet gdy plik już istnieje. Działa w "tcsh".

- `echo "ala ma kota" > plik.txt` – gdy plik.txt istnieje, ta komenda może się nie powieść.
- `echo "ala ma kota" >! plik.txt` – konieczne będzie użycie wykrzyknika **>!**
- `echo "ala ma kota" >| plik.txt` – to samo tylko w 'bash'

10

< – przekierowanie pliku jako standardowego wejścia.

- `szachy < ruchy.txt` – jeśli program szachy przyjmuje ruchy na standardowe wejście, możemy te ruchy spisać do pliku i podać programowi w ten sposób
- `cat ruchy.txt | szachy` – to samo można też zrobić tak

<< – podanie na wejście następujących później linii.

- `szachy << DO_KONCA`
`E2-B4`
`H5-A1`
`C6-F5`
`DO_KONCA` – podanie tzw. dokumentu w miejscu. Wszystkie następujące linie między etykietami 'DO_KONCA' będą podane na standardowe wejście programu 'szachy'.
- `echo "E2-B4`
`H5-A1`
`C6-F5" | szachy` – to samo można też zrobić tak
- `echo -e "E2-B4\nH5-A1\nC6-F5" | szachy` – to samo można też zrobić tak

2> – przekierowanie standardowego wyjścia dla błędów do pliku. Oprócz standardowego wyjścia i wejścia, każdy program ma jeszcze standardowe wyjście dla błędów. Je też możemy przekierowywać, na przykład w inne miejsce niż wyjście zwykłe. Działa w "bash", nie w "tcsh".

- `find -name "plik.*" >znalezione.log 2>bledy.log` – pliki znalezione przez 'find' wylądują w znalezione.log, komunikaty o błędach nie zaciemnią nam wyniku i wpiszą się do innego pliku – bledy.log
- `cp -r dane/ backup/ 2>error.log` – jeśli podczas kopiowania całego katalogu wystąpią błędy, wszystkie komunikaty zostaną wpisane do error.log
- `(ls > plik.log) >& plik.err` – w 'tcsh' nie można przekierować samego wyjścia dla błędów, stąd konieczność takiej konstrukcji.

&> lub >& – przekierowanie obu wyjść do pliku.

- *ls >& plik.log* – standardowe wyjście i standardowe wyjście dla błędów jest przekierowane do plik.log
- *ls >plik.log 2>&1* – to samo, ale działa tylko w 'bash'. Przekierowanie wyjścia, a potem skopiowanie go na wyjście dla błędów.
- *ls &> plik.log* – to samo co >& ale w notacji bardziej właściwej dla 'bash'.

man bash

- *polecam aby uzyskać więcej informacji*

man tcsh

- *polecam aby uzyskać więcej informacji*

Procesy

ps – wypisuje procesy uruchomione na komputerze

- *ps* – wyświetla procesy uruchomione przez użytkownika
- *ps a* – wyświetl również procesy innych użytkowników
- *ps -l, ps -f, ps -F* – więcej informacji o procesach (od: long, full, extra full)
- *ps f* – wyświetla drzewo zależności procesów (od: forest)
- *ps --help* – :P

bg – uruchamia na nowo zatrzymane (Ctrl-Z) zadanie, ale w tle, tak jakby zostało ono uruchomione z &

- *bg* – uruchamia ostatnio zatrzymane zadanie
- *bg NUMER* – uruchamia zadanie o danym numerze na liście zatrzymanych zadań (jobs)

fg – uruchamia na nowo zatrzymane (Ctrl-Z) zadanie, na pierwszym planie

- *fg* – uruchamia ostatnio zatrzymane zadanie
- *fg NUMER* – uruchamia zadanie o danym numerze na liście zatrzymanych zadań (jobs)

jobs – wyświetla listę zatrzymanych zadań

kill – zabija podany proces

PID - jest to numer identyfikacyjny procesu (process id), można go odczytać np. używając polecenia ps

- *kill PID* – wysyła do procesu o numerze PID sygnał do przerwania procesu
- *kill -KILL PID* – zabija proces bez pytania

& – modyfikator który pozwala uruchomić od razu proces w tle.

- *xcalc &* – uruchamia program xcalc w tle, dzięki temu mamy wolną konsolę

12

Edytory

vim – zaawansowany edytor tekstowy w trybie tekstowym

Vi iMproved - nowa wersja znanego edytora Vi. Posiada

- *podświetlanie składni dla wielu języków programowania*
- *możliwość edycji kilku plików na raz*
- *bogatego helpa*
- *bardzo zaawansowane funkcje edycji*
- ...

gvim – vim w trybie graficznym

emacs – zaawansowany edytor tekstowy w trybie graficznym

Emacs podobnie jak Vim jest wszechstronnym edytorem obsługującym wiele języków i posiadającym bogate funkcje.

uemacs – edytor tekstowy w trybie tekstowym

Micro Emacs jest tekstową wersją Emacs

joe – prosty edytor tekstowy

Joe's Own Editor. Nadaje się do pisania prostych dokumentów

mcedit – edytor tekstowy w trybie tekstowym

mcedit jest wbudowanym w Midnight Commandera edytorem.

Posiada m.in. podświetlanie składni.

Sieć

13

pine – program do obsługi poczty

Bardzo dobry, szybki i wygodny program do sprawdzania i wysyłania poczty elektronicznej. Jego największą zaletą jest to, że działa w trybie tekstowym, więc można uruchomić go na zdalnym terminalu.

ssh – program do zdalnego logowania używając szyfrowanego połączenia

Najważniejszy sieciowy program. Umożliwia logowanie się na dowolny komputer na świecie, przy czym połączenie jest bezpieczne dzięki algorytmom szyfrującym opartym na kluczach RSA.

- *ssh antares* – zaloguje mnie na 'antares'a
- *ssh ja@antares* – zaloguje mnie jako użytkownika 'ja' na 'antares'a
- *ssh ja@antares komenda* – zaloguje mnie tylko by wykonać na 'antares'ie komendę

scp – program do kopiowania plików używając szyfrowanego połączenia SSH

scp łączy się z podanym serwerem i kopiuje podane pliki między oboma komputerami

'scp' do połączenia używa programu 'ssh'

- *scp plik ja@antares:~/moje_pliki/* – skopiuje plik do mojego katalogu na antaresie ~/moje_pliki/
- *scp ja@antares:/var/log/plik .* – do bieżącego katalogu skopiuje podany plik z antaresa

rlogin – prosty protokół zdalnego logowania

- *rlogin antares* – zaloguje mnie na 'antares'a

ping – program diagnostyczny sprawdzający czy istnieje połączenie sieciowe z danym komputerem.

- `ping antares.astrouw.edu.pl` – sprawdzamy czy antares odpowiada (i jak szybko)

Procesy w Linuksie³

Jak zapowiadaliśmy, kontynuujemy sagę na temat konsoli i zaawansowanych technik związanych z niskopoziomową obsługą Linuksa. Dziś poznamy tajniki procesów, dowiemy się jak nimi zarządzać, nadawać im priorytet czy... wręcz zabijać(!)

14

Na początek anegdotka:

W pubie siedzi grupka informatyków, obok grupka dresiarzy. Drudzy zachowują się agresywnie i podśmiewają się z postury geeków i ich flanelowych wdzianek. Atmosfera gęstnieje, nie wiadomo czy nie dojdzie do awantury. Wtem... dzwoni telefon.

[odbiera informatyk]

- Co??? W ogóle ci nie odpowiada? W ogóle się ciebie nie słucha? Próbowaleś już „nice”?

Taak? Nic nie dało..? Ok. Dobra — skilluj go.

[odkłada słuchawkę]

W tym momencie najbardziej napakowany z dresów wstaje i z wielkim podziwem odzywa się do informatyka:

- Szacun.

To tym dowcipnym wstępem przejdźmy od razu do rzeczy.

1. Podstawy

Na początek warto poznać definicję procesu, oraz kilka innych terminów z nim związanych. Mówiąc najogólniej proces jest wykonywanym programem, który posiada pewien stan:

- nowy – proces właśnie został utworzony
- gotowy – proces znajduje się w pamięci czeka na wykonanie swojego kodu przez procesor
- działający – proces, który aktualnie jest wykonywany przez procesor
- zablokowany – proces oczekujący na przydzielenie danego zasobu
- zakończony – proces kończący działanie

Proces otrzymuje dostęp do zasobów takich jak np.: procesor, pamięć, pliki, urządzenia wejścia – wyjścia. Każdy posiada swój unikalny identyfikator liczbowy PID (ang. *Process ID*) oraz identyfikator rodzica (PPID – ang. *Parent Process ID*) czyli procesu który go utworzył. Tak więc uruchomione procesy można przedstawić w formie drzewa, w którego korzeniu znajdować się będzie proces z PID równym jeden – `init`, uruchamiany jako pierwszy po załadowaniu jądra. Podobnie jak pliki procesy posiadają swoich właścicieli, zazwyczaj są to użytkownicy którzy uruchomili dany program, wyjątkiem jest tu program z ustawionym

³ <http://jakilinux.org/konsola/procesy-w-linuksie/>

bitem setuid lub też setgid. Prawa dostępu do zasobów systemu (np. plików w katalogu /dev), użytkownika który uruchomił proces, określają prawa dostępu jakie będzie posiadał proces. Do komunikacji z procesami wykorzystywane są tzw. [sygnały](#). Programy uruchomione są z różnym priorytetem, określanym liczbą nice, przyjmującą wartości od -20 (najwyższy) do 19 (najniższy), domyślna wartość to 0.

2. fg, bg, jobs

Terminal umożliwia uruchamianie kilku programów naraz, przełączanie się pomiędzy nimi polega na przesuwaniu programu z „pierwszego planu” w „tło”. Tylko pierwszoplanowy proces może odbierać dane od użytkownika. Przesuwanie procesów z tła na pierwszy plan i w drugą stronę porównać można do aktywnego okna które przykrywa, pozostałe, nieaktywne okna.

Aby uruchomić program w tle wystarczy na końcu polecenia dostawić znak & (ang. *Ampersand*).

```
$ mpg123 -q plik.mp3&
[1] 638
$
```

Uruchamiając mpg123 w tle mogę słuchać muzyki mając cały czas dostęp do powłoki. Powłoka w nawiasie kwadratowym podała mi numer zadania (ang. *job*) - 1, oraz numer PID - 638.

Aby wyświetlić listę uruchomionych w tle zadań, należy użyć polecenia jobs, posiada ono dodatkowy parametr -l podający dodatkowo numer PID danego zadania.

Aby zatrzymać proces pierwszoplanowy należy użyć kombinacji *Ctrl + z* (w tym przypadku do procesu zostanie wysłany sygnał [SIGSTP](#)), aby zatrzymać proces można posłużyć się kombinacją *Ctrl + c*, przesłany zostanie do procesu sygnał [SIGINT](#).

```
$ jobs -l
[1]- 638 Running          mpg123 -q plik.mp3 &
[2]+ 878 Stopped (tty output)  mc
```

W tle uruchomione zostały dwa zadania, mpg123 jest uruchomiony ponieważ do działania nie potrzebuje interakcji z użytkownikiem, mc jest zatrzymany i czeka na dane od użytkownika.

Aby dany proces uczynić pierwszoplanowym, należy posłużyć się poleceniem fg %numer zadania, lub fg %?nazwa polecenia lub fragment nazwy.

```
$ fg %1
```

```
$ fg %mpg
```

W pierwszym przypadku przeniosłem zadanie na pierwszy plan odwołując się do jego numeru zadania, w drugim odwołując się po fragmencie nazwy.

Polecenie `bg %numer zadania`, uruchomi zatrzymane w tle zadanie, w przeciwieństwie do polecenia `fg`, nie przeniesie zadania na pierwszy plan.

```
$ mpg123 -q plik.mp3
Ctrl + z
[3]+ Stopped          mpg123 -q plik.mp3
$ bg %3
[3]+ mpg123 -q plik.mp3 &
```

Na koniec warto zwrócić uwagę na jeden fakt, na ekranie pojawią się dane wysyłane przez program w tle na standardowe wyjście, wyjście błędów, aby tego uniknąć można przekierować strumień do śmietnika `np.polecenie > /dev/null`

3. ps, pstree

Przedstawione wcześniej polecenie `jobs`, wyświetlało procesy uruchomione na danym terminalu. Aby wyświetlić szczegółowy spis procesów uruchomionych w systemie można posłużyć się poleceniem `ps`, uruchomione bez parametrów zadziała tak jak `jobs`. Pełny spis wszystkich parametrów dostępny jest w podręczniku tego polecenia (`man ps`), poniżej kilka najciekawszych opcji:

- `-e` wyświetli każdy proces uruchomiony w systemie
- `-l` szczegółowy opis
- `-f` opis szczegółowy, krótszy od `-l`
- `-H` pokazuje spis procesów w formie drzewa
- `a` wyświetla wszystkie procesy
- `x` wyświetla procesy uruchomione bez terminala
- `u` podaje nazwę użytkownika który uruchomił proces
- `f` wyświetla procesy w formie drzewa

`Ps` przyjmuje parametry w dwóch, nie kompatybilnych ze sobą, formatach: System V (parametry z myślnikiem `-`), oraz BSD. Wyjście polecenia z parametrami `aux` będzie wyglądać:

```
$ ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.1  2952 1852 ?        Ss   10:49   0:01 /sbin/init
adam    5623  0.0  0.6 27892 7100 ?        Ssl  10:51   0:00 /usr/bin/gnome-session
adam    6815  0.2  1.6 39880 16728 ?        S   11:11   0:14 gedit
adam    7105  0.0  0.2  5708 3060 pts/1    Ss   11:16   0:00 bash
```

Poszczególne kolumny oznaczają:

- `USER` – nazwa użytkownika, właściciela procesu
- `PID` – identyfikator procesu

- %CPU – szacowany procent użycia procesora, obliczany poprzez podzielenie czasu użycia procesora przez proces, przez czas uruchomienia
- %MEM – szacowany procent użycia pamięci
- VSZ – ilość użytej pamięci wirtualnej w KB
- RSS – wykorzystana pamięć fizyczna w KB
- TTY – terminal na którym uruchomiono proces
- STAT – stan procesu, D oczekujący na dane z I/O, R działający, S uśpiony, T zatrzymany, X proces martwy(nie powinien być wyświetlany), Z proces zombie
- START – godzina uruchomienia procesu
- TIME – łączny czas zużycia procesora
- COMMAND – polecenie które uruchomiło proces

Polecenie `ps tree`, wyświetla procesy w formie drzewa, posiada parametr `-a` wyświetlający polecenie które uruchomiło proces.

```
$ ps tree -a
init
├─ cupsd
├─ dbus-daemon --system
├─ gedit
├─ gnome-terminal
│   └─ bash
│       ├── man ps
│       │   └─ pager -s
│       └─ bash
│           ├── man ps
│           │   └─ pager -s
│           └─ bash
│               └─ ps tree -a
```

3. top

Polecenie `top` umożliwia podgląd uruchomionych procesów w systemie w czasie rzeczywistym. Parametr `-u` nazwa użytkownika umożliwia podgląd procesów uruchomionych tylko wybranego użytkownika, `-pPID` pozwala na podgląd procesu o określonym PID, przy czym parametr ten może zostać podany do 20 razy.

```
$ top
top - 17:38:09 up 43 min, 2 users, load average: 0.32, 0.17, 0.06
Tasks: 124 total, 1 running, 123 sleeping, 0 stopped, 0 zombie
Cpu(s): 1.5%us, 0.2%sy, 0.0%ni, 98.3%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 1026304k total, 560996k used, 465308k free, 42120k buffers
Swap: 1020116k total, 0k used, 1020116k free, 246652k cached

  PID USER   PR  NI  VIRT  RES  SHR S %CPU  %MEM    TIME+  COMMAND
 7391 adam    15   0 189m  56m  21m S   3  5.6   0:20.42 firefox-bin
 7517 adam    15   0 2368 1164  876 R   1  0.1    0:00.03 top
 4029 root     13  -2 1744  384  280 S   0  0.0    0:00.76 ipw3945d-2
```

Aby zamknąć program należy posłużyć się przyciskiem `q`, niektóre przydatne skróty:

- `F` lub `O` zmienia kolejność sortowania
- spacja odświeża ekran
- `<`, `>` zmienia kolejność sortowania, wg. wyświetlanych kolumn

- r zmienia wartość nice procesu
- W zapisuje bieżące ustawienia top do pliku ~/.toprc
- k wysyła sygnał do procesu
-

4. htop

Htop wyświetla listę procesów w bardziej rozbudowanej graficznie formie, program ten korzysta z biblioteki ncurses, zazwyczaj nie jest domyślnie dostępny.

- kursory – poruszanie się pomiędzy procesami
- F1 – menu pomocy
- F2 – ustawienia
- F3 – wyszukaj proces
- F4 – odwraca sortowanie
- F5 – wyświetla procesy w formie drzewa
- F6 – zmienia kolejność sortowania
- F7 – zmniejsza priorytet wybranego procesu
- F8 – zwiększa priorytet wybranego procesu
- F9 – wysyła sygnał do procesu
- F10 – kończy program

18

5. kill

Wysyła sygnały procesom, składnia to kill sygnał PID, domyślny sygnał to [SIGTERM](#) (wartość liczbowa: 15), „delikatnie” kończący działanie programu pozwalający zamknąć otwarte pliki i „posprzątać” po swoim działaniu. Mocniejszym w działaniu procesem jest [SIGKILL](#) (wartość 9) wymuszający natychmiastowe zakończenie programu, sygnały [SIGSTOP](#) oraz [SIGCONT](#) odpowiednio zatrzymują i wznowiają wykonanie procesu. Procesowi można wysłać sygnał używając jego nazwy, lub korzystając z wartości liczbowej.

```
kill -SIGKILL 7496
kill -9 7496
```

W obu przypadkach zostanie „zabity” proces od PID 7496.

6. killall

W tym poleceniu można przesyłać sygnał wykorzystując nazwę procesu (można użyć wyrażen regularnych) np.:

```
killall mplayer
```

Zabije wszystkie procesy o nazwie mplayer. Ciekawym parametrem jest -l, wyświetlający wszystkie możliwe sygnały. W niektórych systemach np. Solaris polecenie to zabija wszystkie procesy jakie użytkownik jest w stanie zabić, uruchomione przez roota zakończy działanie systemu.

7. nohup

Polecenie to związane jest z sygnałem [SIGHUP](#). Historycznie sygnał ten informował o utracie połączenia z terminalem. Obecnie wykorzystywany jest do informowania, że pseudo-terminal na którym uruchomiono program zakończył działanie, przez co programy uruchomione na tym terminalu też są kończone. Proces uruchomiony z poleceniem nohup będzie ignorował ten sygnał.

```
$ nohup pidgin  
nohup: dołączanie wyników do `nohup.out'
```

Jeżeli wcześniej wyjście programu nie zostało przekierowane do pliku, nohup domyślnie przekieruje je do nohup.out.

8. pidof, pgrep

To dwa małe ale przydatne polecenia, podają one PID polecenia.

```
$ pidof nautilus  
5668  
$ pgrep nautilus  
5668
```

Polecenie pgrep uruchomione z parametrem -l, wyświetla nazwę programu i PID, a z parametrami -u nazwa użytkownika, -G nazwa grupy wyświetla procesy należące do danego użytkownika lub grupy.

```
$ kill `pidof mplayer`
```

W powyższym przykładzie wykorzystałem pidof do zamknięcia mplayera nieznając jego numeru PID.

9. renice

Polecenie to zmienia priorytet procesu, najwyższy priorytet to -20, najniższy to 19. Priorytety zmniejszać może tylko root.

```
$ renice 19 `pidof top`  
12344: old priority 0, new priority 19
```

Parametr -u nazwa użytkownika zmieni priorytet wszystkim procesom których właścicielem jest podany użytkownik, -g nazwa grupy podana grupa.

```
$ renice 5 -u adam  
1000: old priority 2, new priority 5
```

Konsola — podstawy⁴

W tym artykule przedstawimy podstawowe pojęcia związane z pracą w konsoli. Dowiesz się czym jest właściwie konsola, czym jest terminal i co to takiego powłoka.

Czym jest powłoka, czym jest terminal, czym jest konsola?

Bardzo często pojęcia te używane są zamiennie do opisanego trybu tekstowego. W obecnych czasach rzeczywiście różnice pomiędzy nimi się zacierają. Pojęcia te wywodzą się z czasów, kiedy komputery osobiste nie były jeszcze tak popularne a Unix królował na maszynach typu mainframe.

- **Terminal** jest to urządzenie pozwalające człowiekowi na pracę z komputerem. Umożliwia on wprowadzanie instrukcji do wykonania oraz wyprowadzanie wyników pracy komputera. W zamierzonych czasach rolę terminala pełnił dalekopis i drukarka. Z czasem wprowadzono ekran komputerowy i klawiaturę.
- Mianem **konsoli** określano terminal, na którym pracował administrator systemu. W tym miejscu warto nadmienić, że budowa terminali, dostępność do sprzętu oraz jego kosztu miały olbrzymi wpływ na kształtowanie się Uniksa. Pierwsze terminale nie były za szybkie stąd też programiści musieli tworzyć programy, które były zwięzłe i nie produkowały zbyt dużo danych na wyjściu. W ten sposób powstała reguła KISS (Keep it simple stupid!), która oznacza dążenie do prostych rozwiązań. Programy pobierają dane i wprowadzają je w postaci zwykłego tekstu dzięki czemu polecenia można łączyć za pomocą potoków, można przekierować je do i z pliku za pomocą przekierowań z wykorzystaniem potęgi wyrażeń regularnych.
- Ostatnim pojęciem jest pojęcie **powłoki** (ang. *shell*), zwanej też interpreterem. Powłoka stanowi swoisty interfejs pomiędzy użytkownikiem a jądrem systemu operacyjnego tworząc środowisko do uruchamiania i obsługi programów. Wyróżnia się dwa rodzaje powłok: graficzne takie jak np. Explorer znany z Windows lub finder z Mac OS, oraz tekstowe takie jak np. bash, sh, tcsh.

Bash

⁴ <http://jakilinux.org/konsola/konsola-podstawy/#toc1>

Bash jest jedną z wielu powłok dostępnych w systemach z rodziny Unix. Zwana też jest „Bourne again shell” na cześć Stevena Bourna, twórcy klasycznej powłoki sh. Bash jest wstecznie kompatybilny z powłoką sh. Jedną z ważnych cech każdej powłoki jest fakt, że większość poleceń jest tak naprawdę samodzielnymi programami, które znajdują się w drzewie katalogów. Sam bash też jest programem, który w Linuksie znajduje się w lokalizacji /bin/bash (w Solarisie w /usr/bin/bash, we FreeBSD i OpenBSD w /usr/local/bin/bash). Część poleceń wbudowana jest w powłokę np. cd, break, exec.

Powłoki wykorzystują intensywnie strumienie wejścia/wyjścia:

- stdin — standardowy strumień wejścia, dostarczający informacje do komputera, domyślnie instrukcje pobierane są z klawiatury,
- stdout — standardowy strumień wyjścia, na który wyprowadzane są dane wyjściowe komputera, domyślnie jest to ekran monitora,
- stderr — standardowy strumień błędów, wyprowadzane są na niego błędy jakie napotka program w czasie działania, standardowo jest to ekran monitora.

Sesja z powłoką bash na koncie zwykłego użytkownika wygląda tak, jak pokazano poniżej. Jako tzw. znak zachęty używany jest symbol \$. Po znaku zachęty użytkownik może wpisywać polecenia.

```
adz@laptop:~$  
[adz@laptop ~/]$  
$
```

W przypadku pracy na koncie administratora (ang. *root*), jako znak zachęty używany jest symbol #.

```
root@laptop:~#  
[root@laptop /]#  
#
```

Żaden z przedstawionych przykładów nie będzie wymagał uprawnień administratora, chyba że będzie wspomniane to wcześniej.

Zarządzanie plikami⁵

Systemy uniksowe zapewniają szereg poleceń służących do manipulowania plikami oraz katalogami. Ich mocną stroną jest fakt, że w miarę prosty sposób można ich użyć na grupie plików/katalogów, spełniających określone wymagania. Na przykład można usunąć wszystkie pliki, które spełniają określone kryteria lub dokonać masowej zmiany nazw.

1. ls

ls to bardzo często wykorzystywane polecenie. Wyświetla ono na standardowym wyjściu zawartość katalogu. Jeżeli jako parametr nie poda się ścieżki do katalogu, wyświetlona zostanie zawartość katalogu bieżącego (tj. tego, w którym aktualnie się znajdujemy).

```
adam@laptop:~/Dokumenty/jakilinux.org$ pwd
/home/adam/Dokumenty/jakilinux.org/
adam@laptop:~/Dokumenty/jakilinux.org$ ls
przykład.txt  wszystko_o_konsoli.txt
adam@laptop:~/Dokumenty/jakilinux.org$ ls /var/
backups cache crash games lib local lock log mail opt run spool
tmp
```

W drugim przykładzie powyżej została wyświetlona zawartość katalogu podanego jako parametr. Polecenie ls może wyświetlać bardziej szczegółowe dane, jeżeli uruchomi się je z parametrem -l.

```
adam@laptop:~/Dokumenty/jakilinux.org/przykłady$ ls -l
razem 0
-rw-r--r-- 1 adam adam 0 2007-05-30 11:31 przykład.txt
-rw-r--r-- 1 adam adam 0 2007-05-30 11:31 wszystko_o_konsoli.txt
```

W powyższym listingu w pierwszej linii wyświetlana jest suma bloków na dysku użytych przez pliki w katalogu. Dalej wyświetlane są:

⁵ <http://jakilinux.org/konsola/zarzadzanie-plikami/>

- `-rw-r--` uprawnienia danego pliku, katalogu (szerzej na temat uprawnień w dalszej części przewodnika),
- liczba twardych dowiązań do danego pliku,
- właściciel pliku, grupa do której należy właściciel,
- długość pliku,
- data ostatniej modyfikacji,
- nazwa pliku/katalogu.

Polecenie `ls` umożliwia wyświetlanie plików ukrytych (tzw. pliki z kropką). Pliki ukryte zaczynają się od znaku kropki `.`, do wyświetlania plików ukrytych służy parametr `-a`.

```
adam@laptop:~/Dokumenty/jakilinux.org/przykłady$ ls -a
.  ..  .plik_ukryty  przykład.txt  wszystko_o_konsoli.txt
```

Polecenie `ls` umożliwia sortowanie wyników. Do dyspozycji są parametry:

- `-t` — wyświetla zawartość posortowaną wg daty modyfikacji (od najnowszej do najstarszej),
- `-S` — wyświetla zawartość wg rozmiaru od największego do najmniejszego,
- `-r` — odwraca porządek sortowania.

Istnieje również możliwość wyświetlenia rekursywnego zawartości katalogów, tzn. wyświetlona zostanie zawartość wszystkich podkatalogów w katalogu, służy do tego parametr `-R`.

```
adam@laptop:~/Dokumenty/jakilinux.org/przykłady$ ls -R
.:
katalog1  katalog2  przykład.txt  wszystko_o_konsoli.txt
```

```
./katalog1:
plik1  plik2
```

```
./katalog2:
plik3  plik4
```

2. `cd`

Polecenie `cd` (ang. *change directory*) służy do poruszania się po drzewie katalogów. Jako parametr polecenie pobiera miejsce w drzewie katalogów, do którego chcemy się przenieść.

```
adam@laptop:~$ cd /usr/bin
adam@laptop:/usr/bin$ pwd
/usr/bin
```

W tym miejscu warto nadmienić, że znak `~` (tylda) jest traktowany przez powłokę jako znak specjalny — jest to skrót do katalogu domowego (ang. *home directory*) użytkownika.

```
adam@laptop:/usr/bin$ cd ~
adam@laptop:~$ pwd
/home/adam
```

Jeżeli wpisujemy `cd ~nazwa_użytkownika`, to polecenie to przeniesie nas do katalogu domowego użytkownika o loginie `nazwa_użytkownika`.

```
adam@laptop:~$ cd ~zoidberg
adam@laptop:/home/zoidberg$ pwd
/home/zoidberg
```

Znak `.` - oznacza katalog, który był poprzednio katalogiem roboczym. Aby przejść do katalogu nadrzędnego, należy wpisać `cd ..`. Każdy (nawet pusty) katalog w systemach uniksowych zawiera 2 elementy `.` i `..`. Pojedyncza kropka `.` jest to dowiązanie katalogu do niego samego, natomiast dwukropka `..` jest dowiązaniem do katalogu nadrzędnego w drzewie katalogów.

```
adam@laptop:~$ cd ..
adam@laptop:/home$ pwd
/home
```

Wraz z poleceniem `cd`, warto wprowadzić terminy *ścieżka względna* i *ścieżka bezwzględna*.

- Ścieżka bezwzględna (zwana też absolutną, ang. *absolute path*) to ścieżka, która rozpoczyna się od korzenia drzewa katalogów / np. `/home/adam`.
- Ścieżka względna (ang. *relative path*) to ścieżka rozpoczynająca się od katalogu, w którym obecnie się znajdujemy.

3. mkdir, rmdir

Polecenia te służą odpowiednio do tworzenia i usuwania katalogów. Jako parametr podajemy nazwę tworzonego/usuwanego katalogu. Poleceniem `{rmdir}` można usunąć tylko pusty katalog.

```
adam@laptop:~/Dokumenty/jakilinux.org/przykłady$ mkdir katalog
adam@laptop:~/Dokumenty/jakilinux.org/przykłady$ ls
katalog  przykład.txt  wszystko_o_konsoli.txt
adam@laptop:~/Dokumenty/jakilinux.org/przykłady$ rmdir katalog/
adam@laptop:~/Dokumenty/jakilinux.org/przykłady$ ls
przykład.txt  wszystko_o_konsoli.txt
```

cp, mv, rm

- `{cp}` — polecenie to służy do kopiowania plików, pobiera ono co najmniej dwa parametry: *plik źródłowy* i *miejsce docelowe*, do którego chcemy go skopiować,
- `mv` — służy do przenoszenia lub zmiany nazwy plików lub katalogów, działa na identycznej zasadzie jak polecenie `cp`,
- `rm` — służy do usuwania plików.

Polecenia `cp`, `mv` i `rm` mają następujące wspólne parametry:

- `-f force` — wymusza usunięcie docelowego pliku, nawet w przypadku braku uprawnień do zapisu,
- `-i interactive` — użytkownik zostanie poproszony o potwierdzenie wykonywanej operacji,

- `-b` —backup — zostanie utworzona kopia zapasowa plików nadpisywanych (tylko `cp` i `mv`),

Polecenia `cp` i `rm` mogą również pracować w trybie rekursywnym, umożliwia to parametr `-r` (`-R`, `—recursive`).

```
adam@laptop:~/Dokumenty/jakilinux.org/przyklady$ ls
katalog1 katalog2 katalog3 przyklad.txt wszystko_o_konsoli.txt
adam@laptop:~/Dokumenty/jakilinux.org/przyklady$ cp -R katalog2/ katalog3/
adam@laptop:~/Dokumenty/jakilinux.org/przyklady$ ls katalog3/
katalog2
adam@laptop:~/Dokumenty/jakilinux.org/przyklady$ rm -r katalog3/
adam@laptop:~/Dokumenty/jakilinux.org/przyklady$ ls
katalog1 katalog2 przyklad.txt wszystko_o_konsoli.txt
```

Poleceniem `rm -r` można usuwać niepuste katalogi, tak jak na przykładzie powyżej.

5. ln

Jest to polecenie wykorzystywane do tworzenia dowiązań do plików, dowiązania można interpretować jako odpowiednik skrótu do pliku znanego z systemów Windows (jest to trochę naciągana interpretacja!)

Wyróżniamy dwa rodzaje dowiązań:

- miękkie (tzw. symboliczne), odwołują się one do pliku — ten rodzaj dowiązań można interpretować jako odpowiednik skrótu z Windows,
- twarde — jest to dowiązanie do konkretnego miejsca na dysku, innymi słowy do fizycznego obszaru na dysku gdzie znajduje się plik.

Jeżeli usunie się plik, do którego prowadzi dowiązanie symboliczne, dowiązanie to zostanie podświetlone na czerwono (jeżeli powłoka wspiera kolory), jeżeli usuniemy plik, do którego utworzone zostało dowiązanie twarde, nic się nie stanie, taki plik zostanie usunięty dopiero wtedy, gdy liczba twardych dowiązań spadnie do zera.

Dowiązania symboliczne tworzy się uruchamiając polecenie `ln` z parametrem `-s`.

```
adam@laptop:~/Dokumenty/jakilinux.org/przyklady$ cat plik
Przykład dowiązań.
adam@laptop:~/Dokumenty/jakilinux.org/przyklady$ ln -s plik \
dowiazanie_symboliczne
adam@laptop:~/Dokumenty/jakilinux.org/przyklady$ ls -l
lrwxrwxrwx 1 adam adam  4 2007-06-01 19:11 dowiazanie_symboliczne -> plik
-rw-r--r-- 1 adam adam 22 2007-06-01 19:10 plik
adam@laptop:~/Dokumenty/jakilinux.org/przyklady$ rm plik
adam@laptop:~/Dokumenty/jakilinux.org/przyklady$ ls
dowiazanie_symboliczne
```

W powyższym przykładzie można zaobserwować działanie dowiązań symbolicznych. Polecenie `ls` wskazuje, że dla pliku nie zmieniła się liczba twardych dowiązań. Zupełnie inaczej

jest na przykładzie poniżej, gdzie liczba twardych dowiązań wynosi 2, po usunięciu pliku spada do 1, plik cały czas jest dostępny.

```
adam@laptop:~/Dokumenty/jakilinux.org/przykłady$ ln plik dowiazanie_twarde
adam@laptop:~/Dokumenty/jakilinux.org/przykłady$ ls -l
-rw-r--r-- 2 adam adam 22 2007-06-01 19:11 dowiazanie_twarde
-rw-r--r-- 2 adam adam 22 2007-06-01 19:11 plik
adam@laptop:~/Dokumenty/jakilinux.org/przykłady$ rm plik
adam@laptop:~/Dokumenty/jakilinux.org/przykłady$ ls -l
-rw-r--r-- 1 adam adam 22 2007-06-01 19:11 dowiazanie_twarde
```

6. touch

touch to proste polecenie mające dwa zastosowania. Jeżeli podamy jako parametr istniejący plik, to zmieni mu datę modyfikacji. Jeżeli podamy nazwę nieistniejącego pliku, to zostanie utworzony pusty plik.

```
adam@laptop:~/Dokumenty/jakilinux.org/przykłady$ ls -l przykład.txt
-rw-r--r-- 1 adam adam 0 2007-05-30 11:31 przykład.txt
adam@laptop:~/Dokumenty/jakilinux.org/przykłady$ touch przykład.txt
adam@laptop:~/Dokumenty/jakilinux.org/przykłady$ ls -l
razem 0
-rw-r--r-- 1 adam adam 0 2007-06-07 13:27 przykład.txt
-rw-r--r-- 1 adam adam 0 2007-05-30 11:31 wszystko_o_konsoli.txt
adam@laptop:~/Dokumenty/jakilinux.org/przykłady$ touch nowy.txt
adam@laptop:~/Dokumenty/jakilinux.org/przykłady$ ls -l
razem 0
-rw-r--r-- 1 adam adam 0 2007-06-07 13:28 nowy.txt
-rw-r--r-- 1 adam adam 0 2007-06-07 13:27 przykład.txt
-rw-r--r-- 1 adam adam 0 2007-05-30 11:31 wszystko_o_konsoli.txt
```

Uruchamiając touch z parametrem -c lub --no-create, zapobiegniemy utworzeniu nowego pliku. Parametr -d lub -t przy czym parametr t wymaga określenia daty w formacie MMDDhhmm.

```
adam@laptop:~/Dokumenty/jakilinux.org/przykłady$ ls -l
razem 0
-rw-r--r-- 1 adam adam 0 2007-06-07 13:28 nowy.txt
-rw-r--r-- 1 adam adam 0 2007-06-07 13:27 przykład.txt
-rw-r--r-- 1 adam adam 0 2007-05-30 11:31 wszystko_o_konsoli.txt
adam@laptop:~/Dokumenty/jakilinux.org/przykłady$ ls -l
razem 0
-rw-r--r-- 1 adam adam 0 2007-06-07 13:28 nowy.txt
-rw-r--r-- 1 adam adam 0 2007-06-07 13:27 przykład.txt
-rw-r--r-- 1 adam adam 0 2007-05-30 11:31 wszystko_o_konsoli.txt
adam@laptop:~/Dokumenty/jakilinux.org/przykłady$ touch -t \\
200706101200 nowy.txt
adam@laptop:~/Dokumenty/jakilinux.org/przykłady$ touch -d \\
"last monday" przykład.txt
adam@laptop:~/Dokumenty/jakilinux.org/przykłady$ touch -d \\
"2 days ago 12:00" wszystko_o_konsoli.txt
adam@laptop:~/Dokumenty/jakilinux.org/przykłady$ ls -l
razem 0
```

```
-rw-r--r-- 1 adam adam 0 2007-06-10 12:00 nowy.txt
-rw-r--r-- 1 adam adam 0 2007-06-04 00:00 przykład.txt
-rw-r--r-- 1 adam adam 0 2007-06-05 12:00 wszystko_o_konsoli.txt
```

7 df, du

Służą do wyświetlania miejsca zajmowanego przez system plików (df) oraz przez konkretny plik (du). Uruchomione z parametrem -h dadzą wyniki w megabajtach. Przykład dla polecenia df:

```
adam@laptop:~/Dokumenty/jakilinux.org/przykłady$ df -h
System plików      rozm. użyte dost. %uż. zamont. na
/dev/sda5          40G   34G   4,0G   90% /
varrun             502M   136K   502M    1% /var/run
varlock            502M    0   502M    0% /var/lock
procbususb         502M   148K   502M    1% /proc/bus/usb
udev               502M   148K   502M    1% /dev
devshm             502M    0   502M    0% /dev/shm
```

Przykład dla polecenia du:

```
adam@laptop:~/Dokumenty/jakilinux.org/przykłady$ du -h error.txt
4,0K   error.txt
```

8. echo

Jest to proste polecenie, które jako wynik swojej pracy zwraca tekst, który został podany mu jako parametr.

```
adam@laptop:~$ echo Hello World!
Hello World!
```

Polecenie to dodaje na końcu znak przejścia do nowej linii, aby to ominąć, należy uruchomić program, z parametrem -n. Na przykład:

```
adam@laptop:~$ echo -n Hello World!
Hello World!adam@laptop:~$
```

W przypadku zamknięcia tekstu w cudzysłowy, tekst będzie interpretowany bezpośrednio (o tym później).

9. pwd

pwd (ang. *print working directory*) to polecenie wyświetlające pełną ścieżkę w miejscu w drzewie katalogów, w którym aktualnie się znajdujemy.

```
adam@laptop:~$ pwd
/home/adam
```

10. cat

Polecenie `cat` można wykorzystywać w celu tworzenia pliku a dokładniej rzecz ujmując przekierowania standardowego wejścia do pliku oraz wyświetlenia na standardowym wyjściu. Plik tworzy się w następujący sposób:

```
adam@laptop:~$cat > plik.txt
Bardzo ciekawy tekst.
<Ctrl+d>
```

Temat użycia operatora `>` zostanie poruszony szerzej w dalszej części przewodnika. Aby wyświetlić zawartość pliku, wystarczy wpisać `cat nazwa_pliku`.

```
adam@laptop:~$cat plik.txt
Bardzo ciekawy tekst.
```

Możesz również uzyskać numery linii wyświetlanego pliku w lewym rogu ekranu.

```
adam@laptop:~$cat -n plik.txt
1  Bardzo ciekawy tekst.
```

Polecenie `cat` może również łączyć pliki, w poniższym przykładzie scalam 5 fragmentów obrazu ISO w jedną całość.

```
cat plik1 plik2 plik3 plik4 plik5 > file.iso
```

11. `wc`, `head`, `tail`

Te trzy polecenia wykorzystywane są do operacji na ciągach tekstu. Polecenie `wc` wyświetla liczbę linii, słów oraz bitów w danym pliku.

```
adam@laptop:~/Dokumenty/jakilinux.org$ wc wszystko_o_konsoli.txt
94  908 6828 wszystko_o_konsoli.txt
```

W powyższym przykładzie pierwsza kolumna to liczba linii, druga słów, a trzecia liczba bitów. Kolejne dwa polecenia wyświetlają odpowiednio początek pliku (`head`) oraz koniec (`tail`) — domyślnie wyświetlane jest 10 linii tekstu. Wartość tę w obu programach można regulować parametrem `-n` podając liczbę linii, które chcemy zobaczyć.

```
adam@laptop:~/Dokumenty/jakilinux.org$ head -n 1 przykład.txt
Pierwsza linia tekstu!
adam@laptop:~/Dokumenty/jakilinux.org$ tail -n 2 przykład.txt
Przed ostatnia linia tekstu!
Ostatnia linia tekstu!
```

12 `less`

Polecenie `less` służy do przeglądania pliku z możliwością jego przewijania, zarówno do przodu, jak i wstecz.

```
adam@laptop:~$less plik.txt
```

Powyższe polecenie spowoduje wyświetlenie pliku plik.txt. Wprowadzenie `:f` podczas korzystania z programu spowoduje wyświetlenie interesujących informacji, np. numer linii na górze ekranu czy wielkości pliku. Dodatkowo zastosowanie polecenia `cat` z przełącznikiem `-n` umożliwia wyświetlenie numeru każdej linii przy lewej krawędzi ekranu, co widać poniżej.

```
adam@laptop:~$cat plik.txt | less
```

Strumienie, potoki i przekierowania⁶

1. Potoki

Siłą konsoli systemów uniksowych jest możliwość łączenia małych zwięzłych poleceń w większe, bardziej skomplikowane polecenia, za pomocą potoków (ang. *pipe*). Dane na wyjściu jednego programu będą stanowiły dane wejściowe drugiego programu. To właśnie dlatego w systemach uniksowych tak powszechnie wykorzystywane są tekstowe strumienie danych. Te założenia legły u podstaw filozofii Uniksa.

Polecenia łączy się za pomocą operatora `|` zwanego czasem w polskojęzycznej literaturze „rurą” (ang. *pipe*). W skrócie wygląda to następująco:

```
polecenie1 | polecenie2 | polecenie3
```

W przykładzie poniżej wyjście polecenia `ls` przekierowuję do polecenia `wc` z parametrem `-l`, dzięki czemu zostanie wyświetlona liczba plików i katalogów w danym katalogu.

```
adam@laptop:~/Dokumenty/jakilinux.org/przyklady$ ls  
nowy.txt  przykład.txt  wszystko_o_konsoli.txt  
adam@laptop:~/Dokumenty/jakilinux.org/przyklady$ ls | wc -l  
3
```

Potoki wykorzystywane są intensywnie w przypadkach takich jak przeszukiwanie logów systemowych z wykorzystaniem wyrażeń regularnych lub w wyszukiwaniu plików (o tym w dalszej części poradnika).

2. Strumienie i przekierowania

Tak jak wspominałem już na początku poradnika, powłoki w systemach uniksowych intensywnie wykorzystują strumienie: wejścia, wyjścia i błędów. Strumienie można przekierowywać z urządzenia, które standardowo je obsługuje na inne lub też do pliku, np. można przekierować komunikaty o błędach z monitora do pliku.

Każdy ze strumieni ma swój unikalny deskryptor oznaczany liczbowo:

⁶ <http://jakilinux.org/konsola/strumienie-potoki-i-przekierowania/>

- stdout — standardowe wyjście (deskryptor 1) — przeważnie jest to monitor komputera,
- stderr — standardowe wyjście błędów (deskryptor 2) — przeważnie jest to monitor komputera,
- stdin — standardowe wejście (deskryptor 0) — przeważnie jest to klawiatura komputera.

Do przekierowania używa się operatora deskryptor>, gdzie za deskryptor podstawia się jedną z wartości liczbowych przedstawionych powyżej. Jeżeli plik, do którego przekierowujemy, nie istnieje, to zostanie utworzony (oczywiście o ile mamy uprawnienia do zapisu w danym katalogu). Jeżeli plik istnieje (i mamy uprawnienia do zapisu w nim), jego zawartość zostanie zastąpiona. Aby tego uniknąć, należy użyć operatora deskryptor», który dopisze zawartość strumienia na koniec pliku.

W poniższym przykładzie przekieruję standardowy strumień błędów do pliku error.txt, następnie przekieruję standardowe wejście z klawiatury do pliku i na koniec przekieruję na standardowe wyjście polecenie z pliku polecenie.txt

```
adam@laptop:~/Dokumenty/jakilinux.org/przyklady$ cat \\  
nieistniejacy_plik.txt 2> error.txt  
adam@laptop:~/Dokumenty/jakilinux.org/przyklady$ cat error.txt  
cat: nieistniejacy_plik.txt: No such file or directory  
adam@laptop:~/Dokumenty/jakilinux.org/przyklady$ cat > polecenie.txt  
ls -l  
(Ctrl+D)  
adam@laptop:~/Dokumenty/jakilinux.org/przyklady$ bash < polecenie.txt  
razem 8  
-rw-r--r-- 1 adam adam 55 2007-06-26 14:04 error.txt  
-rw-r--r-- 1 adam adam 0 2007-06-10 12:00 nowy.txt  
-rw-r--r-- 1 adam adam 5 2007-06-26 14:07 polecenie.txt  
-rw-r--r-- 1 adam adam 0 2007-06-04 00:00 przyklad.txt  
-rw-r--r-- 1 adam adam 0 2007-06-05 12:00 wszystko_o_konsoli.txt  
adam@laptop:~/Dokumenty/jakilinux.org/przyklady$
```

Przekierowania dość powszechnie wykorzystywane są w przypadku zadań uruchamianych w tle, kiedy nie ma potrzeby przeglądania komunikatów programu na bieżąco. Warto również wspomnieć o tym, jak całkowicie pozbyć się komunikatów.

```
adam@laptop:~/Dokumenty/jakilinux.org/przyklady$ cat \\  
nieistniejacy_plik.txt 2> /dev/null
```

Ponieważ wszystkie urządzenia w Uniksie są reprezentowane jako pliki, istnieje możliwość takiego przekierowania jak powyżej, innymi słowy wszystkie komunikaty błędów trafiły do śmietnika czyli urządzenia /dev/null.

3. xargs

Poleceniem, którego działanie wiąże się z ideą strumieni i potoków, jest polecenie xargs. Otrzymuje ono na wejściu strumień tekstu i rozdziela go wg kryteriów (znak null lub znak końca wiersza) i następnie tak rozdzielone fragmenty przekazuje po kolei jako argumenty do

kolejnego polecenia. `xargs` jest bardzo często wykorzystywany w połączeniu z poleceniami `find`, `locate` i `grep`.

```
adam@laptop:~$ ls | grep raport | xargs -i cp {} Dokumenty/
```

W powyższym przykładzie zastosowano polecenie `grep`, które szerzej zostanie omówione później. Wszystkie pliki, zawierające w nazwie ciąg znaków `raport`, zostaną przekopiowane do katalogu `Dokumenty`, dzięki zastosowaniu opcji `-i`, nawiasy `{}` zastępowane są fragmentem strumienia, w tym przypadku nazwą pliku. Kolejnym często używanym parametrem tego programu jest parametr `-0` (lub równoważny `-Z`). Dzięki niemu jako znak rozgraniczający fragmenty strumienia będzie wykorzystywany znak null. Opcja ta przydatna jest w przypadku nazw plików zawierających spację. **Uwaga!** Opcja `-0` (null) dostępna jest tylko w wersji [GNU](#) polecenia `xargs`.

Wyszukiwanie, wyrażenia regularne⁷

Systemy uniksowe oferują kilka przydatnych poleceń służących do wyszukiwania plików, ciągów tekstu, które w połączeniu z przedstawionymi powyżej strumieniami i przekierowaniami oraz z wyrażeniami regularnymi stanowią bardzo potężne narzędzie administratorskie.

1. Wyrażenia regularne

Wyrażenia regularne są wzorcami opisującymi, zastępującymi łańcuchy tekstów, zostały one wprowadzone od samego początku istnienia Uniksa przez jednego z jego twórców — Kena Thompsona. Wyrażenia regularne są bardzo efektywnym sposobem pracy z tekstem i zdecydowanie warto je opanować.

Symbol Zastępuje

- `.` dowolny znak
- `^` dopasuj występujące po operatorze wyrażenie do początku wiersza

⁷ <http://jakilinux.org/konsola/wyszukiwanie-wyrazenia-regularne/>

\$	dopasuj poprzedzające wyrażenie do końca wiersza
\x	znaki specjalne, gdzie x to znak specjalny np. \\$ zastąpi znak dolara
[lista]	zastępuje dowolny znak spośród tych wymienionych na liście, mogą to być przedziały np. [0-9] lub [a-d]
()	grupowanie wyrażeń regularnych
?	dokładnie jeden element wcześniejszy
a b	dopasuje wyrażenie a lub wyrażenie b
*	dopasuj zero lub więcej wyrażeń znaku poprzedzający operator
+	jeden lub więcej elementów poprzedzających operator

2. Wyrażenia regularne a znaki globalne

Warto nadmienić że, bash do wersji 3.0 nie miał wbudowanej obsługi wyrażeń regularnych, które to były wykorzystywane przez programy pracujące na strumieniach tekstu np.: [sed](#), [awk](#) czy też [grep](#). Za to wbudowana była obsługa wyrażeń globalnych i znaków wieloznacznych (ang. *wildcards*).

Symbol Zastępuje

*	dowolny ciąg znaków
?	dokładnie jeden znak
[lista]	zastępuje dowolny znak spośród tych wymienionych na liście, mogą to być przedziały np. [0-9] lub [a-d]
[^lista]	wybrane zostaną znaki, które <i>nie są</i> na liście
{}	grupuje wyrażenie globalne

Wyrażenia regularne pozwalają wyszukać dany łańcuch w strumieniu tekstu, podczas gdy wyrażenia globalne zastępują fragmenty tekstu. W poniższym, bardzo prostym przykładzie usunięto wszystkie pliki rozpoczynające się na literę „a”.

```
adam@laptop:~/Dokumenty/jakilinux.org/przyklady$ ls
aa abc    nowy.txt  przykład.txt
ab error.txt polecenie.txt wszystko_o_konsoli.txt
adam@laptop:~/Dokumenty/jakilinux.org/przyklady$ rm a*
adam@laptop:~/Dokumenty/jakilinux.org/przyklady$ ls
error.txt nowy.txt polecenie.txt przykład.txt wszystko_o_konsoli.txt
```


W tym natomiast usunięto wszystkie które *nie mają* jako pierwszej litery z zakresu b do z i dalsza ich nazwa to dowolny ciąg znaków.

```
adam@laptop:~/Dokumenty/jakilinux.org/przykłady$ ls
aa abc      nowy.txt   przykład.txt
ab error.txt polecenie.txt wszystko_o_konsoli.txt
adam@laptop:~/Dokumenty/jakilinux.org/przykłady$ rm [^b-z]*
adam@laptop:~/Dokumenty/jakilinux.org/przykłady$ ls
error.txt nowy.txt polecenie.txt przykład.txt wszystko_o_konsoli.txt
```

3. grep

grep to powszechnie wykorzystywany program do wyszukiwania w strumieniu wejścia ciągów tekstowych, pasujących do podanego wyrażenia regularnego. Występuje on w każdym systemie uniksowym a jego autorem jest Ken Thompson.

grep ma kilka przydatnych parametrów:

- -c — wyświetla tylko liczbę znalezionych linii,
- -n — wyświetlany jest numer linii w pliku, w którym znaleziono dany ciąg znaków,
- -w — wyszukuje tylko całe słowa,
- -x — wyszukuje tylko całe linie.

```
adam@laptop:~/Dokumenty/jakilinux.org/przykłady$ dmesg | grep Mouse
[ 15.436000] input: USB-PS/2 Optical Mouse as /class/input/input2
[ 15.436000] input: USB HID v1.10 Mouse [USB-PS/2 Optical Mouse]
on usb-0000:00:1d.0-2
```

W powyższym przykładzie wyjście polecenia dmesg (wyświetlającego informacje dziennika zdarzeń jądra systemu) przekierowałem za pomocą potoku (operator |) do polecenia grep, gdzie jako wyrażenia regularnego użyłem słowa „Mouse” (wielkość liter ma znaczenie). Polecenie przefiltrowało wejście i wyświetliło jedynie te linie, które zawierają słowo „Mouse”.

Do kolejnych przykładów założmy, że mamy plik tekstowy, *wyrażenia.txt* o zawartości:

```
1. owoc
2. rower
3. dom
4. auto
5. płyta
ananas
```

Tak więc:

```
grep ^[1-6]..a wyrażenia.txt
4. auto
ananas
```

Powyższe wyrażenie wyszuka wszystkie łańcuchy, które zaczynają się od liczb z przedziału od 1 do 6, dalej zawierają dwa dowolne znaki (w naszym przypadku kropkę i spację), dalej

zawierają literę „a”, za nią dowolny już ciąg znaków.
Wyrażenie poniżej wyświetli każdą linię kończącą się na literę „a”.

```
grep a$ wyrażenia.txt  
5. płyta  
grep a.*a wyrażenia.txt  
ananas
```

Powyższe wyrażenie wyszuka dowolny ciąg zaczynający się na a, dalej zawierający dowolny ciąg znaków, i kończący się na a. W tym przypadku warto zwrócić uwagę na zapis „.*”. Jak pamiętamy, znak * w wyrażeniach regularnych zwróci zero lub więcej znaków poprzedzających ten operator, natomiast operator . oznacza dowolny znak. Ujmując to prościej, wyszukujemy w ten sposób zero lub więcej wystąpień dowolnego znaku.

```
grep "(1|4)" wyrażenia.txt  
1. owoc  
4. auto
```

Przedstawione powyżej wyrażenie, jest i tyle ciekawe, że użyto w nim znaków, które bash traktuje jako znaki specjalne. Aby to ominąć, postawiłem przed nimi znak \, a całość zamknąłem w cudzysłów, aby uniknąć interpretowania przez bash tego wyrażenia (zrobi to grep).

Tematyka wyrażeń regularnych jest bardzo rozbudowana, dlatego zachęcam do samodzielnego eksperymentowania i ćwiczeń.

4. find

Polecenie find przeszukuje drzewo katalogów w poszukiwaniu plików lub katalogów o podanej nazwie lub jej części, lub o podanych kryteriach takich jak: rozmiar, typ, właściciel plików, data utworzenia lub data ostatniej modyfikacji. Najprostsze wywołanie programu find może wyglądać następująco:

```
adam@laptop:~$ find . -name linux  
./Downloads/Firefox/vmware-server-distrib/lib/perl5/\  
site_perl/5.005/i386-linux/linux  
./Downloads/Firefox/tp_smapi-0.31/include/linux
```

Składnia tego polecenia jest następująca: find katalogi_startowe kryterium wyszukiwania i operacje, które należy wykonać na wyszukanych elementach. W powyższym przykładzie katalogiem startowym jest katalog bieżący, jak już wcześniej wspominałem każdy katalog zawiera dowiązanie do siebie samego reprezentowane przez kropkę, oraz kryterium wyszukiwania — wszystkie pliki i katalogi zawierające frazę „linux”. Wyszukiwanie rozpoczyna się od katalogu startowego i postępuje w dół drzewa katalogów, tzn. najpierw katalog bieżący, a potem jego podkatalogi, oczywiście o ile ma się prawo do ich odczytu. W programie find można (i warto a wręcz należy) używać znaków globalnych przedstawionych

wcześniej.

find umożliwia wyszukiwanie według typu.

```
adam@laptop:~$ find . -type d
```

Powyższy przykład wyszuka wszystkie katalogi znajdujące się w bieżącej lokalizacji. Kryteria oczywiście można łączyć.

```
adam@laptop:~$ find . -type d -name Dokumenty
./Dokumenty
./Dokumenty/jakilinux.org/Dokumenty
```

Powyższe wywołanie znajdzie wszystkie katalogi w bieżącej lokalizacji zawierające w swej nazwie zwrot „Dokumenty”. Możliwe typy plików przedstawia poniższa tabela.

Parametr Rodzaj pliku

b	urządzenie blokowe
c	urządzenie znakowe
d	katalog
f	zwykły plik
l	dowiązanie symboliczne
s	gniazdo (ang. <i>socket</i>)

find wywołany z parametrem -size wartość, wyszuka pliki o wielkości podanej w parametrze wartość, jeżeli do wartości dodamy znak + (np. -size +wartość), program wyszuka pliki *większe* od podanej wartości. Jeżeli dodamy znak - (np. -size -wartość), wyszukane zostaną pliki *mniejsze* od podanej wartości. Domyślna wartość to 512 bitowe bloki, pozostałe wartości to:

- c — bajty,
- k — kilobajty,
- M — megabajty,
- G — gigabajty.

W poniższym przykładzie zostaną wyszukane wszystkie pliki o rozmiarze większym niż 100 MB ale mniejszym niż 200 MB.

```
adam@laptop:~$ find . -size +100M -size -200M
./plan9_07010zip
```

Pozostałe kryteria wyszukiwania przedstawia poniższa tabela.

-atime n Ostatni dostęp miał miejsce n dni temu

-mtime n	Plik został zmodyfikowany n dni temu
-newer plik	Wyszukiwany plik został zmodyfikowany wcześniej niż podany plik
-links n	Plik zawiera dokładnie n twardych dowiązań
-perm p	Plik ma uprawnienia, gdzie p to liczbowy tryb dostępu
-user użytkownik	Właścicielem pliku jest użytkownik
-group grupa	Właścicielem pliku jest grupa
-empty	Puste pliki

Opcje liczbowe można poprzedzać znakami + i -, które oznaczają odpowiednio „więcej niż” oraz „mniej niż”, podobnie jak miało to miejsce w kryterium size opisanym wcześniej.

Jak już wcześniej wspominałem, polecenie `find` może wykonać określone operacje na plikach, które znajdzie. Domyślną operacją jest `-print`, która wypisuje nazwy łącznie z adresami plików. W niektórych powłokach należy dodawać tę opcję za każdym razem.

Kolejna możliwa akcja to `-ls`, która wypisuje informacje o plikach w ten sam sposób, co polecenie `ls` uruchomione z parametrami `-lids`. Ostatnia możliwość to uruchomienie z parametrem `-exec` i wykonanie dowolnego polecenia na znalezionych plikach.

```
adam@laptop:~$ find . -size +100M -size -150M -ls
2485508 115744 -rw-r--r-- 1 adam adam 118398976 maj 2 22:44
./plan9/plan9_compressed.img
find . -size +110M -size -150M -exec cp {} /home/adam/pliki/ \;
```

Pierwszy z powyższych przykładów, jak widać, wypisze szczegółowe dane wyszukiwanych plików. Drugi jest ciekawszy i wymaga dokładniejszej analizy. `find` wykona na plikach operację podaną wraz z parametrem. W tym przypadku, wszystkie pliki większe niż 110 MB i mniejsze niż 150 MB zostaną skopiowane do katalogu pliki, podwójny nawias klamrowy `{}` oznacza, że operacja ma być wykonana na każdym pliku a `\` przed `;` chroni przed błędną interpretacją polecenia przez powłokę.

`find` ma jeszcze inne ciekawe opcje: `-ok` — działa podobnie jak `-exec` z tą różnicą, że przed każdą operacją użytkownik proszony jest o potwierdzenie działania, `-prune` powoduje, że `find` nie wchodzi do żadnego z napotkanych katalogów.

Składnia `find` umożliwia tworzenie złożonych wyrażeń, łączenia kryteriów. Domyślnie kryteria łączone są za pomocą logicznej koniunkcji (AND): `(-a)`. Wszystkie kryteria muszą być spełnione, aby plik został uznany za zgodny z kryteriami. Operator łączenia alternatywy logicznej (OR): `-o`, natomiast operator negacji to: `!` Istnieje możliwość wykorzystania nawiasów w celu grupowania kryteriów `(\ \)`.

Ważnym parametrem polecenia `find` jest parametr `-print0`, w przypadku jego użycia, nazwy znalezionych plików nie są rozdzielane znakiem nowej linii a znakiem null. Rozpatrzmy przykład poniżej, w którym użyjemy dwóch plików: *raport czerwiec.txt* i *raportczerwiec.txt*.

```
adam@laptop:~$ find . -name "raport*" | xargs rm
rm: nie można usunąć './raport': No such file or directory
rm: nie można usunąć 'czerwiec.txt': No such file or directory
```

Poleceniu `rm` nie udało się usunąć pliku *raport czerwiec.txt*, ponieważ zawiera on spację w nazwie i jego nazwa została w tym miejscu rozdzielona.

```
find . -name "raport*" -print0 | xargs -0 rm
```

Polecenie `print` otrzymało opcję `-print0`, natomiast polecenie `xargs` opcję `-0`, dzięki czemu bez problemu usunięty został plik zawierający spację. **Uwaga!** Opcja `-print0` dostępna jest tylko w wersji [GNU](#) polecenia `find`.

Zmienne środowiskowe⁸

Jak podaje Wikipedia, zmienne środowiskowe to zbiór dynamicznych wartości, wpływających na sposób w jaki działać będą uruchomione procesy (programy). Zmienne systemowe występują w każdym systemie operacyjnym Unix, Unix-like oraz M3S DOS i Windows. Oczywiście różne systemy korzystać będą z różnych zbiorów zmiennych, jednak najczęściej większość zmiennych będzie wspólna.

Początkujący użytkownicy bardzo często boją się korzystać ze zmiennych systemowych w obawie przed uszkodzeniem systemu, traktując je jako rzecz nie do ogarnięcia. Oczywiście jest to podejście jak najbardziej błędne. Bash pozwala również na tworzenie tzw. zmiennych powłoki, które działają na zasadzie identycznej jak zmienne środowiskowe przy czym ich zasięg dotyczy tylko powłoki, w której zostały utworzone.

⁸ <http://jakilinux.org/konsola/zmienne-srodowiskowe/>

1. Niektóre zmienne środowiskowe

Zmienna	Opis
---------	------

PATH	Zmienna zawiera oddzieloną dwukropkami listę katalogów w której, powłoka będzie szukać programu którego nazwę do wykonania wprowadził użytkownik, jeżeli taki program nie zostanie znaleziony, powłoka wyświetli komunikat „polecenie nie odnalezione”
EDITOR	Domyślny edytor tekstu, zmienna wykorzystywana przez niektóre programy np. przez klienta poczty mutt. W jego przypadku emaily edytowane są w programie określonym przez tę zmienną
SHELL	Powłoka wykorzystywana przez użytkownika
USER	Nazwa użytkownika
SHLVL	Liczba uruchomionych powłok
TERM	Domyślnie uruchamiany emulator terminala
HOME	Domyślna ścieżka do katalogu domowego użytkownika
UID	Unikalny liczbowy identyfikator zalogowanego użytkownika
\$LANG, \$LC_ALL	Zmienne przechowujące ustawienia językowe (locale)

2. Wyświetlanie wartości zmiennej

Aby wyświetlić wartość jaką ma dana zmienna, używamy znanego już polecenia `echo $zmienna`.

```
adam@laptop:~$ echo $USER $UID $SHELL $HOME
adam 1000 /bin/bash /home/adam
```

Aby wyświetlić wszystkie używane zmienne, można użyć polecenia `env`. W poniższym przykładzie pokazany jest tylko fragment bardzo długiego wyjścia. Alternatywnie można użyć wbudowanego w basha polecenia `set`.

```
adam@laptop:~$ env
SSH_AGENT_PID=5605
TERM=xterm
DESKTOP_STARTUP_ID=
SHELL=/bin/bash
GTK_RC_FILES=/etc/gtk/gtkrc:/home/adam/.gtkrc-1.2-gnome2
WINDOWID=58896938
GTK_MODULES=gail:atk-bridge
USER=adam
USERNAME=adam
DESKTOP_SESSION=gnome
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
PWD=/home/adam
LANG=pl_PL.UTF-8
GDMSESSION=gnome
HOME=/home/adam
```

SHLVL=2
LOGNAME=adam

3. Tworzenie nowych zmiennych

Nową zmienną tworzy się wpisując jej nazwę i wartość, jaką będzie ona miała. Na przykład `zmienna=wartość`. Zmienne są czułe na wielkość liter, tak, więc `ZMIENNA` nie będzie tym samym co `zmienna` pisana małymi literami. Zwyczajowo przyjęto się pisać nazwy zmiennych wielkimi literami.

```
adam@laptop:~$ ZMIENNA=wartość
adam@laptop:~$ echo $ZMIENNA
wartość
```

39

Utworzona powyżej zmienna powłoki ma pewną nieprzyjemną cechę, mianowicie dostępna będzie jedynie w powłoce, w której została utworzona, a programy uruchomione z tej powłoki, nie będą miały do niej dostępu. Aby obejść to ograniczenie, należy wyeksportować zmienną za pomocą polecenia `export`.

```
adam@laptop:~$ ZMIENNA=wartość
adam@laptop:~$ ZMIENNA2=wartość2
adam@laptop:~$ export $ZMIENNA2
adam@laptop:~$ export ZMIENNA2
adam@laptop:~$ bash #uruchamiam nową powłokę
adam@laptop:~$ echo $ZMIENNA1 $ZMIENNA2
wartość2
```

W powyższym przykładzie wyeksportowałem tylko jedną zmienną, jej wartość jest dostępna w nowo uruchomionej powłoce.

4. Zapamiętywanie wartości zmiennej

Wszystkie nowo utworzone zmienne, zarówno te wyeksportowane jak i niewyeksportowane, będą dostępne tak długo, jak użytkownik będzie zalogowany. Aby zmienna dostępna była po ponownym zalogowaniu się, należy dodać ją do ukrytego pliku `.profile`. Jeżeli zmienna ma być dostępna dla każdego użytkownika systemu, należy dodać ją do pliku `/etc/profile`. Oto przykładowa linia z pliku `.profile` (konfiguracja serwera anoncvs do pobierania plików źródłowych systemu NetBSD).

```
export CVSROOT=anoncvs@anoncvs.netbsd.org:/cvsroot
```

5. Usuwanie zmiennych

Aby usunąć zmienną, należy użyć polecenia `unset`.

```
adam@laptop:~$ ZMIENNA=wartość
adam@laptop:~$ echo $ZMIENNA
```

wartość

```
adam@laptop:~$ unset ZMIENNA
```

```
adam@laptop:~$ echo $ZMIENNA
```
