



<http://jakilinux.org>

Wszystko o konsoli

Adam Zieliński
adz@jakilinux.org

wersja 0.4 październik 2007

Podziękowania

Chciałbym podziękować Borysowi "michukowi" Musielakowi za umożliwienie mi publikacji niniejszego opracowania na łamach wortalu jakilinux.org jak i za całą okazane wsparcie. Tomaszowi "t.ziel" Zielińskiemu za pomoc merytoryczną i cierpliwość okazaną przy korekcie błędów, anonimowemu użytkownikowi wiki jakilinux.org za opis polecenia less oraz cat.

Wersje

wersja	autor	data	zmiany
0.1	adz	15.09.07	Wersja do użytku prywatnego
0.2	adz	22.09.07	Wersja do użytku prywatnego
0.3	adz	05.10.07	Wersja do użytku prywatnego
0.4	adz	09.10.07	Pierwsza wersja udostępniona publicznie

Licencja

Creative Commons: Uznanie autorstwa-Użycie niekomercyjne 2.5 Polska

Spis treści

1. Konsola - podstawy	3
1.1. Czym jest powłoka, czym jest terminal, czym jest konsola?	3
1.2. bash	3
2. Zarządzanie plikami	4
2.1. ls	4
2.2. cd	5
2.3. mkdir, rmdir	6
2.4. cp, mv, rm	6
2.5. ln	7
2.6. touch	8
2.7. df, du	8
2.8. echo	9
2.9. pwd	9
2.10. cat	9
2.11. wc, head, tail	10
2.12. less	10
3. Strumienie, potoki i przekierowania	11
3.1. Potoki	11
3.2. Strumienie i przekierowania	11
3.3. xargs	12
4. Wyszukiwanie, wyrażenia regularne	12
4.1. Wyrażenia regularne	13
4.2. Wyrażenia regularne a znaki globalne	13
4.3. grep	14
4.4. find	15
5. Zmienne środowiskowe	17
5.1. Niektóre zmienne środowiskowe	18
5.2. Wyświetlanie wartości zmiennej	18
5.3. Tworzenie nowych zmiennych	19
5.4. Zapamiętywanie wartości zmiennej	19
5.5. Usuwanie zmiennych	19

1. Konsola - podstawy

1.1. 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 zacieraają. 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 koszt 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, tsh.

1.2. bash

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 #.

```
adz@laptop:~#  
[adz@laptop ~]#  
#
```

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

2. Zarządzanie plikami

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.

2.1. 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).

```
$ pwd  
/home/adam/Dokumenty/jakilinux.org/  
$ ls  
przykład.txt  wszystko_o_konsoli.txt  
$ 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.

```
$ 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:

- -rw-rr 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`.

```
$ 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`.

```
$ ls -R
.:
katalog1  katalog2  przykład.txt  wszystko_o_konsoli.txt

./katalog1:
plik1  plik2

./katalog2:
plik3  plik4
```

2.2. 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.

```
$ cd /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.

```
$ cd ~
$ pwd
/home/adam
```

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

```
$ cd ~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 unixowych zawiera 2 elementy `.` i `..`. Pojedyncza kropka `.` jest to dowiązanie katalogu do niego samego, natomiast dwukropka `..` jest dowiązaniem do katalogu nadrzędnego w drzewie katalogów.

```
$ cd ..
$ 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.

2.3. `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.

```
$ mkdir katalog
$ ls
katalog  przykład.txt  wszystko_o_konsoli.txt
$ rmdir katalog/
$ ls
przykład.txt  wszystko_o_konsoli.txt
```

2.4. `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`).

```
$ ls
katalog1 katalog2 katalog3 przykład.txt wszystko_o_konsoli.txt
$ cp -R katalog2/ katalog3/
$ ls katalog3/
katalog2
$ rm -r katalog3/
$ ls
katalog1 katalog2 przykład.txt wszystko_o_konsoli.txt
```

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

2.5. 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 twardej dowiązań spadnie do zera.

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

```
$ cat plik
Przykład dowiązań.
$ ln -s plik dowiazanie_symboliczne
$ 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
$ rm plik
$ ls
dowiazanie_symboliczne
```

W powyższym przykładzie można zaobserwować działanie dowiązań symbolicznych. Polecenie `ls` wskazuje, że dla pliku nie zmieniła się liczba twardej dowiązań. Zupełnie inaczej jest na przykładzie poniżej, gdzie liczba twardej dowiązań wynosi 2, po usunięciu pliku spada do 1, plik cały czas jest dostępny.

```

$ ln plik dowiazanie_twarde
$ 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
$ rm plik
$ ls -l
-rw-r--r-- 1 adam adam 22 2007-06-01 19:11 dowiazanie_twarde

```

2.6. 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.

```

$ ls -l przykład.txt
-rw-r--r-- 1 adam adam 0 2007-05-30 11:31 przykład.txt
$ touch przykład.txt
$ 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
$ touch nowy.txt
$ 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 MMD-Dhhmm.

```

$ 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
$ touch -t 200706101200 nowy.txt
$ touch -d "last monday" przykład.txt
$ touch -d "2 days ago 12:00" wszystko_o_konsoli.txt
$ 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

```

2.7. 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:

```
$ 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:

```
$ du -h error.txt
4,0K   error.txt
```

2.8. echo

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

```
$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:

```
$echo -n Hello World!
Hello World!$
```

W przypadku zamknięcia tekstu w cudzysłowy, tekst będzie interpretowany bezpośrednio.

2.9. pwd

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

```
$ pwd
/home/adam
```

2.10. 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:

```
$ 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`.

```
$ cat plik.txt
Bardzo ciekawy tekst.
```

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

```
$ 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
```

2.11. `wc`, `head`, `tail`

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

```
$ 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.

```
$ head -n 1 przykład.txt
Pierwsza linia tekstu!
$ tail -n 2 przykład.txt
Przed ostatnia linia tekstu!
Ostatnia linia tekstu!
```

2.12. `less`

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

```
$ 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` umożliwi wyświetlenie numeru każdej linii przy lewej krawędzi ekranu, co widać poniżej.

```
$ cat plik.txt | less
```

Program opuszczamy wpisując `:q`.

3. Strumienie, potoki i przekierowania

3.1. 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 stanowić 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 się liczba plików i katalogów w danym katalogu.

```
$ ls
nowy.txt  przykład.txt  wszystko_o_konsoli.txt
$ ls | wc -l
3
```

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

3.2. 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:

- `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 przekierujemy, 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 do pliku i na koniec przekieruję standardowe wejście z klawiatury do pliku.

```

$ cat nieistniejacy_plik.txt 2> error.txt
$ cat error.txt
cat: nieistniejacy_plik.txt: No such file or directory
$ cat > polecenie.txt
ls -l
(CTRL+D)
$ 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

```

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.

```
$ cat nieistniejacy_plik.txt 2> /dev/null
```

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

3.3. 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`.

```
$ 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 przekopowane 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 `null`). 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`.

4. Wyszukiwanie, wyrażenia regularne

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.

4.1. 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
\\$	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

4.2. 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*).

*	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".

```
$ ls
aa abc          nowy.txt       przykład.txt
ab error.txt    polecenie.txt  wszystko_o_konsoli.txt
$ rm a*
$ 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.

```
$ ls
aa abc          nowy.txt       przykład.txt
ab error.txt    polecenie.txt  wszystko_o_konsoli.txt
$ rm [^b-z]*
$ ls
error.txt      nowy.txt      polecenie.txt  przykład.txt  wszystko_o_konsoli.txt
```

4.3. 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.

```
$ 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, `wyrazenia.txt` o zawartości:

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

Tak więc:

```
grep ^[1-6]..a wyrazenia.txt
4. auto
6. 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$ wyrazenia.txt
5. płyta
```

```
$ grep a.*a wyrazenia.txt
6. 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.

4.4. 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:

```
$ find . -name linux
./Downloads/Firefox/vmware-server-distrib/perl5/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.

```
$ find . -type d
```

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

```
$ 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.

```
$ find . -size +100M -size -200M
./plan9_07010zip
```

Pozostałe kryteria wyszukiwania przedstawia poniższa tabela.

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

Opcje liczbowe można poprzedzać znakami `+` i `-`, które oznaczają odpowiednio “więcej niż” oraz “mniej niż”, podobnie jak miało to miejsce w kryterium `time` 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.

```
$ 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 wyszukanych 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 *raport-czerwiec.txt*.

```
$ 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 `find` 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`.

5. Zmienne środowiskowe

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 MS DOS i Windows. Oczywiście różne systemy korzystają 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.

5.1. 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 odnaleziono"
EDITOR	Domyślny edytor tekstu, zmienna wykorzystywana przez niektóre programy np. przez klienta poczty mutt . W jego przypadku emaile 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)

5.2. Wyświetlanie wartości zmiennej

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

```
$ 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`.

```
$ 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
```

5.3. 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 przyjęło się pisać nazwy zmiennych wielkimi literami.

```
$ ZMIENNA=wartość
$ echo $ZMIENNA
wartość
```

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`.

```
$ ZMIENNA=wartość
$ ZMIENNA2=wartość2
$ export $ZMIENNA2
$ export ZMIENNA2
$ bash #uruchamiam nową powłokę
$ echo $ZMIENNA1 $ZMIENNA2
wartość2
```

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

5.4. 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
```

5.5. Usuwanie zmiennych

Aby usunąć zmienną, należy użyć polecenia `unset`.

```
$ ZMIENNA=wartość
$ echo $ZMIENNA
wartość
$ unset ZMIENNA
$ echo $ZMIENNA
```