Sieci komputerowe

Wstęp do systemu Linux/UNIX, część II

Ewa Burnecka / Janusz Szwabiński

ewa@ift.uni.wroc.pl / szwabin@ift.uni.wroc.pl

Wstęp do systemu Linux/UNIX, część II

- Przeszukiwanie plików: `grep`
- Wyrażenia regularne
- Poszukiwanie plików: `find`
- Mała dygresja: `vi`
- Przetwarzanie plików: `awk`
- Skypty w powłoce

Materiały w Sieci

- Steve Mansour, *„A Tao of Regular Expressions”*, `http://sitescooper.org/tao-regexps.html`
- Mendel Cooper, *„Advanced Bash-Scripting Guide”*, `http://www.tldp.org/LDP/abs/html/`

Przeszukiwanie plików: `grep`

```
grep [opcje] wzorzec plik(i)
```

Opcja	Znaczenie
`-c`	liczba linii zawierających wzorzec
`-i`	nie rozróżniaj małych/dużych liter
`-n`	numer wiersza zawierającego wzorzec
`-d akcja`	wykonaj `akcję` na katalogu (np. `recurse`, `skip`)
`-f plik`	wczytaj wzorzec z pliku

`-l`	wyświetl nazwę pliku zawierającego wzorzec
`-r`	przeszukaj pliki w podkatalogach
`-v`	odwrócone przeszukiwanie

Prosty przykład

Plik tekst.txt:

```
To jest kot.
Ala ma kota.
Kot jest czarny.
Pies macha ogonem.
```

- `grep -i -n "kot" *`

  ⇒
  ```
  1:  To jest kot.
  2:  Ala ma kota.
  3:  Kot jest czarny.
  ```

- `grep -i -v -n "kot" *`

  ⇒
  ```
  4:  Pies macha ogonem.
  ```

Wyrażenia regularne

Metaznak	Opis
.	dowolny pojedynczy znak
$	znak końca linii
^	znak początku linii
*	dowolna (również zerowa) liczba wystąpień znaku poprzedzającego *
\	cytowanie (wyłączenie specjalnego znaczenia)
[ ]	dowolny ze znaków w nawiasie (jeśli pierszym znakiem jest ^ - odwrócenie wyrażenia)

`\<` `\>`	znaki na początku lub końcu słowa
`|`	jeden lub drugi wzorzec (`egrep`)
`+`	jedno lub więcej wystąpień poprzedzającego wyrażenia lub znaku
`?`	żadne lub jedno wystąpienie poprzedzającego wyrażenia lub znaku (`egrep`)
`\(` `\)`	umieszcza wzorzec w specjalnym schowku (do schowka odwołujemy się przez `\1` do `\9`)

`\{i\}`	`i` wystąpień poprzedzającego wyrażenia
`\{i,j\}`	między `i` a `j` wystąpień

Klasy znaków

Klasa	Znaki dopasowywane
`[:alnum:]`	znaki alfanumeryczne
`[:alpha:]`	znaki alfabetu
`[:blank:]`	spacja lub tabulacja
`[:cntrl:]`	znaki sterujące
`[:digit:]`	cyfry dziesiętne
`[:graph:]`	znaki drukowalne i widoczne
`[:lower:]`	znaki pisane małą literą
`[:print:]`	znaki drukowalne

`[:space:]`	znaki białe
`[:upper:]`	znaki pisane dużą literą
`[:xdigit:]`	cyfry szesnastkowe

Przykłady wyszukiwania

```
grep [opcje] 'wzor' plik
```

Wzór	Działanie
`kot`	ciąg znaków *kot*
`^kot`	*kot* na początku
`kot$`	*kot* na końcu wiersza
`^kot$`	*kot* jako jedyne słowo w wierszu
`[Kk]ot`	*Kot* lub *kot*
`k[ioa]t`	*kit* lub *kot* lub *kat*

`k[^ioa]t`	druga litera to dowolny znak z wyjątkiem *i*, *o* lub *a*
`k.t`	druga litera jest dowolnym znakiem
`^...$`	dowolny wiersz zawierający trzy znaki
`^\.`	dowolny wiersz zaczynający się kropką
`^\.[a-z]\{2\}`	wiersz zaczynający się kropką, po której następują dwa znaki pisane małą literą
`^[^.]`	wiersz nie rozpoczynający się kropką
`kot*`	*kot*, *kota*, *kotara* itd.

`[A-Z].*`	duża litera, po której następuje zero lub więcej znaków
`firm(a|y)`	*firma* lub *firmy* (`egrep`)

`grep` w innych zastosowaniach

- `ps aux | grep szwabin`
- `ps aux | egrep 'szwabin|PID'`
- `rpm -qa | grep 'mozilla'`
- `w | cut -d " " -f 1 - | grep -v USER | sort -u`

Wyszukiwanie plików: `find`

```
find [ścieżka] [kryteria] [akcje]
```

Kryterium	Znaczenie
`-atime n`	ostatnie żądanie dostępu do pliku *n* dni temu
`-mtime n`	plik modyfikowany *n* dni temu
`-newer nazwa`	plik modyfikowany później niż plik *nazwa*
`-size n`	rozmiar pliku wynosi *n* bloków 512–bajtowych

`-type c`	plik jest określonego typu (np. *f* - plik zwykły)
`-fstype typ`	umożliwia podanie typu systemu plików
`-name nazwa`	plik o nazwie *nazwa*
`-perm p`	prawa dostępu do pliku wynoszą *p*
`-user usr`	plik, którego właścicielem jest *usr*
`-group grp`	plik, którego grupą jest *grp*
`-nouser`	właściciel pliku nie figuruje w pliku haseł
`-nogroup`	grupa pliku nie figuruje w pliku grup

Typowe akcje

Akcja	Znaczenie
`-print`	wyświetl nazwę znalezionego pliku
`-ls`	wyświetl długą informację o pliku
`-exec cmd`	wykonaj na znalezionym pliku polecenie *cmd*
`-ok cmd`	jak wyżej, ale z żądaniem potwierdzenia
`-xdev`	ograniczenie wyszukiwania do systemu plików zawierającego katalog podstawowy
`-prune`	zaniechaj wyszukiwania w znajdowanych katalogach

- w wielu systemach opcja `-print` uznawana jest za domyślną
- polecenia wykonywane na znajdowanych plikach za pośrednictwem `-exec` lub `-ok` muszą być zakończone za pomocą `\;`
- klamry `{}` umieszczone za nazwą wykonywanego polecenia symbolizują przekazywanie nazw znajdowanych plików jako jego argumentów

Przykłady

- `find . -name '*.f90' -print`
- `find /home -size +2048 -mtime +30 -exec ls -l {} \;`
- `find /home/szwabin -type f -name core -exec rm {} \;`
- `find / -type f \(-perm 2000 -o -perm -4000 \) -print`
- `find ~ -name '*.f90' -exec mv {} ~/f90 \;`

Mała dygresja, czyli `vi`

Klawisz	Znaczenie
`i`	przejdź do trybu wstawiania
`Esc`	wyjdź z trybu wstawiania
`h/j/k/l`	przesuń kursor w lewo/dół/górę/prawo
`x`	usuń znak
`dd`	usuń bieżącą linię
`p`	wstaw skasowaną linię za linią bieżącą
`u`	undo
`:`	przejdź do trybu złożonych poleceń

Tryb poleceń złożonych

`:w nazwa`	zapisz tekst w nowym pliku *nazwa*
`:wq`	zapisz zmiany i wyjdź
`:q!`	wyjdź nie zapisując zmian
`:help`	pomoc

Przydatne polecenie: `vimtutor`

awk

```
awk [opcje] 'skrypt'  plik(i)
awk [opcje] -f plik_skryptu plik(i)
```

Opcja	Znaczenie
`-Ffs`	ustala znak rozdzielający pola na *fs*
`-v zmienna=wartosc`	przypisuje *zmiennej wartość* przed wykonaniem skryptu

Skrypty w `awk`

```
wzorzec {akcja}
```

Wzorce:

```
/wyrażenia regularne/
wyrażenie relacyjne
operator pasowania do wyr. regularnych
BEGIN
END
```

- znaki `^` i `$` odnoszą się do początku i końca ciągu w polu (nie w wierszu!)
- dwa operatory pasowania do wyrażeń regularnych: `~` (zgodne) i `!~` (niezgodne)
- wzorzec `BEGIN` (`END`) służy do określenia akcji wykonywanych przed (po) przetworzeniem pierwszego wiersza danych wejściowych

Akcje w `awk`

- przypisanie zmiennych lub tablic
- polecenie drukowania
- wbudowane funkcje
- polecenie przepływu
- funkcje użytkownika

Przykłady

`{print $1}`	drukowanie pierwszego pola każdego wiersza
`/wzorzec/ {print $1}`	drukowanie 1. pola wierszy zawierających *wzorzec*
`BEGIN {FS="\n"; RS=""}`	wiersz pojedynczym polem
`NF>2`	więcej niż dwa pola w rekordzie
`$1 ~ /ważne/ {print $3,$2}`	drukowanie pól 3 i 2, jeśli pole 1 ma wartość *ważne*

`/wzorzec/ {x++}` `END {print x}`	zliczanie i drukowanie liczby wystąpień *wzorca*
`{suma +=$1}` `END{print suma}`	sumowanie liczb w pierwszej kolumnie
`length($0)<20`	drukowanie wierszy krótszych niż 20 znaków

(Niektóre) Wbudowane zmienne

`FILENAME`	nazwa bieżącego pliku
`FS (OFS)`	znak oddzielający pola (na wyjściu)
`NF`	liczba pól w bieżącym rekordzie
`NR`	numer bieżącego rekordu
`RS (ORS)`	znak oddzielający rekordy (wyjściowe)
`$0`	cały rekord danych wejściowych
`$n`	*n*-te pole bieżącego rekordu

Operatory

```
(...)
$
++ --
+ - !
* / % + - ^ **
[space]
< > <= >= != ==
~ !~
in
&& ||
?:
```

Wybrane funkcje i polecenia

Funkcje arytmetyczne	`sin`, `cos`, `log`, `sqrt`, `rand`, `atan`, `int`
Funkcje na łańcuchach znaków	`index`, `length`, `match`, `split`, `tolower`, `toupper`
Instrukcje sterujące	`for`, `while`, `do/while`, `if`, `break`, `exit`, `return`, `continue`
Funkcje czasu	`systime`, `strftime`

Przykłady prostych zastosowań `awk`

- `ls -l | awk '{print $8,$5}'`
- `awk -F: '{print $1,$3,$4}' /etc/passwd`
- `awk 'NF>0' wykaz.txt`
-
```
df -h | sort -rnk 5 | head -2 |\
awk '{print "partycja " $6 \
          ":\t zapelniona w " $5}'
```

Piszemy skrypty w BASHu

Przykład

```
szwabin@defiant:~> cat -n mojsys.sh
  1  #!/bin/bash
  2  clear
  3  echo "Witaj $USER!"\\ %$
  4  echo "Oto informacje dostarczone przez skrypt mojsys.sh..."
  5  echo "Dzisiaj jest  `date`, `date +"%V"`. tydzień roku."
  6  echo "Zalogowani użytkownicy: "
  7  w | cut -d " " -f 1 - | grep -v USER | sort -u
  8  echo "System `uname -s` na procesorze `uname -m`."
  9  echo "Status systemu:"
  10  uptime
  11 echo "...zrobione!"
```

- skrypt to zwykły plik tekstowy, zawierający w pierwszej linii ciąg znaków "`#!/bin/bash`"
- jeśli w jednej linii chcemy umieścić więcej poleceń, oddzielamy je średnikami
- aby skrypt można było uruchomić, należy nadać mu atrybut wykonywalności

Zmienne

Przypisanie wartości

- `var=123`
- `var="abc efg"`

Rozwijanie zmiennych

- `var1="$var 345"`

  (apostrofy zamiast cudzysłowów zapobiegają rozwijaniu)

- `sciezka="$HOME/skrypty/"`

Przypisanie wyniku działania funkcji

- `stan=`uptime``

Niektóre wbudowane zmienne

`$HOME`	katalog domowy
`$HOSTNAME`	nazwa komputera
`$PATH`	ścieżka dostępu do binariów
`$PWD`	bieżący katalog
`$USER`	bieżący użytkownik
`$?`	wynik ostatniego polecenia
`$!`	numer ostatnio uruchomionego w tle procesu
`$$`	numer procesu aktualnej powłoki

`$0`	ścieżka do skryptu
`$#`	liczba argumentów skryptu
`$1` do `$9`	kolejne argumenty wywołania skryptu
`$*` lub `$@`	wszystkie argumenty wywołania skryptu

Wybrane operacje na łańcuchach znaków

`${#lancuch}`	długość łańcucha znaków
`${zmienna#wzorzec}`	jeśli *zmienna* zawiera na początku *wzorzec*, zostanie on usunięty
`${zmienna%wzorzec}`	jeśli *zmienna* zawiera na końcu *wzorzec*, zostanie on usunięty

Podstawianie parametrów

`${var}`	wartość zmiennej *var*
`${var:-default}`	jeśli *var* nie jest zdefiniowana, lub jest pusta, podstaw *default*
`${var:=default}`	jeśli zmienna *var* nie jest zdefiniowana, podstaw *default*
`${var:+other}`	jeśli zmienna *var* jest zdefiniowana, podstaw *other*
`${var:?errmsg}`	jeśli *var* nie jest zdefiniowana, wypisz *errmsg*

Instrukcje warunkowe

- `program1 && program2`
  (`program2` zostanie uruchomiony, jeśli `program1` zwróci 0)
- `program1 || program2`
  (`program2` zostanie uruchomiony, jeśli `program1` zwróci wartość $\neq 0$)

Przykład

```
cat plik | grep -q 'kot' && echo "kot się znalazł!" ||\
echo "nie ma kota"
```

if

```
if test; then
   instrukcje1
elif test; then
   instrukcje2
else
   instrukcje3
fi
```

Typowe testy

Test	Prawda, jeśli ...
`[ -d plik ]`	katalog *plik* istnieje
`[ -e plik ]`	*plik* istnieje
`[ -f plik ]`	„zwykły” *plik* istnieje
`[ -r plik ]`	*plik* istnieje i można go czytać
`[ -s plik ]`	*plik* istnieje i ma rozmiar > 0
`[ plik1 -nt plik2 ]`	*plik1* nowszy niż *plik2*
`[ -z string ]`	*string* ma długość równą zeru
`[ -n string ]`	*string* ma długość > 0

`[ string1=string2 ]`	oba łańcuchy są jednakowe
`[ string1!=string2 ]`	oba łańcuchy są różne
`[ !expr ]`	wyrażenie *expr* jest fałszywe
`[ expr1 -a expr2 ]`	oba wyrażenia są prawdziwe
`[ expr1 -o expr2 ]`	przynajmniej jedno wyrażenie jest prawdziwe
`[ arg1 -eq arg2 ]`	$arg1 = arg2$
`[ arg1 -lt arg2 ]`	$arg1 < arg2$
`[ arg1 -gt arg2 ]`	$arg1 > arg2$

Przykład

```
if [ $# -ne 2 ]; then
   echo "Ten skrypt potrzebuje dokładnie"
   echo "dwóch parametrów"
   exit 1
fi
```

case

```
case $zmienna in
    "aaa")
        instrukcje1;;
    "bbb")
        instrukcje2;;
    *)
        instrukcje3;;
esac
```

Pętle

```
while test; do
   instrukcje
done
```

```
for i in test; do
   instrukcje
done
```

Przykład

```
for i in *.eps; do
   mv $i ${i%.*}.ps
done
```

Wczytywanie danych

```
read [opcje] zmienna1 zmienna2 ... zmiennaN
```

- jeżeli użyjemy `read` bez podania zmiennej, wówczas dane zostaną przypisane zmiennej `$REPLY`