Wstęp do Informatyki dla bioinformatyków

Wykład 11 - Programowanie w BASHu Podstawy

Bartek Wilczyński

18.1.2016

Krótka historia powłoki

- Thompson shell 1971 (Ken Thompson), Unix
- Bourne shell 1977 (Stephen Bourne), Unix
- Bourne Again Shell (BASH) 1989 (Brian Fox), Projekt GNU
- Oprócz tego konkurencyjne:
  - ash (Almquist, bardzo mały), ksh (Korn, podobny do bash), busy box (jeszcze mniejszy)
  - Csh (C shell, składnia podobna do c), tcsh (tenex c shell)

Jak dowiedzieć się czegoś w bash'u

- Manual: man
  - `man man`
  - `man python`
  - `man bash`
- Nowszy manual z linkami: info
  - `info emacs`
- Szukanie w manualu: apropos
  - `apropos ssh`
- Słowa kluczowe programów w BASH: help
  - `help if`
  - `help for`

Wypisywanie na ekran

- Żeby wypisać coś na standardowym strumieniu wyjścia możemy użyć poleceń `echo` i `cat`:
  - `echo "Ala ma kota"`
  - `cat /etc/passwd`
- Cat wypisuje cały plik, jeśli chcemy stronicować to możemy użyć programu more, albo nawet nieco bogatszego less (cofanie, wyszukiwanie itp.)
  - `more /etc/passwd`
  - `less /ect/passwd`

Zmienne systemowe - środowisko

- Zmienne oznaczamy przez prefix $, np. `$PATH`, `$PWD` itp
- Zmienne przechowują napisy
- Przypisanie przez zmienna=wartość (przy czym po lewej stronie nie używamy $): np.

  `PATH=$PATH:/opt/bin/`
- `export` – pozwala wpisać zmienne do środowiska
- `env` – wyświetla zawartość środowiska

Co nieco o cudzysłowach w BASHu

- Mamy 3 rodzaje cudzysłowów: ' ” `
- “Słabe” cudzysłowy ”:
  - Spacja nie dzieli argumentów
  - Nie działa pattern matching
- “Mocne” cudzysłowy ':
  - Zmienne nie są interpretowane
- “Odwrotne” cudzysłowy `:
  - Wykonują polecenie w cudzysłowach i “podstawiają” standardowe wyjście w miejsce komendy
- Wewnątrz napisów używamy \' \” \` itp.

Kody zakończenia procesu

- Każdy proces kończąc zwraca swojemu rodzicowi tzw. Kod zakończenia (and. Error code)
- 0 – oznacza prawidłowe zakończenie
- errorcode!=0 → jakiś problem podczas wykonania procesu
- Komendy `true, false` pozwalają na wykenerowaniu odpowiednio kodu zerowego i niezerowego
- Ta semantyka jest **odwrotna** niż w pythonie, gdzie 0 jest interpretowane jako False

Teraz trochę o nawiasach

- Nawiasy [ ] służą do testowania warunków, dokładniej [jest aliasem na polecenie test, np.
  - `[ “$pwd” = “/home/bartek” ]`
- Nawiasy ( ) służą do otwierania zagnieżdżonej powłoki
  - `( export $PATH=/opt/bin/; date ) ; date`
- Nawiasy { } służą do zaznaczania syntaktycznej całości – np. Nazw zmiennych, albo fragmentów komend:
  - `echo ${pwd}.exe`
  - `{ date; ls -l } > logfile`

Instrukcja warunkowa if

```
bartek@telomere:~$ help if
if: if COMMANDS; then COMMANDS; [ elif COMMANDS; then COMMANDS; ]... [ else COMMANDS; ] fi
    Execute commands based on conditional.

    The `if COMMANDS' list is executed.  If its exit status is zero, then the
    `then COMMANDS' list is executed.  Otherwise, each `elif COMMANDS' list is
    executed in turn, and if its exit status is zero, the corresponding
    `then COMMANDS' list is executed and the if command completes.  Otherwise,
    the `else COMMANDS' list is executed, if present.  The exit status of the
    entire construct is the exit status of the last command executed, or zero
    if no condition tested true.

    Exit Status:
    Returns the status of the last command executed.
```

- `if [ “$PWD” = “/home” ]; then echo “tak”; else echo $PWD; cd /home; fi`

Iteracje – komenda for

```
bartek@telomere:~$ help for
for: for NAME [in WORDS ... ] ; do COMMANDS; done
    Execute commands for each member in a list.

    The `for' loop executes a sequence of commands for each member in a
    list of items.  If `in WORDS ...;' is not present, then `in "$@"' is
    assumed.  For each element in WORDS, NAME is set to that element, and
    the COMMANDS are executed.

    Exit Status:
    Returns the status of the last command executed.
```

- np.
  - `for  i in 1 2 3 4 5 6; do echo $i; done`
  - `for x in \`ls /root/\`;`

    `do echo $x; cp $x /tmp;`

    `done`

Wyrażenia arytmetyczne

- Zmienne są napisami, więc aby coś policzyć, używamy składni $(( )) lub polecenia let

```
bartek@telomere:~$ echo $x+$y
5+2
bartek@telomere:~$ echo $(($x+$y))
7
```

```
bartek@telomere:~$ let x=x+y
bartek@telomere:~$ echo $x $y
7 2
bartek@telomere:~$ echo $x++
7++
```

```
bartek@telomere:~$ echo $((x++))
7
bartek@telomere:~$ echo $x
8
```

Sekwencyjna pętla for

- Aby uzyskać pętle for dla określonych przedziałów wartości możemy użyć dwóch niemal równorzędnych składni

```
bartek@telomere:~$ for (( i=1 ; i <=5; i++)) ; do echo $i ; done
1
2
3
4
5
bartek@telomere:~$ for i in `seq 1 5`; do echo $i; done
1
2
3
4
5
```

Wyrażenia przy pomocy `expr`

- Polecenie expr (man expr) pozwala obliczać różne wyrażenia (także mnożenie, op. logiczne)
- Także opercje length, index, substr, match

```
bartek@telomere:~$ expr "$x + $y"
8 + 2
bartek@telomere:~$ expr $x + $y
10
bartek@telomere:~$ expr length $x + $y
3
```

Przydatne polecenia do testowania

- `cut` – wycina kolumnę
- `head/tail` – podaje tylko n pierwszych/ostatnich wierszy z wejścia
- `sort` – sortuje wiersze alfabetycznie
- `uniq` – zwraca tylko unikalne wiersze, zakładając, że były posortowane
- `tee` – wyświetl na ekran I zapisz do pliku jednocześnie
- `grep` – wybierz tylko wiersze spełniające określone warunki (np. zwierające napis)