Wstęp do Informatyki dla bioinformatyków

Wykład 12 - Programowanie w BASHu Trochę bardziej zaawansowane

Bartek Wilczyński 25.1.2016

Organizacyjne

- **Bardzo proszę o wypełnienie ankiet w USOS**
- **Szczególnie zależy mi na komentarzach w polu otwartym**
- To pomoże nam przygotować zajęcia w przyszłym roku
- Wyniki kolokwium poprawkowego w tym tygodniu
- Egzamin 5. lutego o 10tej rano

Jeszcze o zasięgu zmiennych

- Różnica pomiędzy nawiasami okrągłymi (lokalna pod-powłoka) a klamrowymi (wydzielony fragment kodu, brak lokalnych zmiennych)

```
bartek@telomere:~$ x=1
bartek@telomere:~$ export y=1
bartek@telomere:~$  x=1; echo $x; { echo $x; x=2; echo $x; } ; echo $x;
1
1
2
2
bartek@telomere:~$  x=1; echo $x; ( echo $x; x=2; echo $x; ) ; echo $x;
1
1
2
1
```

Zmienne eksportowane

- Różnica pomiędzy $x a $y pojawia się przy wykonaniu polecenia rozpoczynającego nowy proces

```
bartek@telomere:~$ x=1
bartek@telomere:~$ export y=1
bartek@telomere:~$  x=1; echo $x; { echo $x; x=2; echo $x; } ; echo $x;
1
1
2
2
bartek@telomere:~$  x=1; echo $x; ( echo $x; x=2; echo $x; ) ; echo $x;
1
1
2
1
bartek@telomere:~$ bash -c "echo x: $x y: $y"
x: 1 y: 1
bartek@telomere:~$ bash -c 'echo x: $x y: $y'
x: y: 1
```

Łączenie poleceń

- Polecenie1 ; polecenie2 → wykonaj 1, a potem 2 niezależnie od kodu błędu
- Polecenie1 && polecenie2 → wykonaj 1, a jeśli się powiedzie (error=0) to też wykonaj 2
- Polecenie 1 || polecenie 2 → wykonaj 1, a jeśli się nie powiedzie (error<>0) wykonaj 2

Plan na dziś

- Wczytywanie danych poleceniem read;
- Przerywanie iteracji przy pomocy break
- Instrukcja case
- Pisanie funkcji w bashu
- Skrypty i ich argumenty
- Getopt
- Wyszukiwanie wzorca – grep
-

Polecenie read

- Read zmienna1 zmienna2 ...wczytuje na zmienne 1 I 2 wartości z linii standardowego wejścia
- Czyta tylko jedną linię
- Jeśli koniec pliku to zwraca błąd
- Jeśli zmiennych jest więcej to dzieli “po spacjach”
- Jeśli zmiennych za dużo, to zmienne będą puste
- Jeśli zmiennych za mało, to “reszta” wiersza będzie w ostatniej zmiennej

Instrukcja while

- While polecenie; do komendy; done
- Dopóki polecenie zwraca brak błędu, wykonuj komendy
- Np.

```
while read zmienna;
do
    echo $zmienna
done
```

Instrukcja break

- Przerywa działanie pętli for/while

```
while [ true ];
do {
   read zmienna;
   echo $zmienna;
   if [ "$zmienna" = "koniec" ] ;
     then break;
   fi } ; done
```

Parametry skryptu

- Pamiętamy o tym, że możemy użyć składni #!/bin/bash na początku pliku wykonywalnego, aby stworzyć skrypt basha (tak jak skrypty python'a)
- Taki program działa w osobnym procesie bash I otrzymuje pewne zmienne specjalne:
  - $0 $1 $2 …. parametry jak w sys.argv ($0 to nazwa programu)
  - $* lub $@ - wszystkie parametry, bez $0
  - $# - liczba parametrów (pomijając $0)
- Mamy też zmienną $? - kod wykonania ostatniej komendy

Shift i getopts

- Aby dobrać się do parametrów, przydatne są polecenia:
  - shift - przesuń listę parametrów o jeden, “zapominając o pierwszym”
  - getopts – polecenie przetwarzające opcje iteracyjnie
    - Wywołanie getopts “opis parametrow” nazwa
      - np. getopts “f:vhg:” - oznacza przyjmowanie opcji -f I-g z parametrami, a opcji -v I -h bez parametru

Instrukcja case

- Rozszerzony if, wygodny wraz z poleceniem getopts

```
#!/bin/bash

while getopts ":a" opt; do
  case $opt in
    a)
      echo "-a was triggered!" >&2
      ;;
    \?)
      echo "Invalid option: -$OPTARG" >&2
      ;;
  esac
done
```

Nieco bardziej skomplikowany przykład getopt z argumentami

```
#!/bin/bash

while getopts ":a:" opt; do
  case $opt in
    a)
      echo "-a was triggered, Parameter: $OPTARG" >&2
      ;;
    \?)
      echo "Invalid option: -$OPTARG" >&2
      exit 1
      ;;
    :)
      echo "Option -$OPTARG requires an argument." >&2
      exit 1
      ;;
  esac
done
```

Definiowanie funkcji

- Nazwa () {

  komenda

  }

  argumenty jak w skryptach Np.

```
wypisz () {
  echo $1;
}
```

Funkcje ze zmiennymi lokalnymi

- Inaczej zachowują się funkcje zdefiniowane w nawiasach okrągłych:
- `wypisz () { echo $1; x=$1; }`
- `wypisz_local () ( echo $1; x=$1; )`
- `Wypisz 20 → 20`
- `echo $x → 20`
- `wypisz_local 10 → 10`
- `echo $x → 20`

Wyszukiwanie wzorca

- Komenda grep pozwala na wybieranie wierszy z wejścia zawierających określone podsłowa
- Jeśli podamy nazwy plików jako argumenty, to otrzymamy na wyjściu wszystkie wersy wraz z informacją, które pliki zawierają takie linie
- Nowsze wersje grep'a pozwalają na użycie także wyrażeń regularnych we wzorcu
- Można także wyszukiwać “negatywnie”, czyli wiersze **nie zawierające** wzorca (-v)