Laboratorium 2, 3 i 4– Podstawy Bash (czas: 3 laby)

Miłosz Martynow
Laboratoria z Administracji i Architektury Systemów Operacyjnych
Wydział Fizyki Technicznej i Matematyki Stosowanej
Zakład Fizyki Teoretycznej i Informatyki Kwantowej
Zajęcia zdalne ze względu na kwarantanne ze względu na COVID-19 na Politechnice Gdańskiej

Bash jest jedną z najpopularniejszych powłok systemowych systemów UNIX, a co za tym idzie, jest programem komputerowym, który pośredniczy między użytkownikiem, a systemem operacyjnym. Innymi słowy Bash jest programem, który steruje innymi programami. Dzięki temu można powiedzieć, że Bash jest podstawowym narzędziem służącym do automatyzacji pracy na komputerze z systemem np. Linux Ubuntu. Ważne jest to, że Bash jako program interpretujący tekst z poleceniami, jest tym samym interpretowanym językiem programowania jak Python czy JavaScript. Oznacza to w ogólności, że potrzebujemy posiadać na komputerze zainstalowany interpreter, a nie kompilator. Interpreter zazwyczaj w systemach np. Ubuntu czy Raspbian jest już zainstalowany domyślnie. Dzięki temu Bash jest łatwo przenośny między wieloma platformami/komputerami. Pisząc kod w Bashu do wykonywania pewnej czynności tworzymy tzw. skrypt.

Bash umożliwia pracę ze wszystkimi komendami działającymi w terminalu i ze wszystkimi plikami dostępnymi na dysku, serwerze itp. Posiada także podstawowe instrukcje takie jak instrukcje warunkowe, pętle itp. Dzięki temu i dzięki temu, że jest on przenośny, możliwe jest automatyzowanie żmudnych prac na wielu platformach. Jeśli na danej platformie jest zainstalowany interpreter np. Pythona (który jest o wiele wydajniejszy – np. w pisaniu – do niektórych zadań), można wykonywać naprawdę skomplikowane czynności za pomocą paru kilkunastu/kilkudziesięciu-linijkowych skryptów.

Poniżej zostaną przedstawione zadania laboratoryjne. Każde zadanie będzie poprzedzone wstępem z głównymi informacjami koniecznymi do wykonania zadania. Każde następne może wymagać informacji z poprzedniego.

Zadanie 1. Hello world i nadawanie uprawnień

Aby wykonać jakiś program za pomocą Bash należy stworzyć plik z rozwinięciem **.sh** – można i bez tego rozwinięcia, ale dzięki niemu wiemy z czym mamy do czynienia. Wykonaj każdy podpunkt:

1. Otwieramy notatnik definiując nazwę – w tym przypadku **vim**. W przypadku vim, wciskamy **„i”** aby zacząć wprowadzać tekst. Po zapisaniu kodu, wciskamy **„esc”** i potem **shift+Z+Z** aby wyjść i zapisać:
   mimi@komp:~/Pulpit$ vi Z1.sh
2. W pierwszej linii skryptu zamieszczamy informacje z jakiego interpretera będziemy korzystać. W tym przypadku symbol „#” nie jest komentarzem, wywołujemy tzn **shebang** czyli „#!”, zaś ścieżka po nim to ścieżka do naszego interpretera. Dla niektórych języków, możliwe jest także umieszczanie tam parametrów. Skrypty bez tej linii też powinny działać, bo komputery są mądre. W przypadku prostego skryptu Basha zwyczajnie zinterpretuje wewnętrzne komendy jako kolejne linie w terminalu. W przypadku korzystania z np. Pythona tak kolorowo może nie być. W drugiej linii skryptu, lub reszcie umieszczamy kod. Skrypt wywoła program **echo**, który przyjmie jako parametr tekst **„Hello world**
   #!/bin/bash
   echo „Hello world”
3. Sprawdzamy jakie uprawnienia ma plik Z1.sh za pomocą komendy np.:
   mimi@komp:~/Pulpit$ ls -lh
   - Widzimy, że plik Z1.sh ma niepełne uprawnienia (- rw- rw- r--). Widać, że uprawnienia są podzielone # na 3, 3-literowe segmenty, dla każdego należy opisać uprawnienia. Pierwszy segment określa uprawnienia użytkownika głównego. Do nadawania uprawnień wykorzystamy program chmod – **ch**ange **mod**e
     - 7 read, write and execute rwx 111
     - 6 read and write rw- 110
     - 5 read and execute r-x 101
     - 4 read only r-- 100
     - 3 write and execute -wx 011
     - 2 write only -w- 010
     - 1 execute only --x 001
     - 0 none --- 000
   - mimi@komp:~/Pulpit$ chmod 777 Z1.sh
     mimi@komp:~/Pulpit$ ls -lh # sprawdzmy jakie są uprawnienia
   - mimi@komp:~/Pulpit$ chmod 700 Z1.sh
     mimi@komp:~/Pulpit$ ls -lh # sprawdzmy jakie są uprawnienia
4. Uruchamiamy program w tym folderze, w którym się znajdujemy, albo wywołujemy bezpośrednio za pomocą interpretera Bash:
   - mimi@komp:~/Pulpit$ ./Z1.sh # albo <PATH>/Z1.sh
   - mimi@komp:~/Pulpit$ bash Z1.sh # albo bash <PATH>/Z1.sh

#5.

Uruchamiamy program w tym folderze, w którym się znajdujemy
mimi@komp:~/Pulpit$ ./Z1.sh # albo <PATH>/Z1.sh
Albo wywołujemy bezpośrednio za pomocą interpretera Bash
mimi@komp:~/Pulpit$ bash Z1.sh # albo bash <PATH>/Z1.sh

Zadanie 2. Zmienne i parametry

Zmienne jak w każdym innym języku programowania, są ważne ze względu na to, że przechowują informacje które chcemy przekazać komputerowi, albo je od niego wyciągnąć. Przeanalizuj każdy podpunkt (i ewentualnie każdą strzałkę tego podpunktu) przepisując go do skryptu i uruchamiając w terminalu (pamiętaj o #!/bin/bash, każda strzałka to oddzielny działający skrypt):

1. Pomiędzy nazwą zmiennej, a zmienną jest tylko i wyłącznie znak „=". Dodatkowo zmienne nie mogą zaczynać się od liczby. Do zmiennych odwołujemy się za pomocą symbolu $ przed nazwą.
   - Aby operować na zdefiniowanym ciągu znaków wykorzystujemy nawias klamrowy ${str:indeks_poczatkowego_znaku:ilosc_liter}:
     ```
     #!/bin/bash
     zmienna_str="Ala ma kota"
     ilosc=10
     echo $ilosc ${zmienna_str:7:3}ków
     ```
   - Do definiowania stringów można skorzystać z dwóch rodzajów stringów
     ```
     echo "pwd" # wyświetlenie podanego ciągu znaku
     echo `pwd` # odwrotne cudzysłowy ewaluują stringa na linuksowe komendy
     ```

2. Zmienne w Bashu są „untyped", w odróżnieniu od wielu innych języków.
   - Przechowywane są jako ciągi znaków – CHAR STRINGs:
     ```
     int_1=2
     int_2=3
     int_sum_prawie=$int_1+$int_2
     echo $int_sum_prawie
     ```
   - Jak widać na przykładzie powyższym (gdzie Bash potraktował *int_1* i *int_2* jako stringi) i na tym przykładzie, Bash sam odkryje co ma zrobić w zadanym poleceniu, ale trzeba mu trochę pomóc. Na przykład gdy chcemy zrobić operacje arytmetyczne to musimy wykorzystać *arithmetic expansion* – **$((działania))** lub **$[działania]** – czyli bashowej notacji podpowiadającej, że mamy do czynienia z matematyką:
     ```
     int_1=2
     int_2=5
     int_matma_sum_dobrze=$(($int_1+$int_2))
     int_matma_inne_dobrze=$[$int_1+$int_2*$int_2]
     echo $int_matma_sum_dobrze
     echo $int_matma_inne_dobrze
     ```

3. W Bashu nie ma liczb zmiennoprzecinkowych „łatwo" dostępnych ;(Należy odwołać się do *Pythona* (który nie jest domyślnie zainstalowany) albo z kalkulatora *bc* (który jest już domyślnym programem większości Linuxów):
   ```
   float_1=2.72 # brak nawiasów to dla nas znak, że będziemy operować na float
   float_2=3.14 # ale działa jak wpiszemy "3.14"
   # Podziel (albo zrób cokolwiek matematycznego) za pomocą pythona
   string_to_export="print("$float_1"/"$float_2")"
   echo "Python says: "
   echo $string_to_export | python3
   echo
   # Podziel (albo zrób cokolwiek matematycznego) za pomocą bc
   # -l ładuje matematyczną biblioteke i ustawia ilość miejsc po przecinku na 20
   # wykorzystamy `` aby do zmiennej był przypisany ciąg znaków,
   # który w rezultacie przetrzymuje float w postaci innego ciągu znaków
   float_3=`echo $float_1/$float_2 | bc -l`
   echo "bc calculator says: "
   echo $float_3
   ```

```
echo
# Policz wartość funkcji Bessela 2 rzędu z float_3
# za pomocą bc -l
float_bessel=`echo "j(2, $float_3)" | bc -l`
echo "bc calculator says that bessel function of 2 kind of "$float_3" is equal:"
echo $float_bessel
echo
float_sinus_of_bessel_of_float_3=`echo "s($float_bessel)" | bc -l`
echo "bc calculator says that sinus of bessel function of 2 kind of "$float_bessel" is equal:"
echo $float_sinus_of_bessel_of_float_3
echo
```

4. W Bashu (i Linuksie generalnie) mamy także zmienne systemowe, czyli zmienne, które mają do siebie przypisane ważne dane o komputerze, koncie itp.:

```
echo $HOME          # ścieżka do twojego katalogu domowego
echo $USER          # twój login
echo $HOSTNAME      # nazwa twojego hosta
echo $OSTYPE        # rodzaj systemu operacyjnego
echo $UID           # Unique User ID - unikalny numer użytkownika
echo $TERM          # Typ terminala z jakiego korzystamy
```

5. W bashu możemy także zdeklarować tablice/arraye stringów:

```
arr=(1 2 3 "Ala ma kota")
echo ${arr}             # wyświetl 0 element tablicy
echo ${arr[2]}          # wyświetl 2 element tablicy
echo ${arr[@]}          # wyświetl wszystkie elementy tablicy
echo ${arr[3]:7:3}      # wyświetl 3 symbole od 7 znaku trzeciego elementu tablicy
echo ${#arr[@]}         # wyświetl ilość elementów w tablicy
```

6. W bashu możemy deklarować parametry, które przyjmie skrypt, z poziomu terminala:
   - Program skrypt_parametry.sh – parametry zewnętrzne przypisujemy do zmiennych przez odnośniki $1, $2, …, $N, które to kolejno 1, 2, …, N-ty przyjmowany przez skrypt parametr:

     ```
     name=$1
     animal=$2
     echo $name" ma "$animal
     ```

   - Uruchomienie – do skryptu wpadają dwa parametry przypisane $1 i $2

     ```
     mimi@komp:~/Pulpit$ chmod 777 skrypt_parametry.sh
     mimi@komp:~/Pulpit$ ./skrypt_parametry.sh Ala kota. # $1=Ala, $2=kota
     ```

Zadanie 3. Instrukcje i pętla for

Przy instrukcjach warunkowych i funkcjach nie ma wielu różnic pomiędzy Bashem a np., Pythonem – główne to… notacja. Jest dużo sposobów na wykonanie instrukcji warunkowej albo pętli w Bashu, ale tutaj przedstawię dwa najważniejsze podejścia do tematu. Przeanalizuj każdy podpunkt (i ewentualnie każdą strzałkę tego podpunktu) przepisując go do skryptu i uruchamiając w terminalu (pamiętaj o #!/bin/bash, każda strzałka to oddzielny działający skrypt):

1. Warunki if, else, elif – do instrukcji warunkowej należy podać w nawiasie kwadratowym [] wyrażenie logiczne, które zwróci true albo false. Tak też do tego możemy wykorzystać następujące wyrażenia – i tutaj ważne są spacje, które muszą oddzielić wszystkie:
   - -eq # equal to
   - -ne # not equal
   - -gt # greater than
   - -ge # greater or equal
   - -lt # less than
   - -le # less than or equal

```
int_1=$1
int_2=$2

if [ $int_1 -eq $int_2 ]; then
        echo $int_1" jest liczbą równą liczbie "$int_2"."asd
elif [ $int_1 -lt $int_2 ]; then
        echo $int_1" jest liczbą mniejszą i nie równą liczbie "$int_2"."
else
        echo $int_1" i/lub "$int_2" są PRAWDOPODOBNIE jakimiś liczbami."
fi #fin
```

2. Pętla for – Do pętli for mamy dwa podejścia: w stylu Pythona i w stylu C. Obydwa zostaną tu oddzielnie zaprezentowane:
   - Pętla w stylu Pythona – wywołaj to i sprawdź też co zostanie wyświetlone po usunięciu " " w pętli.

```
arr=(1 2 3 "Ala ma kota")
for element in "${arr[@]}"; do
        echo $element
done # fin
```

   - Pętla w stylu C – wywołaj to i sprawdź co zostanie wyświetlone po zamianie < na <= w pętli

```
arr=(1 2 3 "Ala ma kota")
for ((i=0; $i<${#arr[@]}; i++)); do
        echo ${arr[i]}
done # fin
```

Zadanie 4. Funkcje

W Bashu można także definiować własne funkcje, które pomagają zachować przejrzystość kodu, o którą w tym języku ciężko (prywatne zdanie). Przeanalizuj każdy podpunkt (i ewentualnie każdą strzałkę tego podpunktu) przepisując go do skryptu i uruchamiając w terminalu (pamiętaj o #!/bin/bash, każda strzałka to oddzielny działający skrypt):

1. Funkcja bez parametrów

```
# Zdefiniuj funkcję
function co_ala_ma {
  echo "Ala ma kota"
}
# Użyj funkcji
co_ala_ma
```

2. Funkcja z parametrami – w odróżnieniu od innych popularnych języków do funkcji zdefiniowanych w Bashu, parametry przesyłamy nie poprzez argumenty w nawiasach

```
liczba_1=10
liczba_2=20

function razy {
	int_1=$1 # pierwszy argument funkcji
	int_2=$2 # drugi argument funkcji

	echo $(($int_1*$int_2))
}

# wywołaj funkcje z dwoma parametrami
razy $liczba_1 $liczba_2
```

<u>Zadanie końcowe (5p do końcowego wyniku z labów).</u> Jest to 5 punktów trafiających do puli punktów na końcowe rozliczenie – jest tak ze względu na sytuację specjalną, gdyż na dobrą sprawę nie wiemy jak długo i jak często będzie się taka kwarantanna powtarzać, więc już teraz zaczyna się zbieranie punktów. Tak też projekt finalny będzie nie na 15p a na 10p. Każdy ma mieć przygotowany program do sprawdzenia na zajęciach 02.04.2020.

Dany jest plik z wynikami obliczeń Density Functional Theory (DFT) z mechaniki kwantowej ciała stałego 00_15_1101_75.out. Napisz program w Bash, który bazując na tych wynikach/na tym pliku, w automatyczny sposób wyliczy przerwę energetyczną materiału perowskitowego, w każdym kroku iteracji obliczeń pola samouzgodnionego (SCF) – materiału perowskitowego tyczą się te wyniki. Program ma przyjmować nazwę pliku jako argument i wyświetlić każdą wyliczoną przerwę energetyczną wraz z numerem iteracji, w której się wyliczyła oraz to samo zapisać do pliku tekstowego wynik_bg.txt.

Program można wykonać w grupach maksymalnie 3 osobowych – ale zdalnie! bez żadnych kontaktów. Jak ktoś nie może albo nie chce pracować w sposób zdalny to musi zrobić program sam. Na zajęciach 02.04.2020 program zostanie sprawdzony i oceniony. Ważna będzie schludność (nie lubimy natrętnie powtarzającego się kodu w kodzie - redundancji) i wynik końcowy. Samo ocenianie też jest na razie kwestią umowną – nie wiem co będzie za te dwa tygodnie i w jakimś scenariuszu poproszę Was o przesłanie mi gotowych programów i sam je ocenie a w razie wątpliwości mailowo się skonsultujemy.

Wskazówki:

1. Przerwa energetyczna dana jest wzorem
$$B_g = E_{LUMO} - E_{HOMO}$$
gdzie E_{LUMO} i E_{HOMO} są kolejno energią najniższego nieokupowanego orbitalu molekularnego (Lowest unoccupied molecular orbital – LUMO) i energią najwyższego okupowanego orbitala molekularnego (Highest occupied molecular orbit - HOMO)
2. Informacja o obydwu energiach uzyskanej z każdej iteracji znajduje się w linijce zawierającej sentencje „highest occupied, lowest unoccupied level” - za pomocą programu grep można wylistować wszystkie E_{LUMO} i E_{HOMO} . Program grep (w podstawowej wersji, bez parametrów itp.) przyjmuje jako argument sentencje którą poszukujemy w pliku tekstowym i nazwę tego pliku tekstowego a zwraca każdą linijkę w całości (która jest też wynikiem pojedynczej iteracji programu DFT), która posiada ową sentencję. Wpierw spróbuj wyświetlić każdą linie iteracji bez Basha – zwyczajnie tylko z grepa.
3. Po wylistowaniu, reszta to operacja na stringach i trochę matematyki – pamiętaj, że to są liczby zmiennoprzecinkowe.
4. (2 punkty ekstra) Napisz program w Pythonie z wykorzystaniem na przykład numpy i matplotliba do wyświetlania wykresu, tego jak się zmienia przerwa energetyczna z każdą iteracją programu, którego wynikiem jest plik na którym pracujesz. Program ma być wywoływany automatycznie przez Basha z głównym programem. Te dwa punkty nie sprawią, że ktoś kto będzie miał na koniec z laba 15p to otrzyma 17p na koniec – raczej jak komuś się gdzieś noga powinie i będzie miał 14p to te 2 punkty mu dopełnią do 15p :)