תרגול מס׳ 2 : BASH scripts

משתנים ב bash

משתני shell

ניתן להגדיר משתני shell ע״י הצבת ערכים או ע״י שימוש בפקודה declare. ביטול משתנה יעשה ע״י הפקודה unset. אין להוסיף רווח לפני או אחרי הסימן ״=״.ניתן להשתמש בפקודה **declare** כדי להגדיר משתנה בצורה מפורשת. ללא פרמטרים, הפקודה מציגה את כל המשתנים המוגדרים.

```
t2:eesoft>var=”hello”
t2:eesoft>echo $var, world
hello, world
t2:eesoft>declare | grep var
var=hello
t2:eesoft>unset var
t2:eesoft>echo $var, world
, world
```

כדי להפריד שם של משתנה ממחרוזת נשתמש בסוגריים { } :

```
t2:eesoft>echo $var, $variable, ${var}iable
hello, , helloiable
```

משתנים יכולים להחזיק **מערך של ערכים**. התייחסות לאיבר n ע״י X[n].
התייחסות למערך השלם ע״י X[@]. כאשר נציג את X ללא הגדרת תחום, יוצג רק האיבר באינדקס 0 במערך.

```
t2:eesoft>x=( a b c d )
t2:eesoft>echo ${x[@]}  ${#x[@]}
a b c d 4
t2:eesoft>x[7]=e
t2:eesoft>echo $x, ${x[0]}, ${x[7]}, ${#x[@]}
a, a, e ,5
t2:eesoft>echo ${x[@]:0:2}, ${x[@]:1:2}
a b, b c
```

בכל מקום שמופיע התו $, נעשה ניסיון להציב את ערך המשתנה ששמו מופיע צמוד לו. כדי לייצג את התו $ עצמו, נשתמש בתו \ :

```
t2:eesoft>echo \$abc
$abc
```

קיימים ב- bash מספר משתנים שמוגדרים ע״י המערכת. למשל המשתנה **$?** מכיל את ה return value של הפקודה האחרונה שהורצה (בד״כ 0 מורה על הצלחה).

משתני סביבה

משתנים מוגדרים כמשתני סביבה ע"י הפקודה **export** או **declare –x**.
משתנה סביבה חייב להכיל מחרוזת אחת בלבד:

```
t2:eesoft>MyEnv=nothing
t2:eesoft>MyLocal=”never exported”
t2:eesoft>export  MyEnv
t2:eesoft>export  OurEnv="nothing meaningfull"
t2:eesoft>export
declare -x MyEnv=”nothing”
declare -x OurEnv=”nothing meaningfull”
t2:eesoft>echo  my environment variable equals $MyEnv
my environment variable equals nothing
```

הפקודה **export** ללא פרמטרים, או **declare –x**, מציגה למסך את כל משתני הסביבה שקיימים ב shell הנוכחי, בפורמט שיאפשר שחזור קל שלהם.
כל תוכנית שמורצת מתוך- bash מקבלת את רשימת משתני הסביבה שהיו ל shell בזמן ההרצה, וניתן לגשת אליהם ע"י הפקודה getenv() ב-C:

```
/* env,c: an example for using getenv() */
#include<stdlib.h> /* to use getenv( ) */
#include<stdio.h>
voidmain( ) {
	char*	val;
	val = getenv("MyEnv");
	if (val == NULL)
		printf("Env. variable not found\n");
	else
		printf("Env. variable contains %s",val);
}
```

הצבת פלט של פקודה (command substitution)

ניתן להציב למשתנה פלט של פקודה אחרת, דוגמא:

```
t2:eesoft>var=`cat file | wc –l`
t2:eesoft>echo  $var
14
```

שימו לב כי מדובר בגרש שמשמאל למקש 1 במקלדת.

רווחים ב- bash

רווחים ב- bash מפרידים בין פרמטרים שונים. בשביל לציין ערך יחיד שמכיל רווח יש להשתמש במרכאות.

```
t2:eesoft>ls
file 1        file 2
t2:eesoft>cat file 1                #wrong!
cat: file: no such file or directory
cat: 1: no such file or directory
t2:eesoft>cat "file 1"
file 1 contents..
```

מרכאות מאפשרות הצבה של ערך שמכיל רווחים למשתנה, אך המשתנה לא "זוכר" את המרכאות. חובה להקיף את המשתנה במרכאות בכל פעם שמתשמשים בערך שלו. כלל האצבע אומר להשתמש במרכאות תמיד – אלא אם יש סיבה ממש טובה שלא לעשות זאת.

```
t2:eesoft>var="file 1"
t2:eesoft>cat $var                  #wrong!
cat: file: no such file or directory
cat: 1: no such file or directory
t2:eesoft>cat "$var"                #correct
file 1 contents..
```

כאשר מקיפים מערך במרכאות– הדבר שקול להקפת כל אחד מהערכים במרכאות.

```
t2:eesoft>arr[0]="file 1"
t2:eesoft>arr[1]="file 2"
t2:eesoft>cat ${arr[@]}             #wrong!
cat: file: no such file or directory
cat: 1: no such file or directory
cat: file: no such file or directory
cat: 2: no such file or directory
t2:eesoft>cat "${arr[@]}"           #correct
file 1 contents..
file 2 contents..
```

הוספה או החסרה של רווח הם סיבה נפוצה מאוד לבעיות ב- bash.

```
t2:eesoft>var = "file 1"            #wrong!
var: command not found
```

החלפת שמות קבצים

לתוים*,[ ], ?,~ משמעות מיוחדת בשורת פקודה ל shell, והם מוחלפים לרשימה ממוינת של שמות קבצים/מדריכים לפי הדוגמאות הבאות:

t2:eesoft>echo *.c

מראה את רשימת כל הקבצים המסתיימים ב- c.

t2:eesoft>ls a*

מציג את כל הקבצים והתיקיות שמתחילים ב- a (תוכן התיקיות יוצג).

t2:eesoft>echo [a-dfg]xy.[cC]

מציג את שמות כל הקבצים שהאות הראשונה בהם a,b,c,d,f,g, השניה x השלישית y, והסיומת היא c או C.

t2:eesoft>cat fileno?.txt

מציג למסך את תוכן כל הקבצים עם שם שמתחיל בשש האותיות fileno, בנוסף לעוד אות (שלא משנה מה היא) , ומסתיים ב .txt.

t2:eesoft>cd ~/mail

חוזר לתת-מדריך mail של מדריך הבית של המשתמש.

t2:eesoft>ls ~ilana/

מציג את רשימת הקבצים במדריך הבית של המשתמש בשם ilana.

בקרת זרימה ב-Shell

פקודות

if

```
if condition; then
        command list
[elif condition; then
        more command list]
[else
        more command list]
fi
```

while

```
while condition; do
      command list
done
```

for

```
for var in wordlist; do
      command list
done
```

כאשר wordlist הנה מספר מילים המופרדות ע"י רווחים. מילים אילו יכולות להתקבל גם כתוצאה מתבנית. למשל הפקודות הבאות:

```
for file in dir1/abc*; do
      echo $file
done
```

ידפיסו את כל הקבצים בתת-הספרייה dir1 שמתחילים באותיות abc.

case

```
case string  in
      pattern1)
            command list1;;
[     pattern2 | pattern3)          #means pattern1 or pattern 2
            command list2;; ]
```

```
[     *)                    #default
            command list ]
esac
```

תנאים ב bash

ניתן להשתמש כתנאי בכל פקודה חוקית – return value 0 נחשב כ- true וכל ערך מוחזר אחר כ- false.
בנוסף, קיימים תנאים אחרים המוערכים כאשר הם מוקפים ב- [[ ]]. לדוגמא:

-f אם קים קובץ רגיל בשם זה

-nt האם קובץ אחד חדש יותר מאחר

>, < השוואת סדר לקסיקוגרפי בין מחרוזות

==, != התאמת מחרוזת לתבנית

-eq -ne -lt -le -gt -ge השוואת ערך מספרי (**e**qual, **l**ess **t**han, **g**reater **t**han, ..**l**ess or **e**qual)

שילוב ביטויים עפ"י סדר קדימות:

(exp) ערך הביטוי

! exp הערך ההפוך

exp1 && exp2 שני הביטויים מחזירים true

exp1 || exp2 אחד הביטויים מחזיר true

בשני המקרים האחרונים, הביטוי השני לא יחושב אם אין בו צורך לאחר חישוב הערך הראשון.
דוגמאות– שימוש בתבניות:
לבדוק אם מחרוזת מכילה תו נתון:

```
t2:eesoft>if  [[ "a string" == *t* ]]; then echo "has t";fi
has t
```

לבדוק אם מחרוזת מסתיימת באות z או Z:

```
t2:eesoft>if [[ "$end_with_z" == *[zZ] ]]; then echo match; fi
```

לבדוק אם שם הקובץ נגמר בh. או c. , אך לא מתחיל במילה BST:

```
t2:eesoft>if  [[ $filename == *.[CHch] && $filename != BST*) ]]…
```

כמו כן, מומלץ מאוד להשתמש בתבניות בעבודה עם **case**.

תנאים על מערכת הקבצים:

```
t2:eesoft>if [[ ! ( -f $filename) ]]; then echo no file; fi
t2:eesoft>if [[ -d  /tmp ]]; then echo the dir /tmp exists; fi
```

קיימים גם סוגי שאילתות נוספים, כמו -w, -r, -x וכו'.

ערכים נומריים

ביטויים בתוך סוגריים כפולים - (()) מחושבים בצורה נומרית. החישוב מתבצע לערכים שלמים בלבד. הערך שיוחזר מהסוגריים הכפולים הוא אחד אם הביטוי בתוכם היה 0, ואפס אחרת (למה?).

תחביר וקדימות רוב האופרטורים זהה לזו שניתנת להם בשפת C.

פעולות לדוגמא:

!	not
*,/,%,+,-	כפל, חיסור, חיבור, שארית, חלוקה
&&,||	logical or, logical and
>,<,<=,>=	גדול מ-, קטן מ-, קטן-שווה, גדול-שווה.
==,!=	שווה, לא-שווה
=, *=, +=	הצבה, הכפלה והצבה, הגדלה

```
t2:eesoft>t1=5+3
t2:eesoft>(( t2=5+3 ))
t2:eesoft>echo $t1=$t2
5+3=8
t2:eesoft>if [[ 125 > 3 ]]; then echo big; fi
t2:eesoft>  if (( 125 > 3 )); then echo big; fi
big
t2:eesoft>  k=3
t2:eesoft>  while (( $k )); do
>echo $k
>(( k-- ))
> done
3
2
1
```

הערה: **תמיד** יש להשאיר **רווח** בין הסימן)) לבין הביטוי.

scripts

פונקציות

פונקציות ב- bash הן סוג של משתנה. פונקציות יכולות להשתמש במשתנים שהוגדרו מחוץ להן ולשנות אותם.

```
t2:eesoft>hello() {
>echo "hello, $var"
>}
t2:eesoft>var=world
t2:eesoft>  hello
hello, world
```

פרמטרים וערך חזרה

ניתן להעביר **פרמטרים** לפונקציה (ללא סוגריים), וניתן להתייחס לפרמטרים אלה בתוכה:

- $1 – הפרמטר **הראשון**
- $n – **פרמטר מספר n**
- $@ - **רשימת כל הפרמטרים** הנתונים לפקודה
- $#– **מספר הפרמטרים** הניתנים לפקודה

ניתן להחזיר מפונקציה ערך מספרי. כמו בתוכנות אחרות, נהוג להחזיר "0" לציין ריצה תקינה ומספר אחר לציין תקלה (אין חובה להחזיר ערך).

```
t2:eesoft>foo() {
>if (( $#> 0 )); then
>    echo I got $# parameters. The first is $1
>    return 0;
>fi
> return 1
>}
t2:eesoft>if foo a b c d; then echo yes; fi;
I got 4 parameters, the first is a
yes
```

קובץ script

ניתן לאגד קבוצה של פקודות bash לרצף בתוך קובץ אחד בר הרצה, קובץ כזה יקרא script (בדומה לקבצי bat. ב – DOS) :

1. ראשית יש לרשום בשורה הראשונה של הקובץ את שני התוים "#!", ואחריהם (באותה שורה) ה- full-path של bash :

#! /bin/bash

(למתקדמים : שורה זו מורה באיזה אינטרפרטר צריכים להשתמש על מנת לבצע את הקובץ. בצורה דומה ניתן לכתוב קבצי...tcsh, perl, python) .

2. יש למלא בשאר שורות הקובץ את הפקודות שרוצים לבצע.
3. ניתן לרשום הערות בין השורות : שורת **הערה** מתחילה ב- #.
4. כחלק מהסקריפט, ניתן גם להגדיר פונקציות ולהשתמש בהן
5. בסיום כתיבת הקובץ , יש להפוך אותו לקובץ הרצה ע"י chmod :
6.

```
t2:eesoft>nedit print_script &  #creating the file..
t2:eesoft>chmod +x  print_script
t2:eesoft>print_script p1 p2 p3  #executing with parameters
```

הבדלים בין קובץ סקריפט לפונקציה:

1. קובץ סקריפט יכול רק למשתני סביבה של הסביבה שקראה לו, ולא למשתנים רגילים.
2. שינויים שקובץ סקריפט עושה במשתנים לא נשמרים בסיום הריצה שלו
3. קובץ סקריפט תמיד מקבל פרמטר בשם $0– שמכיל את שם הפקודה שהמשתמש קרא לה. הפרמטר הזה לא משפיע על #$ או @$.
4. החזרת ערך מקובץ סקריפט נעשית על-ידי exit ולא על-ידי return

דוגמא: קובץ בשם print_script שתפקידו לשלוח להדפסה את הקבצים שהוא מקבל כפרמטרים:

```
#!/bin/bash
echo hello, $var
for F in $@; do
      echo printing $F
      lpr $F
done
```

```
t2:eesoft>nedit print_script &   #creating the file..
t2:eesoft>chmod +x  print_script
t2:eesoft>var="var value"
t2:eesoft>print_script a.c b.c
hello,                                    #notice - $var is empty!
printing a.c
printing b.c
```

דוגמא ל script

נתון קובץ בנק המכיל נתונים על כל הפעולות של הבנק. כל שורה בקובץ מייצגת פעולה אחת עם שם הלקוח, פעולה (הפקדה + והוצאה –), סניף, יום ושעה:

```
t2:eesoft>cat bank
Dan Shilon      5000      Ramat-Aviv       1/1/1998 10:34
Dudu Topaz      10000     Herzliya-Pituah  1/1/1998 11:22
Haim Yavin      20000     Jerusalem        1/1/1998 15:18
Boris Lavva     –1000     Technion         1/1/1998 17:11
Dan Shilon      3000      Ramat-Aviv       14/1/1998 10:34
Dudu Topaz      11000     Herzliya-Pituah  15/1/1998 13:21
Haim Yavin      18000     Jerusalem        15/1/1998 16:11
Boris Lavva     –2000     Technion         16/1/1998 11:13
```

ברצוננו לכתוב script בשם client, אשר יקבל כפרמטר שם מלא של לקוח והיתרה הנוכחית שלו וידפיס את כל השורות של קובץ bank, בהן מופיע שם הלקוח וגם סה"כ היתרה בחשבונו לאחר ביצוע הפעולות.

למשל, עבור Hayim Yavin נקבל:

```
t2:eesoft>client "Hayim Yavin" 0
Haim Yavin      20000     Jerusalem        1/1/1998    15:18
Haim Yavin      18000     Jerusalem        15/1/1998   16:11
Total balance: 38000
```

שימו לב: רשמנו את השם המלה בתוך **גרשיים " "**, כך שה- script יוכל להתייחס אליו **כמחרוזת אחת** !!

נשתמש ב-2 תוכניות script:

```
t2:eesoft>cat client
#!/bin/bash
if (( $# != 2 )); then
    echo Usage: $0 client_name current_balance
else
    grep"$1" bank | calc_total $2 2
fi
```

גם כאן, רשמנו את $1 בתוך גרשיים, ע"מ ש- grep תחפש את המחרוזת המלאה!

שימו לב: הפרמטר השני של client הוא הראשון של cal_total!

השורה עם ה- grep שופכת את הפלט של ה- grep לתוך script שני שנקרא calc_total. ה- calc_total קורא שורות מהקלט הסטנדרטי שלו ומסכם את השדה השלישי הכל שורה:

```
t2:eesoft>cat calc_total
#!/bin/bash
# reading the first line from the STANDARD INPUT
indx=$2 #number of field to sum in line
(( sum = $1 )) # The initial sum is our first PARAMETER!
while read line; do            #  read returns error on EOF
        line_split=($line)
        echo $line
        (( sum += ${line_split[indx]} +0 ))# +0 helps when line is empty
done
echo"Total balance: $sum"
```

הפונקציה read קוראת שורה מהקלט ומכניסה את הערך שלה למשתנה שניתן לה. לרב, מפצלים את שורה הקלט הנקראת ממחרוזת אחת למערך של מחרוזות, לדוגמא line_split=($line) .

קבצי setup

קבצי setup הם קבצי script מיוחדים אשר מורצים אוטומטית ע"י המערכת בזמנים מסוימים, וכל משתמש יכול להחזיק קבצי setup משלו במדריך הבית שלו (home directory) . קבצים אלה הם:

- **.login** – מבוצע עם כל כניסה של משתמש לחשבון (login-ing)
- **.bashrc** – מבוצע בהתחלת כל bash.

כיוון שקבצים אלה מופעלים בצורה אוטומטית בזמנים המצוינים לעיל, נהוג לאתחל בהם משתני סביבה חשובים, או להגדיר שמות חדשים לפקודות עם **alias** . בדקו, לדוגמא, את הקובץ שהתבקשתם להעתיק מחשבון הקורס.

לקריאה עצמית: שמות חלופיים וקיצורים לפקודות, Alias

הגדרת alias

ניתן להגדיר שם חדש לפקודה ע"י שימוש ב-**alias**:

```
t2:eesoft>  alias  del  rm
```

ואז ניתן להשתמש בפקודה del במקום rm:

```
t2:eesoft>  del file2
```

לקבלת כל ה aliases הנמצאים ב bash הנוכחי:

```
t2:eesoft>  alias
alias del='rm'
alias ll='ls –l'
```

על מנת לדעת מה הפקודה המתאימה ל alias נתון:

```
t2:eesoft>  alias ll
alias ll='ls – l'
```

מחיקת alias

לביטול alias מתוך רשימת ה aliases של ה bash הנוכחי, משתמשים בפקודה **unalias**:

```
t2:eesoft>  unalias del
t2:eesoft>  alias
alias ll='ls – l'
```

עקיפה זמנית של alias

כאשר bash מקבלת פקודה, היא מחפשת אותה בטבלת ה- aliases שלה לפני החיפוש ב- path.
המנגנון הזה מאוד חזק ומאפשר למשתמש לבנות לעצמו רמה של פקודות מעל פקודות ה unix הסטנדרטיות, למשל: הפקודה rm ב- unix מוחקת קבצים בצורה מיידית, ואם מישהו רשם * rm בטעות, אזי הוא מחק את כל המדריך שלו , לא נעים !!
בשביל זה קיימת האופציה **i–** של rm , הגורמת לכך שעל כל קובץ שמנסים למחוק, ה rm תשאל את המשתמש אם הוא באמת רוצה למחוק:
ולכן להרבה משתמשים קיים ה **alias** הבא:

```
t2:eesoft>  alias rm rm –i
t2:eesoft>  rm x.o
rm: remove x.o (y/n)? y
```

וה- rm הנקרא בשורה השניה הוא ה alias rm !!

מאידך, לפעמים אני בטוח במה שאני הולך למחוק, ואני לא צריך את ה- alias , אך גם לא רוצה לבטל אותו לחלוטין, אלא פשוט לעקוף אותו חד-פעמית

```
t2:eesoft>  \rm  *.o
```

הוספת ה "\" לפני הפקודה מורה ל bash **לעקוף** את טבלת ה aliases.

הערה: ל- alias עדיפות גם על שם פונקציה. הוספת "\" לא מבטלת חיפוש פונקציה.

הפקודה type

הפקודה **type** פקודה פנימית של bash שמזהה פקודה נתונה אם היא alias, פקודה פנימית של ה- shell, פונקציה או קובץ בר-הרצה:

```
t2:eesoft>  type  ll
ll is aliased to `ls –l`
t2:eesoft>  type export
export is a shell builtin
t2:eesoft>  type  grep
grep is /bin/grep
```