תוכן עניינים:

פעולות על קבצים: 4
cp 4
rm 4
mv 4
mkdir 4
rmdir 5
cd 5
pwd 5
פקודות לקבלת מידע מקבצים 5
ls 5
cat 6
more 6
head tail 6
wc 7
פקודות לחיפוש/חיתוך ועוד פעולות על תוכן של קבצים: 7
cut 7
echo 9
sort 9
seq 11
tr 11
egrep(וביטויים רגולרים) 12
find 15
Uniq 17
הפניות פלטים קלטים: 19
גלוב סטייל 19
read 20
העברת פרמטרים 20
תנאים: 21
If 22
for 23
while 23
break continue 24

דרכים נפוצות להשתמש בלולאות: 24
case 24
הפעלת תת תהליך(פונקציות וסקריפטים מתוך סקריפט): 25
שימוש בפונקציה בתוך exec של find 31
$x and ${x} 32
ביצוע פעולות על מחרוזות: 33
מערכים: 41
1. Declaring an Array and Assigning values 45
2. Initializing an array during declaration 46
3. Print the Whole Bash Array 46
4. Length of the Bash Array 47
5. Length of the nth Element in an Array 47
6. Extraction by offset and length for an array 48
7. Extraction with offset and length, for a particular element of an array 49
8. Search and Replace in an array elements 50
9. Add an element to an existing Bash Array 50
10. Remove an Element from an Array 51
11. Remove Bash Array Elements using Patterns 52
12. Copying an Array 53
13. Concatenation of two Bash Arrays 53
14. Deleting an Entire Array 54
15. Load Content of a File into an Array 55
SED: 57
אופציה s: 57
אופציה d: 58
אופציה =: 59
אופציה n–: 59
אופציה p: 60
אופציה y: 60
שימוש בפרמטרים: 60
דוגמאות ספיציפיות: 61
AWK: 61
הפעלת awk 61
הלוגיקה של תוכנית awk 62

משתנים ב - awk 63
משתני מערכת 63
תנאים ב - awk 63
הפקודה print 64
printf: 64
מערכים ב AWK: 64
הפונקציה substr 65
הפונקציה sub 65
הפונקציה gsub 66
המשתנה FS 66
הפונקציה split 66
הפונקציה getline: 67
פונקציות שמוגדרות על ידי המשתמש : 67
כתיבה לקובץ 68
שיעור בית: 70
מטלה 6: 70
מטלה 7 74
מטלה 8: 78
מטלה 9: 81
מטלה 10: 86
דוגמאות של אודי מתגבורים: 95
דוגמא מעולה בפתרון לאיך לחתוך את הקלט הפרמטרים לכמה רשימות לפי מילה מפרידה כלשהי: 99

פעולות על קבצים:

cp

פעולה להעתקת קבצים ותיקיות:

```
cp [OPTION]... [-T] SOURCE DEST
cp [OPTION]... SOURCE... DIRECTORY
cp [OPTION]... -t DIRECTORY SOURCE...
```

<u>אופציות</u>

-f – דורס במקרה שהיעד קיים

-i – בודק אם המשתמש לפני שדורס קובץ קיים

-l – יוצר לינק במקום העתק

-r – מעתיק את כל התת תיקיות שנמצאות בתיקיה לתיקיה אחרת

rm

פעולה למחיקת קבצים ותיקיות

```
rm [OPTION]... FILE...
```

<u>אופציות</u>

-f – גורם לכך שלא תיהיה הודעה למסך גם אם קובץ לא קיים

-i – מודיע למסך לפני מחיקה

-r – מוחק את התיקייה על כל תוכנה

כדי להוריד קובץ ששמו מתחיל ב –

```
rm -- -foo
```

mv

שינוי שם או העתקה לתיקית יעד שונה

```
mv [OPTION]... [-T] SOURCE DEST
mv [OPTION]... SOURCE... DIRECTORY
mv [OPTION]... -t DIRECTORY SOURCE...
```

<u>אופציות:</u>

-f - מבצע את הפעולה ללא הודעה למסך

-i – נותן הודעה למסך

-t – מעביר את כל תוכן המקום לתוך תיקיה

mkdir

יוצר תיקיה

```
mkdir [OPTION] DIRECTORY...
```

אופציות

-p – יוצר תיקיות "הורים" במקרה הצורך

rmdir

מוחק תיקיה

```
rmdir [OPTION]... DIRECTORY...
```

אופציות

-p – מוחק את כל העץ שהתבקש

```
Remove DIRECTORY and its ancestors.  E.g., 'rmdir -p  a/b/c'  is
              similar to 'rmdir a/b/c a/b a'.
```

cd

פעולת Terminal משנה את התיקייה בה נמצאים

pwd

מחזיר את "הדרך" המלאה לקובץ.

דוגמא:

kaplshak@mars~>pwd

/home/cs/y13/kaplshak

פקודות לקבלת מידע מקבצים

ls

רשימה – מחזיר רשימה של קבצים.

אופציות

-a – מחזיר את רשימת הקבצים כולל הקבצים המוסתרים

(קובץ מוסתר הוא קובץ שמתחיל ב - .)

-l – נותן רשימה ארוכה כולל כל הפרטים

-r – מדפיס את הרשימה בסדר הפוך

-R - מדפיס פרטים גם עבור תת תיקיות

cat

מדפיס שרשור של קבצים

<u>אופציות:</u>

Default= הדפסת תוכן הקובץ על המסך

-n (number) : מספור השורות בקובץ המוצג

-E (Enter) : $ in the place of every {Enter}

-b: מספור רק על השורות הלא ריקות

-s: מציג שורות רווחים מצומצמות

שימושים נפוצים:

cat file

ידפיס את תוכן הקובץ למסך

cat file1 file2

ישרשר את למסך את תוכן הקבצים

cat >| file

יקלוט מהמקלדת תוכן עד אשר המשתמש ישתמש בשילוב ctr+D

more

בעיקר בשימוש בצד משתמש

more file

more command

הפלט למסך נשלט על ידי המשתמש. כך שקודם יודפס תחילת הקובץ ויוכל לעבור לקריאה של הקטע הבא עד השאר יקיש רווח.

head tail

הדפסת תחילת x שורות מתחילת קובץ או סוף

head –x file

tail –x file

הקוד הבא נותן לפלט את השורות i-j של קובץ:

head -$j $file >| tmp

tail -$[$j - $i + 1] tmp

<u>שימוש באופציה c–:</u>

echo $var|tail –c -n

מדפיס את n התוים האחרונים במשתנה.

```
echo $var|tail –c+-n
```

מדפיס את המחרוזת מהסוף ללא n התוים הראשונים.

```
echo $var|head –c +n
```

מדפיס את n התוים הראשונים .

```
echo $car | head –c –n
```

מדפיס את התוים הראשונים למעט n התוים האחרונים.

חשוב מאד לשים לב שרד שורה נחשב למשתנה.

wc

סופר מילים

```
wc [OPTION]... [FILE]...
```

<u>אופציות</u>

-m – סופר את כמות התווים בקובץ

-c – באופן מוזר עושה את אותו הדבר כמו האופציה מעליו

-l – סופר את כמות השורות

-L – מחזיר את אורך השורה הארוכה ביותר

-w – סופר את כמות המילים בקובץ

<u>הערה חשובה</u> – יש לשים לב \n נספר כתו. גם בספירת תווים וגם בספירת אורך השורה הארוכה בקובץ.

פקודות לחיפוש/חיתוך ועוד פעולות על תוכן של קבצים:

cut

```
cut [OPTION]... [FILE]...
```

<u>אופציות:</u>

-c – חותך תוים מסויים מקלט

-d – מחליט ע"פ איזה ביטוי לחתוך ביטוי

-f- כדי לבחור כמה שדות להדפיס

--output-delimiter=”string” – בין המילים בפלט תופיע המחרוזת

<u>דוגמאות:</u>

kaplshak@mars~/lec4/ex4>cat F2

10 3 6 8 5

```
1 2 3 4 5 6 7 8

2 4 6 8 10

1 2 3 3 2 1

3 4 7 100

kaplshak@mars~/lec4/ex4>cut -c1-2 F2

10

1

2

1

3

kaplshak@mars~/lec4/ex4>cut -d"2" -f1 F2

10 3 6 8 5

1


1

3 4 7 100
```

```
kaplshak@mars~/lec4/ex4>cut -d" " -f4 F2
8
4
8
3
100
```

echo

הדפסה לפלט של מלל או משתנה

```
echo $varible
```

<u>אופציות</u>

-n – לא יורד שורה

sort

מקבל קלט ומוציא אותו לפלט לפי סידור מסויים:

```
cat filename | sort
```

<u>אופציות:</u>

-u – same as sort|uniq

-z – מסיים שורה ב0 במקום שורה חדשה

-n – סידור נומרי

-f – התעלמות מאותיות גדולות התייחסות כקטנות

-d – סידור מילוני

-b – התעלמות מרווחים מובילים

-k – סידור לפי עמודה מסויימת ניתן ליצור תחום ע"י פסיק בין שני מסרים

<u>Sort לפי שדות</u>

ניתן למיין לפי חלקים של השורה כפי שמראות הדוגמאות הבאות:

Sort –k 3,3 ממיין בסדר א"ב לפי המילה השלישית בשורה.

Sort –k 3n,3 ממיין בסדר מספרי לפי המילה השלישית בשורה.

Sort –k 3,3n ממיין בסדר מספרי לפי המילה השלישית בשורה.

Sort –k 3n, 3n ממיין בסדר מספרי לפי המילה השלישית בשורה.

Sort –k 3nr,3 ממיין לפי המילה השלישית בסדר מספרי המיון בסדר יורד.

Sort –k 3n,3 –k 5,5 –k 1n,1 ממיין לפי המילה השלישית בסדר מספרי עבור השורות שזהות במילה השלישית מיין אותן לפי המילה החמישית בסדר א"ב עבור השורות שזהות במילה השלישית והחמישית מיין לפי המילה הראשונה בסדר מספרי.

<u>הערה:</u> לשים לב אפשר ולעשות מיון של 2 מסמכים לתוך פלט אחד.

<u>דוגמאות</u>

basicsys@mars~/lec6>cat F1

Jim Alchin 21 Seattle

Bill Gates 8 Seattle

Steve Jobs 246 Nevada

Scott Neally 2122 Los Angeles

basicsys@mars~/lec6>sort F1

Bill Gates 8 Seattle

Jim Alchin 21 Seattle

Scott Neally 2122 Los Angeles

Steve Jobs 246 Nevada

basicsys@mars~/lec6>sort -k 3,3 F1

Jim Alchin 21 Seattle

Scott Neally 2122 Los Angeles

Steve Jobs 246 Nevada

Bill Gates 8 Seattle

ניתן להוסיף את האות n למספר 1 או למספר (2 או לשניהם) והמשמעות היא

שהמיון לפי התחום יהיה בסדר מספרי .

basicsys@mars~/lec6>sort -k 3n,3 F1

Bill Gates 8 Seattle

Jim Alchin 21 Seattle

Steve Jobs 246 Nevada

Scott Neally 2122 Los Angeles

basicsys@mars~/lec6>sort -k 3,3n F1

Bill Gates 8 Seattle

Jim Alchin 21 Seattle

Steve Jobs 246 Nevada

Scott Neally 2122 Los Angeles

סידור קובץ לפי הfield הראשון בקובץ:

$ sort -t"," -k1,1 file

סידור קובץ לפי ה field השני בקובץ:

$ sort -t"," -k2,2 file

Remove duplicates from the file based on 1st field:

$ sort -t"," -k1,1 -u file

Sort the file numerically on the 2nd field in reverse order:

$ sort -t"," -k2nr,2 file

sort the file alphabetically on the 1st field, numerically on the 2nd field:

$ sort -t"," -k1,1 -k2n,2 file

seq

מוציא לפלט סדרה של מספרים

seq [OPTION]... LAST
seq [OPTION]... FIRST LAST
seq [OPTION]... FIRST INCREMENT LAST

<u>אופציות</u>

-f – פורמט לפי הכללים של פרינט אף

-s, --separator=STRING – default = /n

-w, --equal-width

equalize width by padding with leading zeroes

tr

תרגם החלף.

```
tr [OPTION]... SET1 [SET2]
```

<u>אופציות</u>

-d – מוחק

-s – צמצם

צמצום רווחים:

cat file | tr –s " "

lower case to upper case:

tr a-z A-Z

egrep(וביטויים רגולרים)

חיפוש חכם

<u>אופציות:</u>

-c - הצג את מספר השורות שהתאימו לביטוי

-h - אל תוסיף את שמות הקבצים בהתחלת השורות שמוצגות בפלט

-l - הצג את שמות הקבצים שיש להם שורה אחת לפחות שמתאימה לביטוי .

(כל שם קובץ מוצג פעם אחת בלבד, גם אם יש לו כמה שורות שמתאימות

לביטוי).

-i - אל תבדיל בין אותיות אנגליות קטנות לגדולות .

-w - הצג את השורות שיש בהן מילה שמתאימה לביטוי. (כאשר מילה מוגדרת

כאוסף של תווים שמורכבים מאותיות אנגליות וספרות).

-o - הצג את החלקים של השורה שמתאימים לביטוי .

-v - הצג את השורות שלא מתאימות לביטוי

-L – קבצים שלא נמצאה התאמה עבורם

-n – מחזיר את מספר השורה עבורה נמצאה התאמה

-r – חיפוש בכל התקיות והקבצים שמתחת לתיקיה

<u>שימוש ב backtracking - בביטוי רגולארי :</u>

\1מתאים לחלק של המחרוזת שהתאים לסוגריים הראשונים בביטוי

\2מתאים לחלק של המחרוזת שהתאים לסוגריים השניים בביטוי

\iמתאים לחלק של המחרוזת שהתאים לסוגריים מספר i בביטוי

מספור הסוגריים הוא משמאל לימין, הפותח סוגר הראשון והסוגר שמתאים

לו הוא סוגריים מספר 1 הפותר סוגר השני והסוגר שמתאים לו הוא סוגריים

מספר 2 וכן הלאה...

<u>החוקים של ביטויים רגולרים:</u>

החוקים של ביטוים רגולאריים (Extended Regular Expression Style)

תו בודד כלשהו מתאים לעצמו. לדוגמה a מתאים ל - a .

קבוצת תווים בתוך סוגריים מרובעים מתאימה לתו בודד מתוך הקבוצה .

a,b,c,6,7,8 . מהתווים אחד ל מתאימה [a-c6-8] לדוגמה

קבוצת תווים בתוך סוגריים מרובעים שמתחילים בסימן ^ מתאימה לתו בודד

שאינו בתוך הקבוצה. לדוגמה [^a-c] מתאימה לתו בודד שאינו a b או c.

(. נקודה) מתאימה לתו בודד כלשהו (אבל לא למחרוזת ריקה).

(^ שאינו בתוך סוגריים מרובעים) מציין התאמה לתחילת המחרוזת .

$ מציין התאמה לסוף המחרוזת .

חשוב לזכור: אם לא מופיעים ^ ו $ - בביטוי הרגולארי אז מספיקה התאמה לחלק (רציף) של המחרוזת הנבדקת. לדוגמה: הביטוי a מתאים למחרוזת bab אבל הביטוי $a^ לא מתאים למחרוזת bab (אלא לתו a בלבד)

* מציין 0 או יותר חזרות של מה שלפני ה - *

+ מציין 1 או יותר חזרות של מה שלפני ה + -

? מציין 0 או 1 חזרות של מה שלפני ה ? -

{n} מציין בדיוק n חזרות של מה שלפני ה - {n}

{n,m} מציין מספר חזרות גדול שווה n וקטן שווה m של מה שלפני ה - {n,m}

{n,} מציין מספר חזרות קטן שווה n (כולל 0 חזרות) של מה שלפני ה - {n,}

{,n} מציין מספר חזרות גדול שווה n של מה שלפני ה - {,n}

בכל הכללים הנ "ל מה שלפני יכול להיות :

- תו בודד כלשהו. לדוגמה בביטוי ab*c מה שלפני ה - * הוא b .

ולכן הכוונה ל - a אחד לאחר מכן 0 או יותר חזרות של b ולבסוף c אחד .

- קבוצת תווים בתוך סוגריים מרובעים (עם או בלי ^ בהתחלה). לדוגמה בביטוי a[0-9]+c מה שלפני ה - + הוא [0-9] . ולכן הכוונה ל - a אחד לאחר מכן 1 או יותר חזרות של ספרות ולבסוף c אחד .

• ביטוי רגולארי בתוך סוגריים עגולים. לדוגמה בביטוי a(bc){2}d

הכוונה ל - a אחד לאחר מכן פעמים הרצף bc ולבסוף d אחד .

| מציין or לוגי בין ביטויים לדוגמה cd | $^ab יתאים למחרוזת ab או

למחרוזת שמכילה את הרצף cd באיזשהו מקום .

>\ מציין גבול מילה מצד שמאל <\ מציין גבול מילה מצד ימין

<\word>\

הערה לשים לב: grep –e שקול ל egrep

התאמת מחרוזות לפי ביטויים רגולאריים מורחבים			
דוגמא לחוק	ביטוי	מחרוזת	התאמות
1	a	a	מתאים
	a	A	לא מתאים
2	[a-c]	b	מתאים
	[a-c]	d	לא מתאים
3	[^a-c]	d	מתאים
	[^a-c]	b	לא מתאים
5	a	bac	מתאים
	ab	abab	מתאים
	a	Abc	לא מתאים
	ab	acb	לא מתאים
6	^ab	xab	לא מתאים
	^ab	abxyz	מתאים
7	ab$	abx	לא מתאים
	ab$	xab	מתאים
8	^ab$	xab	לא מתאים
	^ab$	abx	לא מתאים
	^ab$	מתאים רק ל- ab	
	^[a-c][a-c]$	מתאים ל- aa, ac, ,ab bb, ba, bc, cc, ca, cd	

<u>דוגמאות:</u>

סופר את מספר ההופעות של תו בקלט

```
echo -n aabbaba |egrep -o a | wc –l
4
basicsys@mars~/lec7>egrep -i AB F1 F2
F1:abcdef
F1:12334ab
F1:abcd
basicsys@mars~/lec7>egrep -i -l AB F1 F2
F1
basicsys@mars~/lec7>egrep -i -l "AB|1" F1 F2
F1
F2


basicsys@mars~/lec7>cat F1
adogb
acat
acow
catb
```

כל השורות שלא התאימו

```
basicsys@mars~/lec7>egrep -v "cat" F1
adogb
acow
```

find

find <local or sub directory> <search criteria> [<action to be taken>]

<u>מבנה הפקודה :</u>

פקודות לביצוע תנאים תיקיה/תיקיות find

הפקודה עוברת על כל הקבצים בתיקיות שהועברו לה כפרמטרים .

עבור כל קובץ בודקת את כל התנאים (לפי הסדר משמאל לימין) אם כל

התנאים מתקיימים אז מבוצעות הפקודות לביצוע. התנאים והפקודות לביצוע

הם בפורמט מיוחד לפקודה find (שלא עובד בפקודות אחרות).

ניתן גם לערב בין התנאים והפקודות לביצוע, ואז כשמגיעים לפקודה לביצוע

היא מבוצעת אם התקיימו כל התנאים שמשמאל לפקודה

<u>התנאים של הפקודה find</u>

בכל התנאים שבודקים ערך מספרי מתקיים הכלל הבא :

n מציין שווה בדיוק ל - n

n+ מציין גדול מ - n

n- מציין קטן מ - n

size- גודל הקובץ לדוגמה size 4c- הקובץ מכיל 4 תווים בדיוק.

ביטוי name- שם הקובץ מתאים לביטוי (ההתאמה בסגנון glob).

type- סוג הקובץ , האפשרויות: d - תיקיה, f - קובץ

mmin- מתייחס לזמן (בדקות) של העדכון האחרון של הקובץ .

לדוגמה mmin -100- אומר שהקובץ עודכן לאחרונה לפני פחות

מ - 100 דקות .

<u>הפקודות לביצוע של הפקודה find</u>

נדגיש שוב שהפקודות לביצוע הן רק עבור הפקודה find והן מבוצעות

עבור כל קובץ שה - find מגיע אליו בתנאי שהקובץ מקיים את כל התנאים

שמופיעים לפני הפקודה לביצוע .

print – מדפיס את שם הקובץ לפיו הייתה התאמה

print0 – מדפיס את שם הקובץ ורד שורה לאחריו

-exec command {} \; - מבצע את הפקודה כאשר הסוגריים המסוסלסלים מוחלפים בשם הקובץ

-exec command {} + - הסוגריים המסולסלים מוחלפים ברשימת כל הקבצים שמתאימים לחיפוש

הערה: יש לשים לב שהפקודה find מתבצעת בתיקיה שבא הריצו את הסקריפט ולא בתיקיה שבא נמצאה התאמה.

דוגמאות:

נותן לפלט את שמות הקבצים שנמצאה שמכילים ביטוי מסויים

```
find $1 -type f -exec egrep -l $2 {} \;
```

```
basicsys@mars~/lec8>tree t1
```

```
t1
|-- d1
| |-- d3
| | |-- a1
| | |-- f2
| | `-- f3
| |-- f3
| `-- f4
|-- d2
| `-- d3
| `-- f5
|-- f1
`-- f2
```

```
basicsys@mars~/lec8>cat remove_duplicates
find $1 -type f -exec get_file_name {} \; >|tmp
sort tmp | uniq -d >| tmp1
for x in $(cat tmp1)
do
 find $1 -name "$x" -type f -exec rm -i {} \;
done
```

```
basicsys@mars~/lec8>remove_duplicates t1
rm: remove regular file `t1/f2'? n
rm: remove regular file `t1/d1/d3/f2'? n
rm: remove regular file `t1/d1/f3'? n
rm: remove regular file `t1/d1/d3/f3'? n
```

Uniq

uniq [-duc] <filename>

<u>אופציות</u>

Default= הצגת הקובץ ללא שורות כפולות

-d (duplicate) : מציגה **<u>רק את</u>** השורות שיש להן כפילויות

-u (uniq) : מציגה רק רת השורות שאין להן כפילויות

-c (counter) : לפני הצגת כל שורה, מציגה את מס' הכפילויות שלה

הערה: יש להשתמש בפקודה sort לפני השימוש ב uniq

<u>דוגמאות:</u>

הצגת מספור שורות כפולות:

```
sort namesd.txt | uniq –c

      2 Alex Jason:200:Sales

      2 Emma Thomas:100:Marketing

      1 Madison Randy:300:Product Development

      1 Nisha Singh:500:Sales

      1 Sanjay Gupta:400:Support
```

הצגת רק שורות כפולות:

```
$ sort namesd.txt | uniq –cd

      2 Alex Jason:200:Sales

      2 Emma Thomas:100:Marketing
```

הפניות פלטים קלטים:

סימונים -

0	קלט סטנדרטי	Stdin	מקלדת
1	פלט סטנדרטי	Stdout	מסך
2	שגיאות סטנדרטי	stdert	מסך

הפניות פלט-

פלט ז"א abc - "echo "acb יופיע במסך.

הודעת שגיאה "echo1 "abc יופיע במסך.

שם הקובץ <1 פקודה –

הפלט של הפקודה (לא הפלט של השגיאה, דהיינו הפלט הרגיל) הולך לקובץ במקום למסך.

- אם הקובץ <u>לא קיים</u>, יוצר קובץ בשם שהופיע מימין ל < והתוכן שלו יהיה הפלט של הפקודה
- אם הקובץ <u>קיים</u> יש 2 אפשרויות:

1. (set –c) אם האופציה no-clobber אל תדרוס בתוקף מישהו ביצע את הפקודה set –c שגורם לאופציה להיות בתוקף אז נקבל הודעת שגיאה
2. אם האופציה no-clobber אינה בתוקף, ז"א מישהו ביצע את הפקודה set +c לפניכן ואז תוכן הקובץ ידרוס בפלט של הפקודה.

שם הקובץ <2 פקודה –

כל מה שכתבנו עבור <1 מתקיים, פרט להבדל שהפלט שנכתב לקובץ (במידה ומותר לדרוס אותו) הוא לא הפלט הרגיל של הפקודה אלא של השגיאות. אם הפקודה התבצעה בסדר ללא שגיאות אז פלט השגיאות ריק.

שם הקובץ < פקודה –

שקול ל- שם קובץ <1 פקודה

- <u>שם הקובץ |<1 פקודה</u>
במקרה זה התוכן ל הקובץ הופך להיות הפלט הרגיל של הפקודה. אם הקובץ <u>קיים</u> אז הוא נדרס (לא תלוי באופציה ho-clobber), אם הקובץ <u>לא קיים</u> אז הוא נוצר.
- <u>שם הקובץ |<2 פקודה</u>
כמו |<1 אבל הפלט הוא פלט שגיאות ולא פלט רגיל
- על מנת שהפלט הרגי וגם הפלט שגיאות ילכו לאותו קובץ למשל F1 משתמשם בצורה הבאה

פקודה 1>F1 2>&1

1 ⟶ F1

2 ⟶ 1

הפניית פלט רגיל לסוף הקובץ ז"א משרשרת ולא דורסת-

שם קובץ << פקודה

אם הקובץ לא קיים – הוא נוצר.

אם הקובץ קיים הפלט של הפקודה מתווסף לסוף הקובץ (ללא קשר לאופציה ho-clobber)

גלוב סטייל

תו בודד כלשהו לעצמו. לדוגמא a מתאים ל a.

קבוצת תווים בתוך סוגריים מרובעים מתאימה לתו בודד מתוך הקבוצה.

דוגמא: [a-c6-8] מתאימה ל אחד מהתווים a,b,c,6,7,8 .

? – התאמה לתו בודד כלשהו (אבל לא למחרוזת ריקה)

* מתאימה לרצף תווים כלשהו (כלל מחרוזת ריקה)

הערה: בשיטה גלוב סטייל התאמה חייבת להיות התאמה מלאה ולא חלקית לדוגמא

a מתאים ל a אבל a לא מתאים ל ab.

read

שם משתנה read.
למשל: read X
המשתמש מקליד מחרוזת מהפלט הסטנדרטי (כרגע המקלדת) עד שמקליד enter ואז המחרוזת שהמשתמש הקליד (לא כולל התו enter) הופכת להיות התוכן של המשתנה X ז"א שנבצע echo $X נראה את המחרוזת.

העברת פרמטרים

שיטת העברת הנתונים(תקף לסקירפט פונקציה וכו')

העברת פרמטרים לתוכנית סקריפט

$1 - הפרמטר הראשון $2 - הפרמטר השני וכן הלאה ... עד $9

$# - מספר הפרמטרים לתוכנית

$* - רשימה של כל הפרמטרים (בהנחה שאין תווי רווח בפרמטרים)

$@ - רשימה של כל הפרמטרים (בהנחה שאין תווי רווח בפרמטרים)

"$*" - רשימה בעלת איבר אחד שמכילה את כל הפרמטרים כולל תווי הרווח

עם רווח אחד בין הפרמטרים .

"$@" - רשימה שמכילה את כל הפרמטרים כולל תווי הרווח מספר האיברים

ברשימה הוא בדיוק כמספר הפרמטרים .

הפקודה shift

מבצעת הזזה שמאלה של הפרמטרים לתוכנית (כאשר הפרמטר הראשון

נמחק). לדוגמה, אם הפרמטרים לתוכנית הם 10 20 30 40 אחרי ביצוע

הפקודה shift הפרמטרים לתוכנית הופכים להיות 20 30 40

דוגמאות:

```
basicsys@mars~/lec7>cat P2
echo $*
echo $#
shift
```

```
echo $*
echo $#
shift
echo $*
echo $#
```

```
basicsys@mars~/lec7>P2 10 20 30 40
10 20 30 40
4
20 30 40
3
30 40
2
```

תנאים:

משפט תנאי -

לצורך משפטי תנאי קיימת פקודה test שהמטרה שלה לאפשר התניית תנאים.

צורה 1: תנאי test

צורה 2: [תנאי] - חייב להיות רווח אחד לפחות מ2 צדדי הסוגריים.

סוגי תנאים –

<u>תנאים על קבצים:</u>

-s	file exists and is not empty
-f	file exists and is not a directory
-d	directory exists
-x	file is executable
-w	file is writable
-r	file is readable

<u>תנאים על מספרים:</u>

שווה	[3 –eq 3]	התנאי true
לא שווה	[3 –ne 3]	התנאי false
גדול מ	[5 –gt 3]	התנאי true
קטן מ	[5 –lt 3]	התנאי false
גדול או שווה	[5 –ge 5]	התנאי true
קטן או שווה	[5 –le 5]	התנאי true

התנאים על מחרוזות:

שווה	[abc = abd]	התנאי false
שווה	[abc = abc]	התנאי true
שונה	[abc |= abc]	התנאי false

If

if [תנאי] **Then (שורה נפרדת)** **שורות שמכילות פקודות של bash** **fi (שורה נפרדת)**	**אפשר להוסיף else ל- if** **if [תנאי]** **Then (שורה נפרדת)** **סדרת פקודות bash** **else (שורה נפרדת)**

- אם התנאי מתקיים ממשיכים למטה ומבצעים את הפקודות.

דוגמאות:

```
# Prompt for a user name...
echo "Please enter your name:"
read USERNAME

# Check for the file.
if [ -s ${USERNAME}_DAT ]; then
      # Read the age from the file.
      AGE=`cat ${USERNAME}_DAT`
      echo "You are $AGE years old!"
else
      # Ask the user for his/her age
      echo "How old are you?"
      read AGE

          if [ "$AGE" -le 2 ]; then
                    echo "You are too young!"
          else
                    if [ "$AGE" -ge 100 ]; then
                              echo "You are too old!"
                    else
                    # Write the age to a new file.
                    echo $AGE > ${USERNAME}_DAT
          fi
      fi
fi
```

for

לולאת for -

```
for x in aa 10 bb
do
echo $x $X
```

```
for שם משתנה in רשימה
do
```

המשתנה מקבל בהתחלה כערך את האיבר הראשון ברשימה ואז מבצע את הפקודות בתוך הלולאה אחר כך חוזרים לתחילת הלולאה המשתנה מקבל ערך, את האיבר השני ברשימה וכך הלאה עד שהרשימה מסתיימת וממשיכים לבצע את הפקודות שאחרי ה- done.

הרשימה שנוצרה לא מתאימה לשורות בקובץ אלא למילים בקובץ בדוגמא הרשימה תכיל 6 אברים והפלט יהיה:
aa bb cc d def gf

F1
aa bb cc
dd

ולכן

```
for x in $(cat F1)
do
echo $x
```

while

משפט while –

- אם התנאי מתקיים מבצעים את סדרת הפקודות בין ה do ל done בסיום הביצוע בודקים שוב אם התנאי מתקיים אם כן מבצעים שוב את סדרת הפקודות בין ה do ל done וכך הלאה עד שהתנאי לא מתקיים ואז ממשיכים לבצע את הפקודות שאחרי ה done.

```
while [ תנאי ]
do (שורה נפרדת)
סדרת פקודות
done (שורה נפרדת)
```

הפקודה read x –

כאשר משתמשים בה בתור תנאי, לדוגמא: if [read x]
אם משתמשים בה מהמקלדת אז כאשר משתמש מקליד מחרוזת כלשהיא התנאי true, אבל כאשר המשתמש מקליד ctrl +d אז התנאי false.

```
While [ read x ]
do
    הלולאה תמשיך עד המשתמש יקליד ctrl +d.
```

במבנה הנ"ל הוא יקרה את השורות מקובץ F1 כל פעם שיקרא שורה התנאי read x
יהיה true כאר יגיע לסוף הקובץ וינסה לקורא את שורת התנאי read x יהיה false ואז יצא מהלולאה.

```
While [ read x ]
do
```

F1
10
20

```
Y=0
While [ read x ]
do
Y=$[$y+$x]
```

break continue

break – גורם לסגירה של הלולאה כולה

countinue – מסיים את האיטרציה הנוכחית של הלולאה וממשיך להבאה

דרכים נפוצות להשתמש בלולאות:

קליטת כל שורה של קובץ לתוך משתנה

```
while read x

do

done<filename
```

הכנסת כל מילה בקובץ לתוך משתנה

```
for i in $(cat filename)

do

done
```

הכנסת כל מילה במשתנה לתוך משתנה

```
for i in $(echo $variable)

do

done
```

case

case משנה in
(ביטוי בסגנון glob)
סדרת פקודות
;;
(ביטוי בסגנון glob)
סדרת פקודות
;;
esac

מתבצעת בדיקה האם ערך המשתנה מתאים לביטוי 1 לפי חוקים glob
אם יש התאמה מבצעים את סדרת הפקודות עד ;; וממשיכים אחרי ה- esac.
אם אין התאמה בודקים האם ערך המשתנה מתאים לביטוי 2 לפי חוקי glob וכן האלה..

מיוחד למשפט case הוסיפו אפשרות להשתמש ב | כ- or לוגי ז"א ביטוי 2 | ביטוי 1 הכוונה שיש התאמה לביטוי 1 או לביטוי 2.

דוגמא:

```
case "$1" in
 [a-c]r[a-c]) echo "you typed one of a,b,c"
 echo "followed by r followed"
 echo " bye one of a,b,c.";;
 *aef[wqs]?)
 echo "you typed any string followed by aef"
 echo "followed by one of w,q,s followed"
 echo "by one character.";;
 ?)
 echo "you typed exactly one character";;
 ?*)
 echo "you typed at least one character";;
 *)
 echo "you typed nothing of the first options";;
esac
```

הפעלת תת תהליך(פונקציות וסקריפטים מתוך סקריפט):

הגדרת פונקציה :

} שם הפונקציה

function

גוף הפונקציה

{

צורה נוספת להגדרת פונקציה :

} () שם הפונקציה

גוף הפונקציה

{

אפשר לקרוא לפונקציה רק לאחר שהפונקציה הוגדרה(כלומר רק קוד שנכתב מתחת לפונקציה יכול לקרוא).

<u>המשתנים של הפונקציה</u>

המשתנים של הפונקציה שלא הוגדרו על ידי הפקודה local (שתתואר

בהמשך) מוכרים גם בסביבה החיצונית, והפונקציה יכולה לשנות אותם

```
basicsys@mars~/lec12>cat P1
x=8
function f1 {
 echo "inside f1 x=$x"
 x=9
 y=10
}
echo "outside before calling f1 x=$x y=$y"
f1
echo "outside after calling f1 x=$x y=$y"

basicsys@mars~/lec12>P1
outside before calling f1 x=8 y=
inside f1 x=8
outside after calling f1 x=9 y=10
```

<u>הפקודה local</u>

הפקודה: שם משתנה local

מגדירה משתנה מקומי שמוכר רק בתוך הפונקציה, ונעלם ביציאה מהפונקציה .

```
basicsys@mars~/lec12>cat P1
x=8
function f1 {
 echo "inside f1 x=$x"
 local x
 local y=10
 echo "inside f1 after local statement x=$x y=$y
```

```
 x=9
}
echo "outside before calling f1 x=$x y=$y"
f1
echo "outside after calling f1 x=$x y=$y"

basicsys@mars~/lec12>P1
outside before calling f1 x=8 y=
inside f1 x=8
inside f1 after local statement x= y=10
outside after calling f1 x=8 y=
```

<u>העברת פרמטרים לפונקציה(מפורט בקטע אחר במדריך)</u>

העברת הפרמטרים לפונקציה מתבצעת על ידי המשתנים

...,$1,$2,$*,$@,$#$ באופן דומה להעברת פרמטרים לסקריפט. בתוך

הפונקציה אין גישה לפרמטרים של הסקריפט. לדוגמה , כאשר רושמים בתוך

הפונקציה $1 הוא יוחלף בפרמטר הראשון בקריאה לפונקציה (ולא בפרמטר

הראשון בקריאה לסקריפט שבתוכו מוגדרת הפונקציה).

```
basicsys@mars~/lec12>cat P1
echo $1 $2
f1() {
 echo $1 $2
 for x in "$@"
 do
 echo "$x"
 done
}
f1
f1 "20 30" 40
```

```
echo $1 $2


basicsys@mars~/lec12>P1 400 5
400 5


20 30 40
20 30
40
400 5
```

<u>השפעת הפעלת פונקציה על ידי קריאה מהסוג (שם הפונקציה $)</u>

כאשר קוראים לפונקציה בצורה הבאה (: שם הפונקציה $) נפתח תת תהליך שבו מופעלת הפונקציה הקריאה (שם הפונקציה $) תוחלף על ידי הסורק בפלט של תת התהליך הזה. המשתנים של תת התהליך הזה נעלמים כאשר תת התהליך הזה מסתיים .

```
basicsys@mars~/lec12>cat P1
f1() {
 echo $[$1**2 + $2**2]
 y=$[$1**2 + $2**2]
}
f1 2 3
echo y=$y
x=$(f1 3 4)
echo x=$x
echo y=$y
echo "$(f1 6 7)"
echo y=$y
f1 8 9
echo y=$y
```

```
basicsys@mars~/lec12>P1
13
y=13
x=25
y=13
85
y=13
145
y=145
 5


basicsys@mars~/lec12>cat P1
f1() {
 echo $[$1**2 + $2**2]
 y=$[$1**2 + $2**2]
 f2
}
f2(){
 echo y inside f2=$y
}
f1 2 3
echo y=$y
x=$(f1 3 4)
echo x=$x
echo y=$y
echo "$(f1 6 7)"
echo y=$y
```

```
f1 8 9
echo y=$y

basicsys@mars~/lec12>P1
13
y inside f2=13
y=13
x=25 y inside f2=25
y=13
85
y inside f2=85
y=13
145
y inside f2=145
y=145
```

<u>סקריפט ופונקציה בעלי שם זהה</u>

כאשר יש קובץ סקריפט ופונקציה בעלי שם זהה תיקרא הפונקציה (ולא הסקריפט .) כדי לקרוא לסקריפט ולא לפונקציה יש להוסיף /. לפני הקריאה

```
basicsys@mars~/lec12>cat f1
echo "I am file f1"

basicsys@mars~/lec12>cat P1
f1() {
 echo I am f1 in P1
}
f1
./f1
f1
```

```
basicsys@mars~/lec12>P1
I am f1 in P1
I am file f1
I am f1 in P1
```

שימוש בפונקציה בתוך exec של find

כדי לקרוא לפונקציה בתוך exec יש להצהיר עליה כסוג export ויש להוסיף

bash -c מיד לאחר ה - exec כפי שמתואר בדוגמה הבאה

```
basicsys@mars~/lec12>tree
~basicsys/win14/ex7/d2/d2
/home/cs/segel/basicsys/win14/ex7/d2/d2
|-- A
`-- d3
 |-- E
 `-- d4
 `-- S
2 directories, 3 files

basicsys@mars~/lec9>cat P1
get_file_name(){
 echo $1 | tr "/" "\n" | tail -1
}
export -f get_file_name
find $1 -type f -exec bash -c "get_file_name {}" \;

basicsys@mars~/lec12>P1 ~basicsys/win14/ex7/d2/d2
A
E
```

S

$x and ${x}

הצורה שם משתנה $ היא קיצור ל { - שם משתנה .}$ כאשר יש שתי אפשרויות

שונות לפרוש שם משתנה $ מומלץ להשתמש ב { - שם משתנה .}$

```
basicsys@mars~/lec12>x=abcde

basicsys@mars~/lec12>echo $x
abcde

basicsys@mars~/lec12>echo ${x}
abcde

basicsys@mars~/lec12>xx=5

basicsys@mars~/lec12>echo ${x}x
abcdex

basicsys@mars~/lec12>echo ${xx}
5

basicsys@mars~/lec12>echo $xx
5
```

<u>הפעלת סקריפט ב - bash בצורה: שם הסקריפט</u>

כאשר מפעילים סקריפט בצורה: שם הסקריפט

נפתח תת תהליך שמריץ את הסקריפט, המשתנים של הסקריפט הם מקומיים

לסקריפט ושינוי בהם לא ישפיע על המשתנים של התהליך שקרא לסקריפט.

```
basicsys@mars~/lec13>cat P1
x=9
```

```
basicsys@mars~/lec13>x=100

basicsys@mars~/lec13>P1

basicsys@mars~/lec13>echo $x
100

basicsys@mars~/lec13>y=$(P1)

basicsys@mars~/lec13>echo $x
100
```

<u>הפעלת סקריפט ב - bash בצורה: שם הסקריפט . (נקודה)</u>

כאשר מפעילים סקריפט בצורה: שם הסקריפט .

לא נפתח תת תהליך שמריץ את הסקריפט, אלא ששורות הסקריפט מצטרפות לתהליך הנוכחי. ולכן בצורה זו המשתנים של הסקריפט הם אותם משתנים של התהליך שקרא לסקריפט, וכל שינוי בהם ישפיע על התהליך שקרא לסקריפט .

```
basicsys@mars~/lec13>. P1

basicsys@mars~/lec13>echo $x
9
```

ביצוע פעולות על מחרוזות:

{שם משתנה $#} מוחלף במספר התווים שנמצאים במחרוזת שהמשתנה מכיל .

```
basicsys@mars~/lec12>x=abcdef

basicsys@mars~/lec12>echo ${#x}
6
```

הפקודה expr מאפשרת לבצע פעולות שונות על מחרוזות כפי שיתואר להלן .

expr length מחרוזת

הפקודה מדפיסה את מספר התווים שנמצאים במחרוזת .

```
basicsys@mars~/lec12>expr length $x
6

basicsys@mars~/lec12>expr match abcdef ab.[cd]
4
```

ביטוי רגולארי מחרוזת expr match

אם בביטוי הרגולארי אין סוגריים הפקודה מחזירה את מספר התווים בחלק של
המחרוזת שהתאים לביטוי הרגולארי, אם בביטוי הרגולארי יש סוגריים אז
הפקודה מחזירה את תת המחרוזת שהתאימה לסוגריים הראשונים שבביטוי
הרגולארי. את ההתאמה מבצעים על התחלת המחרוזת בלבד, והביטוי
הרגולארי הוא בסגנון ביטוי רגולארי בסיסי (כמו של grep) ולכן צריך להקדים \
לסימנים כמו) (} { + וכו

```
basicsys@mars~/lec12>expr match abcdef a*.[cd]
3

basicsy~/lec12>expr match abcdef '\(a*.[cd]\)'
abc

~/lec12>expr match ababcdcde '\(\(ab\)\+\(cd\)\+\)'
ababcdcd

~/lec12>expr match ababcdcde '\(\(ab\)\+\)\(cd\)\+'
abab
```

קבוצת תווים מחרוזת expr index

מחזירה את ההופעה הראשונה במחרוזת של תו מתוך קבוצת התווים. אם התו
לא מופיע במחרוזת מחזירה 0.

```
basicsys@mars~/lec12>expr index abcde db
2

basicsys@mars~/lec12>expr index abcde rc
3

basicsys@mars~/lec12>expr index abcde rt
0
```

expr substr מחרוזת 1 מספר 2 מספר

מחזירה את תת המחרוזת החל מתו מספר 1 כאשר לוקחים מספר 2 תווים .
מספור התווים במחרוזת מתחיל מ - 1:

```
basicsys@mars~/lec12>expr substr abcdefg 3 4
cdef
basicsys@mars~/lec12>expr substr abcdefg 3 7
cdefg
```

${ שם משתנה : 1 מספר : 2 מספר }

מוחלף בתת המחרוזת שמתקבלת מהמחרוזת שהמשתנה מכיל החל מתו
מספר 1 כאשר לוקחים מספר 2 תווים. במידה ומספר 2 לא קיים אז לוקחים תווים
עד סוף המחרוזת. מספור התווים מתחיל מ 0. –

```
basicsys@mars~/lec12>x=abcdefg

basicsys@mars~/lec12>echo ${x:3:4}
defg

basicsys@mars~/lec12>echo ${x:3}
defg
```

```
basicsys@mars~/lec12>y=${x:2}

basicsys@mars~/lec12>echo "$y"
cdefg
```

${ ביטוי בסגנון גלוב # שם משתנה }

מוחלף בתת המחרוזת שמתקבלת מהמחרוזת שהמשתנה מכיל לאחר קיצוץ מצד שמאל של המחרוזת את החלק (הקצר ביותר) שהתאים לביטוי לפי חוקי גלוב. ערך המשתנה נשאר ללא שינוי.

```
basicsys@mars~/lec12>echo ${x#a??}
defg

basicsys@mars~/lec12>echo ${x#a*}
bcdefg
```

${ ביטוי בסגנון גלוב ## שם משתנה }

מוחלף בתת המחרוזת שמתקבלת מהמחרוזת שהמשתנה מכיל לאחר קיצוץ מצד שמאל של המחרוזת את החלק (הארוך ביותר) שהתאים לביטוי לפי חוקי גלוב. ערך המשתנה נשאר ללא שינוי.

```
basicsys@mars~/lec12>echo ${x##a*}
```

${ ביטוי בסגנון גלוב % שם משתנה }

מוחלף בתת המחרוזת שמתקבלת מהמחרוזת שהמשתנה מכיל לאחר קיצוץ מצד ימין של המחרוזת את החלק (הקצר ביותר) שהתאים לביטוי לפי חוקי גלוב. ערך המשתנה נשאר ללא שינוי.

```
basicsys@mars~/lec12>echo ${x%e*}
abcd
basicsys@mars~/lec12>x=fabcdab
basicsys@mars~/lec12>echo ${x%ab*}
```

fabcd

{ ביטוי בסגנון גלוב %% שם משתנה }$

מוחלף בתת המחרוזת שמתקבלת מהמחרוזת שהמשתנה מכיל לאחר קיצוץ מצד ימין של המחרוזת את החלק (הארוך ביותר) שהתאים לביטוי לפי חוקי גלוב. ערך המשתנה נשאר ללא שינוי.

```
basicsys@mars~/lec12>echo ${x%%ab*}
f
```

{מחרוזת / 1 ביטוי בסגנון גלוב/ שם משתנה }$

מוחלף בתת המחרוזת שמתקבלת מהמחרוזת שהמשתנה מכיל לאחר החלפה אחת של תת המחרוזת הראשונה משמאל שמתאימה לביטוי (כאשר לוקחים את החלק הארוך ביותר שמתאים) לפי חוקי גלוב במחרוזת .1 ערך המשתנה נשאר ללא שינוי.

```
basicsys@mars~/lec12>x=ababcdab

basicsys@mars~/lec12>echo ${x/ab/zzz}
zzzabcdab

basicsys@mars~/lec12>echo ${x/ab*/zzz}
zzz
```

{מחרוזת / 1 ביטוי בסגנון גלוב // שם משתנה }$

מוחלף בתת המחרוזת שמתקבלת מהמחרוזת שהמשתנה מכיל לאחר החלפות של כל תתי המחרוזות שמתאימות לביטוי לפי חוקי גלוב במחרוזת . 1 ערך המשתנה נשאר ללא שינוי.

```
basicsys@mars~/lec12>echo ${x//ab/zzz}
zzzzzzcdzzz
```

הפקודה rev

מבנה הפקודה: רשימת שמות קבצים rev

מדפיסה את תוכן הקבצים לאחר ביצוע reverse על השורות שלהם .

```
basicsys@mars~/lec12>cat F1
abc def
gh 123 45

basicsys@mars~/lec12>rev F1
fed cba
54 321 hg

basicsys@mars~/lec12>echo abc | rev
cba
```

<u>הרחבת סוגריים מסולסלים (brace expansion)</u>

הביטוי {מחרוזת5,מחרוזת4} {מחרוזת3,מחרוזת2,מחרוזת1}
מוחלף על ידי הסורק בכל האפשרויות של הצרופים שלהם .

```
basicsys@mars~/lec12>echo a{b,c}e
abe ace

basicsys@mars~/lec12>echo a{b,c}{1,2,3}r
ab1r ab2r ab3r ac1r ac2r ac3r
```

הביטוי {מספר2..מספר1}
מוחלף על ידי הסורק בסדרת המספרים החל ממספר 1 ועד מספר (2 כולל).

```
basics~/lec12>for x in {1..10}; do echo $x ; done
 1
 2
 3
 4
```

```
5
6
7
8
9
10
```

```
basi~/lec12>for x in $(seq 10); do echo $x ; done
1
2
3
4
5
6
7
8
9
10
```

```
basicsys@mars~/lec12>echo {10..1}
10 9 8 7 6 5 4 3 2 1
```

```
basicsys@mars~/lec12>echo {10..3}
13  10 9 8 7 6 5 4 3
```

```
basicsys@mars~/lec12>echo {c h}
```

{c h}

הביטוי {תו2 .. תו1}

מוחלף על ידי הסורק בסדרת התווים החל מתו 1 ועד תו (2 כולל).

```
basicsys@mars~/lec12>echo {c..h}
c d e f g h
```

<u>הפקודה eval</u>

מבנה הפקודה: רשימת פרמטרים eval

הסורק עובר על הפרמטרים ויוצר מהם מחרוזת. לאחר מכן הפקודה eval

גורמת לכך שהמחרוזת שנוצרה מופעלת כפקודת bash

```
basicsys@mars~/lec12>x=3

basicsys@mars~/lec12>y=8

basicsys@mars~/lec12>echo {$x..$y}
{3..8}

basicsys@mars~/lec12>echo "{$x..$y}"
{3..8}

basicsys@mars~/lec12>echo \{$x..$y\}
{3..8}

basicsys@mars~/lec12>eval echo {$x..$y}
3 4 5 6 7 8

basicsys@mars~/lec12>echo $(echo {$x..$y})
{3..8}
```

מערכים:

הגדרת מערך: (רשימת אברי המערך מופרדים על ידי רווח) = שם המערך

אברי המערך יאותחלו באופן הבא: האיבר הראשון ברשימה יהיה באינדקס 0

האיבר השני ברשימה יהיה באינדקס 1 וכן הלאה..

```
basicsys@mars~/lec12>a=20

basicsys@mars~/lec12>echo $a
20

basicsys@mars~/lec12>a=(20 30 40)

basicsys@mars~/lec12>echo $a
20
```

${ אינדקס]שם מערך}

מוחלף בערך של האיבר במערך שנמצא באינדקס הנתון .

```
basicsys@mars~/lec12>echo ${a[0]}
20

basicsys@mars~/lec12>echo ${a[1]}
30

basicsys@mars~/lec12>echo ${a[2]}
40
```

${]@ שם מערך}

מוחלף ברשימת כל אברי המערך (לאחר צמצום רווחים .)

```
basicsys@mars~/lec12>echo ${a[@]}
20 30 40

basicsys@mars~/lec12>echo "${a[@]}"
```

20 30 40

"${ArrayName[@]}

מוחלף ברשימת כל אברי המערך (ללא צמצום רווחים)

```
basicsys@mars~/lec12>a=("ab 10" "cd 20" 40)

basicsys@mars~/lec12>echo "${a[@]}"
ab 10 cd 20 40

basicsys@mars~/lec12>echo ${a[@]}
ab 10 cd 20 40

basicsys@mars~/lec12>a[1]="40 1111"

basicsys@mars~/lec12>echo "${a[@]}"
ab 10 40 1111 40
```

${#ArrayName[@]}

מוחלף במספרי אברי המערך

```
basicsys@mars~/lec12>b=({2..5}{a..c})

basicsys@mars~/lec12>echo "${b[@]}"
2a 2b 2c 3a 3b 3c 4a 4b 4c 5a 5b 5c

basicsys@mars~/lec12>echo "${#b[@]}"
12

basicsys@mars~/lec12>echo "${#a[@]}"
3
```

```
basicsys@mars~/lec12>echo ${#a[@]}
3

basicsys@mars~/lec12>echo ${a[@]}
ab 10 40 1111 40

basicsys@mars~/lec12>echo "${b[@]}"
2a 2b 2c 3a 3b 3c 4a 4b 4c 5a 5b 5c
```

${מספר : 2 מספר [: 1]@ שם מערך}

מוחלף ברשימה שמתקבלת מאברי המערך החל מאיבר מספר 1 כאשר לוקחים מספר 2 איברים. מספור אברי המערך מתחיל מ 0. –

```
basicsys@mars~/lec12>echo "${b[@]:2:3}"
2c 3a 3b

basicsys@mars~/lec12>b=({a..c}{1..2}{d..f})

basicsys@mars~/lec12>echo "${b[@]}"
a1d a1e a1f a2d a2e a2f b1d b1e b1f b2d b2e b2f
c1d c1e c1f c2d c2e c2f

basicsys@mars~/lec12>echo ${#b[@]}
18
```

${ שם מערך#}

מוחלף במספר התווים באיבר הראשון במערך (זאת אומרת האיבר שנמצא באינדקס 0).

```
basicsys@mars~/lec12>echo ${#b}
3
```

```
basicsys@mars~/lec12>echo ${b}
a1d
```

${ ביטוי בסגנון גלוב [/ # @]שם מערך}

מוחלף ברשימת אברי המערך לאחר ביצוע קיצוץ מימן בכל אחד מאברי המערך של החלק הקצר ביותר שמתאים לביטוי הרגולארי.

```
basicsys@mars~/lec12>echo ${b[@]/%?}
a1 a1 a1 a2 a2 a2 b1 b1 b1 b2 b2 b2 c1 c1 c1 c2
c2 c2
```

באופן דומה ניתן לבצע את שאר הפעולות על מחרוזות על כל אחד מאברי המערך כפי שמוצג בדוגמאות הבאות:

```
basicsys@mars~/lec12>echo ${b[@]/%%?}
a1d a1e a1f a2d a2e a2f b1d b1e b1f b2d b2e b2f
c1d c1e c1f c2d c2e c2f

basicsys@mars~/lec12>echo ${b[@]/a/zzz}
zzz1d zzz1e zzz1f zzz2d zzz2e zzz2f b1d b1e b1f
b2d b2e b2f c1d c1e c1f c2d
c2ec2f

basicsys@mars~/lec12>b=({a..c}1{a..c})

basicsys@mars~/lec12>echo ${b[@]}
a1a a1b a1c b1a b1b b1c c1a c1b c1c

basicsys@mars~/lec12>echo ${b[@]/a/zzz}
zzz1a zzz1b zzz1c b1zzz b1b b1c c1zzz c1b c1c

basicsys@mars~/lec12>echo ${b[@]//a/zzz}
zzz1zzz zzz1b zzz1c b1zzz b1b b1c c1zzz c1b c1c
```

1. Declaring an Array and Assigning values

In bash, array is created automatically when a variable is used in the format like,

```
name[index]=value
```

- name is any name for an array
- index could be any number or expression that must evaluate to a number greater than or equal to zero.You can declare an explicit array using declare -a arrayname.

```
$ cat arraymanip.sh

#! /bin/bash

Unix[0]='Debian'

Unix[1]='Red hat'

Unix[2]='Ubuntu'

Unix[3]='Suse'



echo ${Unix[1]}



$./arraymanip.sh

Red hat
```

To access an element from an array use curly brackets like ${name[index]}.

2. Initializing an array during declaration

Instead of initializing an each element of an array separately, you can declare and initialize an array by specifying the list of elements (separated by white space) with in a curly braces.

```
Syntax:

declare -a arrayname=(element1 element2 element3)
```

If the elements has the white space character, enclose it with in a quotes.

```
#! /bin/bash

$cat arraymanip.sh

declare -a Unix=('Debian' 'Red hat' 'Red hat' 'Suse' 'Fedora');
```

declare -a declares an array and all the elements in the parentheses are the elements of an array.

3. Print the Whole Bash Array

There are different ways to print the whole elements of the array. If the index number is @ or *, all members of an array are referenced. You can traverse through the array elements and print it, using looping statements in bash.

```
echo ${Unix[@]}



# Add the above echo statement into the arraymanip.sh

#./t.sh
```

```
Debian Red hat Ubuntu Suse
```

Referring to the content of a member variable of an array without providing an index number is the same as referring to the content of the first element, the one referenced with index number zero.

4. Length of the Bash Array

We can get the length of an array using the special parameter called $#.

${#arrayname[@]} gives you the length of the array.

```
$ cat arraymanip.sh

declare -a Unix=('Debian' 'Red hat' 'Suse' 'Fedora');

echo ${#Unix[@]} #Number of elements in the array

echo ${#Unix}  #Number of characters in the first element of the array.i.e Debian

$./arraymanip.sh

4

6
```

5. Length of the nth Element in an Array

${#arrayname[n]} should give the length of the nth element in an array.

```
$cat arraymanip.sh

#! /bin/bash
```

```
Unix[0]='Debian'

Unix[1]='Red hat'

Unix[2]='Ubuntu'

Unix[3]='Suse'



echo ${#Unix[3]} # length of the element located at index 3 i.e Suse



$./arraymanip.sh

4
```

6. Extraction by offset and length for an array

The following example shows the way to extract 2 elements starting from the position 3 from an array called Unix.

```
$cat arraymanip.sh

Unix=('Debian' 'Red hat' 'Ubuntu' 'Suse' 'Fedora' 'UTS' 'OpenLinux');

echo ${Unix[@]:3:2}
```

```
$./arraymanip.sh


Suse Fedora
```

The above example returns the elements in the 3rd index and fourth index. Index always starts with zero.

7. Extraction with offset and length, for a particular element of an array

To extract only first four elements from an array element . For example, Ubuntu which is located at the second index of an array, you can use offset and length for a particular element of an array.

```
$cat arraymanip.sh

#! /bin/bash



Unix=('Debian' 'Red hat' 'Ubuntu' 'Suse' 'Fedora' 'UTS' 'OpenLinux');

echo ${Unix[2]:0:4}



./arraymanip.sh

Ubun
```

The above example extracts the first four characters from the 2nd indexed element of an array.

8. Search and Replace in an array elements

The following example, searches for Ubuntu in an array elements, and replace the same with the word ‘SCO Unix’.

```
$cat arraymanip.sh

#!/bin/bash

Unix=('Debian' 'Red hat' 'Ubuntu' 'Suse' 'Fedora' 'UTS' 'OpenLinux');

echo ${Unix[@]/Ubuntu/SCO Unix}

$./arraymanip.sh

Debian Red hat SCO Unix Suse Fedora UTS OpenLinux
```

In this example, it replaces the element in the 2nd index ‘Ubuntu’ with ‘SCO Unix’. But this example will not permanently replace the array content.

9. Add an element to an existing Bash Array

The following example shows the way to add an element to the existing array.

```
$cat arraymanip.sh

Unix=('Debian' 'Red hat' 'Ubuntu' 'Suse' 'Fedora' 'UTS' 'OpenLinux');

Unix=("${Unix[@]}" "AIX" "HP-UX")

echo ${Unix[7]}
```

```
$./arraymanip.sh

AIX
```

In the array called Unix, the elements ‘AIX’ and ‘HP-UX’ are added in 7th and 8th index respectively.

10. Remove an Element from an Array

unset is used to remove an element from an array.unset will have the same effect as assigning null to an element.

```
$cat arraymanip.sh

#!/bin/bash

Unix=('Debian' 'Red hat' 'Ubuntu' 'Suse' 'Fedora' 'UTS' 'OpenLinux');



unset Unix[3]

echo ${Unix[3]}
```

The above script will just print null which is the value available in the 3rd index. The following example shows one of the way to remove an element completely from an array.

```
$ cat arraymanip.sh

Unix=('Debian' 'Red hat' 'Ubuntu' 'Suse' 'Fedora' 'UTS' 'OpenLinux');

pos=3
```

```
Unix=(${Unix[@]:0:$pos} ${Unix[@]:$(($pos + 1))})

echo ${Unix[@]}



$./arraymanip.sh

Debian Red hat Ubuntu Fedora UTS OpenLinux
```

In this example, ${Unix[@]:0:$pos} will give you 3 elements starting from 0th index i.e 0,1,2 and ${Unix[@]:4} will give the elements from 4th index to the last index. And merge both the above output. This is one of the workaround to remove an element from an array.

11. Remove Bash Array Elements using Patterns

In the search condition you can give the patterns, and stores the remaining element to an another array as shown below.

```
$ cat arraymanip.sh

#!/bin/bash

declare -a Unix=('Debian' 'Red hat' 'Ubuntu' 'Suse' 'Fedora');

declare -a patter=( ${Unix[@]/Red*/} )

echo ${patter[@]}



$ ./arraymanip.sh
```

```
Debian Ubuntu Suse Fedora
```

The above example removes the elements which has the patter Red*.

12. Copying an Array

Expand the array elements and store that into a new array as shown below.

```
#!/bin/bash

Unix=('Debian' 'Red hat' 'Ubuntu' 'Suse' 'Fedora' 'UTS' 'OpenLinux');

Linux=("${Unix[@]}")

echo ${Linux[@]}



$ ./arraymanip.sh

Debian Red hat Ubuntu Fedora UTS OpenLinux
```

13. Concatenation of two Bash Arrays

Expand the elements of the two arrays and assign it to the new array.

```
$cat arraymanip.sh

#!/bin/bash

Unix=('Debian' 'Red hat' 'Ubuntu' 'Suse' 'Fedora' 'UTS' 'OpenLinux');
```

```
Shell=('bash' 'csh' 'jsh' 'rsh' 'ksh' 'rc' 'tcsh');



UnixShell=("${Unix[@]}" "${Shell[@]}")

echo ${UnixShell[@]}

echo ${#UnixShell[@]}



$ ./arraymanip.sh

Debian Red hat Ubuntu Suse Fedora UTS OpenLinux bash csh jsh rsh ksh rc tcsh

14
```

It prints the array which has the elements of the both the array ‘Unix’ and ‘Shell’, and number of elements of the new array is 14.

14. Deleting an Entire Array

unset is used to delete an entire array.

```
$cat arraymanip.sh

#!/bin/bash

Unix=('Debian' 'Red hat' 'Ubuntu' 'Suse' 'Fedora' 'UTS' 'OpenLinux');

Shell=('bash' 'csh' 'jsh' 'rsh' 'ksh' 'rc' 'tcsh');
```

```
UnixShell=("${Unix[@]}" "${Shell[@]}")

unset UnixShell

echo ${#UnixShell[@]}



$ ./arraymanip.sh

0
```

After unset an array, its length would be zero as shown above.

15. Load Content of a File into an Array

You can load the content of the file line by line into an array.

```
#Example file

$ cat logfile

Welcome

to

thegeekstuff

Linux
```

```
Unix



$ cat loadcontent.sh

#!/bin/bash

filecontent=( `cat "logfile" `)



for t in "${filecontent[@]}"

do

echo $t

done

echo "Read file content!"



$ ./loadcontent.sh

Welcome

to

thegeekstuff
```

```
Linux

Unix

Read file content!
```

In the above example, each index of an array element has printed through for loop.

:SED

אופציה s:

החלפת הופיעה ראשונה בכל שורה של ביטוי בביטוי אחר

sed ‘s/regpex1/regpex2/’ filename

ניתן לתחום תחום שורות להחלפה לדוגמא הפעלה רק על השורה השניה:

sed ‘2s/regpex1/regpex2/’ filename

או שורות 2 ו3

sed ‘2,3s/regpex1/regpex2/’ filename

ניתן להפעיל פקודת sed רק על שורות שמכילות ביטוי רגולרי:

בדוגמא מתחת יתבצע חילוף מסויים רק בשורות שמכילות את הביטוי and.

sed '/and/s/\([^]\+\).*/\1/' F1

ניתן לתחום שורות שיתבצע עבורם החלפה ע"י 2 ביטויים רגולרים והמשמעות שההחלפות יתבצעו מהשורה הראשונה שמכילה את הביטוי הראשון עד השורה שבא מופיע הביטוי השני.

דוגמא:

נניח שתוכן הקובץ F2 :הוא

1line not in range

2another line not in range

3this line is in range since it has string hello

4this line is too in range

5this line is last line in range since it has string you

6this line is not in range

7hello this line starts a new range

8you this line ends the new range

לאחר הפעלת הפקודה :

```
sed '/hello/,/you/s/\([^ ]\+\).*/\1/' F2
```

יתקבל הפלט הבא :

1line not in range

2another line not in range

3this

4this

5this

6this line is not in range

7hello

8you

בכדי לבצע החלפות עבור כל התאמות נוסיף g בסוף הביטוי דוגמא:

```
sed ‘s/regpex1/regpex2/g’ filename
```

ניתן לבצע את ההחלפות ממספר מסוים של החלפות והלאה לדוגמא:

```
sed ‘s/regpex1/regpex2/3g’ filename
```

החל מההתאמה השלישית.

ניתן גם לבצע החלפה על ההחלפה השלישית בלבד בצורה הבאה:

```
sed ‘s/regpex1/regpex2/3’ filename
```

דוגמא כיצד להשתמש בפמטרים בתוך sed:

```
y=R
x=5
echo abcdefgh | sed 's/./'$y'/'$x
```

יתקבל הפלט :

abcdRfgh

אופציה d:

האופציה גורמת לכך שכל השורות שמתאימות לא יודפסו:

כדי לא להדפיס שורות בתחום מסויים לדוגמא לא להדפיס את כל השורות בתום 2-6 (כולל)

```
sed '2,6d' F2
```

אפשר גם להגדיר תחום ע"י ביטוי רגולרי כלומר להדפיס את כל השורות שלא עונות לביטוי מסויים:

```
sed ‘/regpex/d’ filename
```

אפשר לעשות איזה מין הפוך הפוך כדי להדפיס את כל השורות שכן מתאימות לביטוי מסוים בצורה הבאה:

```
sed ‘/regpex/!d’ filename
```

אופציה =:

הפקודה = גורמת להדפסת מספר השורה (\n בתוספת) לפני כל שורה בתחום .

F3:

```
line A
line B
line C
line D
```

לאחר הפעלת הפקודה:

```
sed '2,3=' F3
```

יתקבל הפלט הבא:

```
line A
2
line B
3
line C
line D
```

אופציה n–:

גורמת לכך שרק שורות שיתבקשו להדפס ע"י p יודפסו לדוגמא:

```
sed -n '2,3=' F3
```

יתקבל הפלט :

```
2
3
```

הסימן $ - מסמל את השורה האחרונה בשורה.

אופציה p:

הדפסה של שורות שמתיאמות לביטוי רגולרי

```
sed –n ‘/regex/p’
```

ניתן להדפיס תחם שורות ע"י השורה הבא:

```
sed –n ‘2,4p’ filename
```

שימוש באופציה p ללא n– יגרום להכפלת שורות כמו בדוגמא הבאה:

```
sed 'p' F1
```

הפלט הוא:

```
I have three dogs and two cats
I have three dogs and two cats
The dog chase the cat
The dog chase the cat
A dog is a dog
A dog is a dog
```

כדי להכפיל הופעה של שורה שמתאימה לביטוי רגולרי נשתמש ב p בצורה הבאה:

```
sed ‘/regpex/p’ filename
```

אופציה y:

מאפשרת לבצע החלפה של תווים לדוגמא:

```
echo 1231| sed 'y/1/a/'
```

הפלט:

```
a23a
```

שימוש בפרמטרים:

כאשר רוצים לבנות את פקודת ה - sed תוך שימוש בפרמטרים שהינם ערכים של משתנים ,
יש לדאוג שהמשתנים לא יהיו בתוך הגרשיים הבודדים , כי למשל הביטוי '$x' נשאר $x ולא
הופך להיות הערך של המשתנה x. הדוגמה הבאה מראה איך לבקש להחליף את תו מס' $x
במחרוזת בתו $y .

לאחר הפעלת הפקודות :

```
y=R
x=5
echo abcdefgh | sed 's/./'$y'/'$x
```

יתקבל הפלט :

```
abcdRfgh
```

דוגמאות ספיציפיות:

הדפסת מספר השורות בקובץ:

```
sed -n '$=' F3
```

החלפת האותיות הקטנות בגדולות:

```
sed 'y/abcdefghijklmnopqrstuvwxyz/ABCDEFGHIJKLMNOPQRSTUVWXYZ/' F1
```

הדפסת שורות שמכילות 65 תווים או יותר

```
sed -n '/^.\{65\}/p'
```

הדפסת שורות שמכילות 65 תווים או פחות

```
sed '/^.\{65\}/d'
```

AWK:

הפעלת awk

ישנן שלוש אפשרויות להפעלת awk

. 1 הפעלה משורת הפקודה באופן הבא :

awk '1 תנאי { סדרת פקודות1} תנאי 2 {סדרת פקודות2} ...' רשימת קבצים

. 2 הפעלה על ידי קריאה לתוכנית awk שנמצאת בקובץ נפרד (לדוגמה קובץ

בשם B) באופן הבא:

awk -f B רשימת קבצים

: הינה במבנה הבא B שנמצאת בקובץ awk התוכנית

{ תנאי 1

סדרת פקודות לביצוע1

}

תנאי2 {

סדרת פקודות לביצוע2

}

...

הפעלת תוכנית סקריפט ב - awk שנמצאת בקובץ נפרד (לדוגמה קובץ בשם

C) באופן הבא :

רשימת קבצים C

הלוגיקה של תוכנית awk

תוכנית awk בנויה מסדרת זוגות של תנאים ופקודות לביצוע כמו במבנה הבא :

תנאי 1 {

סדרת פקודות לביצוע1

}

תנאי2 {

סדרת פקודות לביצוע2

}

...

הקלט לתוכנית מגיע מרשימת הקבצים שהועברו בזמן הפעלת awk (בכל

אחת משלושת הצורות להפעלת awk ישנה רשימת קבצים שמועברת). במידה

ורשימת הקבצים לא קיימת אז הקלט לתוכנית ה - awk מגיע מהקלט

הסטנדרטי (ברירת מחדל: מקלדת).

תוכנית ה-awk מתבצעת באופן הבא :

בשלב ההתחלתי (לפני שקוראים שורות קלט):

אם תנאי 1 הוא BEGIN מתבצעת סדרת פקודות לביצוע , 1

אם תנאי 2 הוא BEGIN מתבצעת סדרת פקודות לביצוע ,2

וכן הלאה ...

בשלב האמצעי מתבצע מעבר על כל שורות הקלט ועבור כל שורת קלט :

אם תנאי 1 מתקיים מתבצעת סדרת פקודות לביצוע ,1

אם תנאי 2 מתקיים מתבצעת סדרת פקודות לביצוע ,2

כן הלאה ...

בשלב הסופי (אחרי שכל שורות הקלט נקראו):

אם תנאי 1 הוא END מתבצעת סדרת פקודות לביצוע , 1

אם תנאי 2 הוא END מתבצעת סדרת פקודות לביצוע2, וכן הלאה

משתנים ב - awk

המשתנים ב - awk מחולקים לשני סוגים: משתני מערכת (בדרך כלל

באותיות גדולות) ומשתנים שהמשתמש מגדיר .

הטיפול במשתנים הוא בדומה לשפת C (ולא כמו ב - bash) כך שלקבלת

ערך של משתנה לא מוספים $ לפני המשתנה. משתנה שלא קיבל ערך

התחלתי מקבל ערך 0 כאשר משתמשים בו כבעל ערך מספרי (, לדוגמה

בביטוי x+5) ומקבל ערך מחרוזת ריקה כאשר משתמשים בו כמחרוזת

משתני מערכת

$0 מכיל את שורת הקלט הנוכחית

$1 מכיל את המילה הראשונה בשורת הקלט הנוכחית

$2 מכיל את המילה השנייה בשורת הקלט הנוכחית

וכן הלאה ...

עבור משתנה i $i מכיל את המילה ה - i בשורת הקלט הנוכחית .

NF מכיל את מספר המילים בשורת הקלט הנוכחית

NR מכיל את מספר שורת הקלט הנוכחית (מספור שורות הקלט מתחיל מ)1 -

FNR מכיל את מספר שורת הקלט הנוכחית בתוך הקובץ הנוכחי. (מספור

השורות מתחיל מ).1 -

FILENAME מכיל את שם קובץ הקלט הנוכחי

ARGV מערך שמכיל את רשימת הפרמטרים לתוכנית (בדרך כלל זו רשימת

הקבצים שמועברים בקריאה לתוכנית ה - awk ,) כאשר

ARGV[0] מכיל את המחרוזת awk

ARGV[1] מכיל את הפרמטר הראשון לתוכנית

ARGV[2] מכיל את הפרמטר השני לתוכנית, וכן הלאה ...

ARGC מכיל את מספר הפרמטרים לתוכנית ועוד 1

תנאים ב - awk

ביטוי רגולארי/ התנאי מתקיים אם שורת הקלט הנוכחית מתאימה לביטוי (לפי

חוקים של ביטויים רגולאריים).

כל התנאים הבאים אפשריים (המשמעות של כל תנאי ברורה):

מספר < 2 מספר1 מספר 2 =< מספר 1

מספר > 2 מספר 1 מספר 2 => מספר 1

מספר 2 == מספר 1 מספר 2 =! מספר 1

מחרוזת 2 == מחרוזת1 מחרוזת 2 =! מחרוזת1

תנאי 2 && תנאי1 תנאי 2 || תנאי1

ניתן להשתמש בסוגריים (אין צורך בהקדמת \ לסוגריים) בהרכבת תנאים

כמו למשל: (תנאי 4 || תנאי) 3 && (תנאי 2 || תנאי1)

ניתן להשתמש ב - ! לשלילת תנאי כמו למשל (תנאי)1 !

BEGIN התנאי הזה מתקיים רק בשלב ההתחלתי שלפני קריאת שורות הקלט .

END התנאי הזה מתקיים רק בשלב הסופי שלאחר קריאת שורות הקלט .

הפקודה print

print string1, string2 מדפיסה את המחרוזות כאשר בין המחרוזות

מודפס הערך של משתנה המערכת OFS כאשר הברירת מחדל הוא תו רווח בודד,"OFS="delim" ישנה את התו המפריד בין השורות.

כאשר מתבצעת הפקודה print string1 string2 הן יודפסו ללא רווח ובסוף יודפס ערך המשתנה ORS שוב הפקודה

"ORS="delim ישנה את המשתנה כאשר ברירת מחדל היא n/.

printf:

...פרמטר ,2 פרמטר , 1 מחרוזת שמכילה ביטויי המרה printf

מדפיסה את המחרוזת לאחר הח פל ת בביטויי ההמרה מרפב סירט המתאימים

להם לפי הסדר משמאל לימין. לדוגמה הפקודה:

printf "ab%5dcd%-6d",123,456

ביטוי ההמרה %d5 יוחלף על יד הדפסת 123 בתוספת שני רווחים מצד שמאל

ביטוי ההמרה %-d6 יוחלף על ידי הדפסת 456 בתוספת 3 רווחים מצד ימין.

מערכים ב AWK:

מערכים ב - awk הם אוסף לא מסודר של זוגות מהצורה: ערך, אינדקס. כאשר

האינדקסים והערכים הם מחרוזות. מערכים מהסוג הזה נקראים מערכים

אסוציאטיביים .

איתחול איבר במערך מתבצע על ידי פקודה :

שם המערך [אינדקס] = ערך

לדוגמה ההפקודה x["ab"]=12 מאתחלת את המערך a כך שבאינדקס ab

הערך הוא . 12

הוצאת איבר ממערך מתבצעת על ידי פקודה :

delete Array[index]

לדוגמה הפקודה delete x["ab"] מוציאה את האיבר בעל אינדקס ab מהמערך .

מחיקת כל אברי המערך מתבצעת על ידי פקודה :

delete ArrayName

לדוגמה הפקודה delete x מוחקת את כל אברי המערך x .

לבדיקה האם איבר נמצא במערך אפשר להשתמש בפקודה :

if(index in Araay){..}

לדוגמא if("ab" in x){..} בודק אם איבר ab נמצא במערך.

לקבלת מספר איבר המערך ניתן להשתמש length(ArrayName).

בכדי לעבור על כל אברי המהלך אפשר להשתמש בלולאת for במבנה מיוחד

for(i in Array){..}

בכל איטרציה i יקבל שם אינדקס אחר, הסדר נקבע בצורה אוט' ע"י המערכת ולא לפי סדר לכסוגרפי או נומרי כזה או וגם לא לפי סדר יצירת אברי המערך.

הפונקציה substr

מבנה הפונקציה: substr(מחרוזת1, מספר1, מספר2)

מחזירה את תת המחרוזת של מחרוזת 1 החל מתו מספר (1 מספור התווים

מתחיל מ)1 - כאשר לוקחים מספר 2 תווים . לדוגמה אם המשתנה s מכיל את

. bcd מחזירה substr(s,2,3) אזי abcde המחרוזת

הפונקציה sub

מבנה הפונקציה: sub(ביטוי רגולארי, מחרוזת 1, מחרוזת2)

משנה את מחרוזת 2 על ידי החלפת תת המחרוזת (הראשונה משמאל) של

מחרוזת 2 שמתאימה לביטוי הרגולארי במחרוזת . 1 לדוגמה אם המשתנה s

מכיל את המחרוזת cdababd אזי לאחר ביצוע sub("(ab)+","xyz",s) המשתנה

s יכיל cdxyzd

אם מחרוזת2 לא קיימת , במקומה מופיע $0 (דהינו השורה הנוכחית ש - awk

עובד עליה ,) ואז השורה הנוכחית משתנה בהתאם לכללים הנ"ל .

הפונקציה sub מחזירה את מספר החלפות שבציעה כלומר 1 או 0.

הפונקציה gsub

מבנה הפונקציה: gsub(מחרוזת2 ,מחרוזת 1 , ביטוי רגולארי)

הפונקציה פועלת באופן דומה לפונקציה sub אלא שמבצעת ההחלפות של כל

תתי המחרוזת של מחרוזת 2 שמתאימות לביטוי הרגולארי .

לדוגמה, אם המשתנה s מכיל את המחרוזת cdababwababz אזי לאחר ביצוע

gsub(“(ab)+”,"xyz",s) המשתנה s יכיל cdxyzwxyzz

הפונקציה gsub מחזירה את מספר החלפות שבצעה.

המשתנה FS

המשתנה FS מכיל ביטוי רגולארי שלפיו awk מפריד את שדות שורת הקלט. דהינו הערך של המשתנה הזה משפיע על האופן שבו יוצבו ערך למשתנים NF $1 $2 וכו' לדוגמה אם ערך המשתנה הוא +(ab) ושורת הקלט היא

zababcdcdababef אזי $1 יקבל ערך z, $2 יקבל ערך $cdc 3 יקבל ערך ef ו -NF יקבל ערך . 3

כברירת מחדל המשתנה FS מכיל תו רווח בודד ואז (באופן מיוחד) awk מפריד את שדות הקלט לפי מילים (דהינו מצמצם רצפים של רווחים ומכנים ל - $1 את המילה הראשונה, ל - $2 את המילה השניה וכו).'

אם המשתנה FS מכיל מחרוזת ריקה אז awk (באופן מיוחד) מפריד את שדות הקלט לפי תווים. לדוגמה אם שורת הקלט מכילה abc והערך של FS הוא מחרוזת ריקה, אזי $1 יכיל a, $2 יכיל b, $3 יכיל c ו - NF יכיל 3 .

משנה את מחרוזת 2 על ידי החלפת תת המחרוזת (הראשונה משמאל) של מחרוזת 2 שמתאימה לביטוי הרגולארי במחרוזת . 1 לדוגמה אם המשתנה s מכיל את המחרוזת cdababd אזי לאחר ביצוע sub(/(ab)+/,"xyz",s) המשתנה s יכיל cdxyzd .

הפונקציה split

מבנה הפונקציה: split(ביטוי רגולארי,שם מערך ,מחרוזת)

מפרידה את המחרוזת לחלקים בהתאם לביטוי הרגולארי ומכניסה את החלקים

שהופרדו כאברי המערך ששמו ניתן בפרמטר השני (האינדקסים של אברי

המערך הם מספרים שלמים החל מ 1 -).

לדוגמה לאחר הקריאה לפונקציה split(“xababcdabe”,A,”(ab)+“) המערך A

המערך אם A[1]=x ,A[2]=cd ,A[3]=e בלבד הבאים האיברים שלושת את יכיל:

A לא היה קיים לפני הקריאה לפונקציה, אזי הוא נוצר על ידי הפונקציה. אם

המערך A היה קיים לפני הקריאה לפונקציה, אזי האיברים שהיו בו לפני

הקריאה לפונקציה ימחקו ולאחר הקריאה לפונקציה הוא יכיל רק את שלושת

האיברים הנ"ל .

אם בקריאה לפונקציה לא מופיע ביטוי רגולארי, אזי ההפרדה של המחרוזת

לחלקים תהיה לפי הערך של המשתנה FS, דהינו כפי שמתבצעת ההפרדה של

שורת הקלט הנוכחית למשתנים ...$1,$2,$3:

הפונקציה getline:

הפונקציה גורמת לכך שמתבצע מעבר לשורת הקלט הבאה באופן מידי (ולא

לפי מתעדכנים $0,$1,$2,NF,NR,FNR,... המשתנים . (awk - ה תוכנית בסיום

שורת הקלט הבאה. כאשר תוכנית ה - awk תגיע לסיומה יתבצע מעבר לשורת

קלט נוספת (כפי שקורה בכל סיום תוכנית awk).

<u>שם משתנה getline:</u>

גורמת לכך שערך המשתנה מתעדכן לפי שורת הקלט הבאה. פרט לכך לא מתבצע כל שינוי נוסף, ולכן ערכי המשתנים $0,$1,$2,NF,NR,FNR,... לא מתעדכנים.

לדוגמא : getline x לאחר

ערך המשתנה x ישווה לשורת הקלט הבאה והמשתנה $0 ישאר ללא שינוי.

כאשר תוכנית ה - awk תגיע לסיומה יתבצע מעבר לשורת הקלט הבאה (כפי

שקורה בכל סיום תוכנית awk ,) ואז הערך של המשתנה $0 יהיה שווה לערך

המשתנה x

<u>getline <"שם קובץ":</u>

גורמת לכך שערכי המשתנים $0,$1,$2,NF,NR,FNR,... מתעדכנים לפי שורת הקלט הבאה מתוך הקובץ. ערכי המשתנים NR ו - FNR לא מתעדכנים.

<u>getline שם משתנה <"שם קובץ"</u>

גורמת לכך שערך המשתנה מתעדכן לפי שור הקלט הבאה מהקובץ . לפי שורת הקלט הבאה מתוך הקובץ. פרט לכך לא מתבצע כל שינוי נוסף, ולכן ערכי המשתנים ...$0,$1,$2,NF,NR,FNR לא מתעדכנים.

הערה: בכלל אחת מצורות של שימוש בפונקציה היא מחזירה 1 אם נקראה שורה ע"י הפונקציה, במקרה והפונקציה סיימה לקרוא מהקובץ כי הגיע לסופו היא מחזירה 0 ואם היא מחזירה 1- סימן שקרתה שגיאה.

פונקציות שמוגדרות על ידי המשתמש :

המבנה של הגדרת פונקציה:

(רשימת ארגומנטים) שם הפונקציה function

}

גוף הפונקציה

{

המשתנים ברשימת הארגומנטים מקבלים ערך בזמן הקריאה לפונקציה .

משתנים שאינם מערכים מועברים by value ומשתנים מסוג מערך

מועברים by reference. המשתנים שברשימת הארגומנטים הם לוקליים

לפונקציה, ביציאה מהפונקציה הם נעלמים .

לדוגמה , אם x הוא משתנה בעל ערך 5 ונקרא לפונקציה f1(x) ובהגדרת

הפונקציה רשום :

function f1(a)

אזי המשתנה a בתחילת הפונקציה יקבל ערך . 5 השמת ערך 6 למשתנה a

בתוך הפונקציה לא תשנה את ערך המשתנה x. לכן העברת פרמטרים כזו

נקראת העברה by value .

אם x הוא משתנה מסוג מערך ו ל - x[1] יש ערך 5 ונקרא לפונקציה f1(x) ובתוך

הפונקציה יהיה רשום a[1]=8 אזי ביציאה מהפונקציה הערך של x[1] יהיה 8.

לכן העברת פרמטרים כזו נקראת העברה by reference .

הפקודה ערך return יוצאת מהפונקציה ומחזירה ערך. אם לא רשום ערך

אז הערך המוחזר מהפונקציה הוא null. אם פונקציה מסתיימת ללא פקודת

return אזי הערך המוחזר מהפונקציה אינו מוגדר .

כתיבה לקובץ

<u>כתיבה לקובץ ב - awk על ידי "שם הקובץ" < print</u>

הפקודה: print 1 > מחרוזת "שם קובץ"

ב - awk פותחת את הקובץ לכתיבה (החל מתחילת הקובץ) וכותבת את

מחרוזת 1 בשורה הראשונה של הקובץ. אם הקובץ לא קיים הפקודה יוצרת

אותו. אם הקובץ קיים אז תוכנו נדרס ולאחר ביצוע הפקודה מופיעה בו רק

מחרוזת 1 בשורה הראשונה .

כל פקודה נוספת מהצורה הנ"ל כותבת שורה נוספת לקובץ. ניתן לסגור את הקובץ על ידי הפקודה ("שם הקובץ") close ואז הפקודה הבאה מהסוג הנ "ל תכתוב שוב לתחילת הקובץ (ותדרוס את מה שהיה שם קודם לכן).

```
basicsys@mars~/lec13>cat F1
aaa

basicsys@mars~/lec13>cat P1
BEGIN { print "abc" >"F1 "
 print "def" >"F1 "
 print "gh" >"F1 "
 {
basicsys@mars~/lec13>awk -f P1

basicsys@mars~/lec13>cat F1
abc
def
gh

basicsys@mars~/lec13>cat P1
BEGIN { print "abc" >"F1 "
 print "def" >"F1 "
 print "gh" >"F1 "
 close("F1")
 print "xyz">"F1 "
 {
basicsys@mars~/lec13>awk -f P1
basicsys@mars~/lec13>cat F1
xyz
```

הפקודה: "שם קובץ" << מחרוזת print 1

ב - awk פותחת את הקובץ לכתיבה (החל מסוף הקובץ) וכותבת את מחרוזת 1 לסוף הקובץ (תוכן הקובץ אינו נדרס: מחרוזת 1 מצטרפת לתוכן הקובץ בסופו). אם הקובץ לא קיים הפקודה יוצרת אותו. לאחר מכן כל כתיבה מהצורה הנ"ל או מהצורה "שם קובץ" < מחרוזת print 1 כותבת את המחרוזת לסוף הקובץ .

```
basicsys@mars~/lec13>cat P1
BEGIN { print "abc" >>"F1 "
 print "def" >"F1 "
 print "gh" >>"F1 "
 {
basicsys@mars~/lec13>awk -f P1

basicsys@mars~/lec13>cat F1
xyz
abc
def
gh
```

שיעור בית:

מטלה 6:

כתוב תוכנית Script -ב Bash בשם P6.1 שמקבלת כפרמטר שם קובץ ומדפיסה לפלט מספר שמציין את מספר המילים השונות מבין המילים שמופיעות בין שלוש לחמש פעמים בקובץ (כולל שלוש וכולל חמש).

```
f=$(cat $1)
echo -n $f | tr " " "\n"|sort| uniq -c | sort >| tmp
```

```
ct=$[0]
while read i
do
echo -n $i >| tmp2
z=$(cut -d" " -f1 tmp2)
if [ $z -eq 3 -o $z -eq 4 -o $z -eq 5 ]
then
ct=$[$ct+1]
fi
done<tmp
echo $ct
```

כתוב תוכנית Script -ב Bash בשם P6.2 שמקבלת כפרמטר מחרוזת ומדפיסה לפלט את מספר החזרות של התו הלפני אחרון במחרוזת (. ניתן להניח שבמחרוזת יש לפחות שני תווים)

```
echo $1 | tr -d "\n" >| tmp
z=$(wc -m < tmp)
t=$(cut -c$[$z-1] tmp)
egrep -o $t tmp >| tmp2
wc -l < tmp2
```

כתוב תוכנית Script -ב Bash בשם P6.3 שמקבלת כפרמטר מחרוזת ומדפיסה מספר שמציין את המספר הגדול ביותר של חזרות רצופות של איזשהו תו במחרוזת .

לדוגמה ,

```
echo >| tmp2
echo $1 | tr -d "\n"  >| tmp
z=$(wc -m < tmp)
max=0
for i in $(seq 1 $z)
do
t=$(cut -c$i tmp)
```

```
egrep -o "$t"+ tmp >> tmp2
done
wc -L < tmp2
```

כתוב תוכנית Script -ב Bash בשם P6.4 שמקבלת כפרמטר מחרוזת שמכילה שם קובץ ומדפיסה לפלט שורה אחת שמכילה את רשימת המילים שחוזרות מספר גדול ביותר של פעמים בקובץ, כאשר המילים ברשימה ממוינות בסדר לכסיקוגרפי יורד ומופרדות על ידי תו פסיק בודד בין כל שתי מילים .
ניתן להניח שהמילים בקובץ מורכבות מאותיות אנגליות קטנות בלבד.

```
z=$(cat $1)
echo >| temp4
echo $z|tr " " "\n" | sort | uniq -c| sort -r >| temp
t=$(head -1 temp )
echo $t >| temp2
max=$(cut -d" " -f1 <temp2 )
while read i
do
echo $i >| temp3
maxt=$(cut -d" " -f1 temp3)
if [ $maxt -eq $max ]
then
cut -d" " -f2 temp3 >> temp4
fi
done<temp
t=$(cat temp4)
echo $t|tr " " ","
```

כתוב תוכנית Script -ב Bash בשם P6.5 ש מקבלת כפרמטרים רשימת שמות קבצים באורך כלשהו (בהמשך נקרא לה רשימה).1 התוכנית מדפיסה לפלט שורה אחת עבור כל קובץ ששמו מופיע ברשימה 1 שמכילה את שם הקובץ, לאחריו תו רווח אחד, לאחריו מספר שמציין את

מספר המילים השונות בקובץ, לאחריו תו רווח אחד ולאחריו מספר שמציין את מספר החזרות הרצופות הגדול ביותר של מילים בקובץ. על סדר השורות בפלט להיות לפי סדר הקבצים ברשימה .

```
for i in $*
do
z=$(cat $i)
echo $z|tr " " "\n" | sort -u >|tmp2
ct=$(wc -l < tmp2)
ct2=$(echo $z | tr " " "\n" | uniq -c | sort | tail -1)
echo $ct2>|tmp3
ct3=$(cut -d" " -f1 tmp3)
echo $i $ct $ct3
done
```

כתוב תוכנית Script -ב Bash בשם P6.6 שמקבלת כפרמטרים רשימת מילים באורך כלשהו . התוכנית מדפיסה לפלט את המילים ברשימה שמורכבות משתי מילים רצופות זהות כאשר כל אחת מהמילים מורכבת מרצף של שתי ספרות זהות ולאחריו רצף של שתי אותיות זהות ולאחריו רצף של שתי ספרות זהות ולאחריו רצף של שתי אותיות זהות וכן הלאה ...

על כל מילה בפלט להיות מודפסת בשורה נפרדת וסדר המילים בפלט צריך להיות לפי סדר הופעתן ברשימת הפרמטרים

```
for i in $*
do
z=$(cat $i)
echo $z|tr " " "\n" | sort -u >|tmp2
ct=$(wc -l < tmp2)
ct2=$(echo $z | tr " " "\n" | uniq -c | sort | tail -1)
echo $ct2>|tmp3
ct3=$(cut -d" " -f1 tmp3)
echo $i $ct $ct3
```

```
done
kaplshak@mars~/lec6/ex6>
kaplshak@mars~/lec6/ex6>cat P6.6
for i in $*
do
echo $i |  egrep "(([0-9])\2([a-z])\3([0-9])\4([a-z])\5)\1"
done
```

מטלה 7

כתוב תוכנית Script -ב Bash בשם P7.1 שמקבלת כפרמטרים שני מספרים שלמים גדול ים (1 -מ בהמשך נקרא להם i -ו j) ולאחריהם רשימת מילים.

התוכנית מדפיסה לפלט את המילים ברשימה שניתנות לחלוקה ל - x מי לים זהות כאשר x הוא מספר שלם כלשהו בין i -ל j (כולל i -ו j).

על כל מילה להיות מודפסת בשורה נפרדת וסדר המילים בפלט צריך להיות לפי סדר הופעתן ברשימת הפרמטרים .

ניתן להניח שהפרמטר הראשון i קטן מהפרמטר השני j

```
i=$1
j=$2
shift
shift
for z in $*
do
echo $z | egrep  "^([^ ]+)\1{$[$i-1],$[$j-1]}$"
done
```

כתוב תוכנית Script -ב Bash בשם P7.2 שמקבלת כפרמטרים מספר (בהמשך נקרא לו i) לאחר מכן שם קובץ (בהמשך נקרא לו קובץ) 1 ולאחר מכן רשימת תווים (אורך רשימת התווים אינו מוגבל).

התוכנית ומדפיסה לפלט את כל המילים בקובץ 1 שמספר ההופעות בהן של כל אחד מהתווים ברשימה שווה בדיוק ל - i .

על כל מילה להיות מודפסת בשורה נפרדת (פעם אחת בדיוק) וסדר המילים בפלט צריך

להיות לפי סדר לכסיקוגרפי עולה (בהתאם לפקודה sort ללא אופציות).

```
i=$1
fname=$2
shift
shift
cat $fname | tr " " "\n" | sort | uniq >| tmp
while read word
do
bol=1
 for tav in $*
 do
 z=$(echo $word | egrep -o . | egrep -c $tav)
 if [ $z != $i ]
  then
  bol=0
  fi
 done
 if [ $bol == 1 ]
 then
 echo $word
 fi
done<tmp
```

כתוב/כתבי תוכנית ב - Bash (דהינו קובץ Script) בשם P7.3 שמקבלת כפרמטרים
שם תיקיה (בהמשך נקרא לה תיקיה ,)1 מחרוזת (בהמשך נקרא לה מחרוזת) 1 ורשימת שמות של קבצים (מספר הקבצים ברשימה אינו מוגבל).

התוכנית מדפיסה עבור כל קובץ ששמו מופיע ברשימת הקבצים, שורה המכילה את שם הקובץ, לאחריו תו רווח יחיד ולאחריו ומספר המציי ן את מספר הקבצים בשם זה שנמצאים בתיקיה 1 בעומק כלשהו אשר מכילים את מחרוזת ב 1 שורה השנייה בקובץ .

יש לסדר את שורות הפלט לפי שמות הקבצים ברשימה בסדר לכסיקוגרפי עולה (בהתאם לפקודה sort ללא אופציות .)

```
d1=$1
str1=$2
shift
shift
for i in $(echo $* | tr " " "\n" | sort)
do
 z=$(find $d1 -type f -name $i -exec egrep -n $str1 {} \; )
 z=$(echo $z|tr " " "\n"| egrep -c 2)
 echo $i $z
done
```

כתוב/כתבי תוכנית ב - Bash (דהינו קובץ Script) בשם P7.4 שמקבלת כפרמטרים שתי תיקיות (בהמשך נקרא להן תיקיה 1 ותיקיה). 2 התוכנית מדפיסה לפלט את שמות כל הקבצים שמופיעים לפחות פעמיים בתיקיה (1 בעומק כלשהו) ואינם מופעים כלל בתיקיה 2 (בעומק כלשהו).

כל שם קובץ יודפס בשורה נפרדת וסדר השמות הוא לפי סדר לכסיקוגרפי עולה (בהתאם לפקודה sort ללא אופציות).

```
>|tmp2
find $1 -type f -exec echo {} \; >| tmp
while read z
do
t=$(echo $z|tr "/" " " | wc -w)
echo -n $z | tr "/" " " | cut -d" " -f$[$t+1] >> tmp2
done < tmp
cat tmp2 | sort | uniq -d >| tmp2
while read x
do
z=$(find $2 -type f -name $x | wc -l)
```

```
if [ $z -eq 0 ]
then
echo $x
fi
done<tmp2
```

כתוב/כתבי תוכנית ב - Bash (דהינו קובץ Script) בשם P7.5 ש מקבלת כפרמטרים רשימת מילים (בהמשך נקרא לה רשימה) 1 לאחריה המילה dirs- ולאחריה רשימת תיקיות (בהמשך נקרא לה רשימה).2 התוכנית מדפיסה לפלט עבור כל מילה שברשימה 1 שורה אחת שמכילה הפרטים הבאים :

המילה עצמה ,

תו רווח אחד ,

רשימת מספרים (עם תו רווח אחד בדיוק בין המספרים) שמכילה את מספר הקבצים שבהם מופיעה המילה כמילה בכל אחת מהתיקיות שברשימה (2 לפי סדר התיקיות שברשימה).2

תו רווח אחד ,

המילה ALL או המילה NOTALL, כאשר המילה ALL תודפס אם בכל אחת מהתיקיות שברשימה (2 בעומק כלשהו) ישנו קובץ שהמילה מופיעה בו כמילה, אחרת תודפס המילה NOTALL .

1. על סדר השורות בפלט להיות לפי סדר המילים שברשימה

```
echo "$*" | sed 's/ -dirs /\n/' >| tmp
cat tmp | head -1 >| w1
cat tmp | tail -1 >| d2
cat w1 | tr " " "\n"  >| w1
cat d2 |tr " " "\n"  >| d2
while read w
do
>|r
t=0
```

```
while read d
 do
  find $d -type f -exec egrep -l "(^|[ ]+)($w)([ ]+|$)" {} \; >| tmp
  z=$(cat tmp | wc -l)
  if [ $z -gt 0 ]
  then
  t=$[$t+1]
  fi
 echo -n "+$z"  >> r
done<d2
  tm=$(wc -l < d2)
  if [ $t -eq $tm ]
  then
   echo +ALL >> r
  else
   echo +NOTALL >>r
  fi
  echo -n "$w"
  cat r | tr "+" " "
done<w1
```

מטלה 8:

1. כתוב תוכנית Script -ב sed בשם P8.1 שמקבלת כפרמטר שם קובץ (בהמשך נקרא לו קובץ)

התוכנית מדפיסה לפלט את השורות בקובץ 1 שמכילות בדיוק 4 מילים, כאשר

בכל שורה בפלט מופיעה המילה הראשונה 3 פעמים ברצף (עם תו רווח אחד בין המילים ,) כפי שמודגם בדוגמה שלהלן .

```
sed -n '/^[^ ]\+[ ]\+[^ ]\+[ ]\+[^ ]\+[ ]\+[^ ]\+$/p' $1 >| tmp
sed 's/[^ ]\+/& & &/' tmp
```

כתוב תוכנית Script -ב sed בשם P8.2 שמקבלת כפרמטרים מחרוזת בהמשך נקרא לה

מחרוזת1) ושם קובץ (בהמשך נקרא לו קובץ).1

התוכנית מדפיסה לפלט את השורות בקובץ 1 שאינן מכילות את מחרוזת 1 כאשר

בשורות אלה מופיעה המילה האחרונה שלוש פעמים ברצף (עם תו רווח אחד בין המילים ,) כפי

שמודגם בדוגמה שלהלן

```
sed -n '/'$1'/!p' $2 >| tmp
sed 's/\([^ ]\+\)[ ]*$/\1 \1 \1/' tmp
```

כתוב תוכנית Script -ב sed בשם P8.3 שמקבלת כפרמטר שם קובץ (בהמשך נקרא לו

קובץ). 1 התוכנית מדפיסה לפלט את התוכן של קובץ 1 לאחר החלפת כל מילה בת

3 אותיות בקובץ -ב 1 reverse שלה. (תוכן קובץ 1 נשאר ללא שינוי לאחר ביצוע

התוכנית)

```
sed 's/\<\(.\)\(.\)\(.\)\>/\3\2\1/' $1
```

כתוב/כתבי תוכנית ב - Bash (דהינו קובץ Script) בשם P8.4 שמקבלת כפרמטרים שם קובץ

(בהמשך נקרא לו קובץ ו) 1 רשימת מספרים (בהמשך נקרא לה רשימה) 1 ומדפיסה לפלט את

העמודות מתוך קובץ 1 לאחר ישור ן לימין או לשמאל לפי המספרים ברשימה , 1 כפי שמתואר

בדוגמאות שבהמשך (. ניתן להשתמש בפקודה printf של awk לפתרון השאלה).

ניתן להניח שהמספרים ברשימה 1 גדולים יותר מאורכי כל המילים שנמצאות בעמודות

המתאימות להם בקובץ . (1 לדוגמה, אם בעמודה הראשונה בקובץ 1 המילה הארוכה ביותר הינה

באורך ,5 אזי בכל הפעלה של התוכנית המספר הראשון ברשימה 1 יהיה גדול מ).5 -

בנוסף ניתן להניח שבכל שורה בקובץ 1 מספר המילים הוא לפחות כמו מספר המילים

ברשימה . (1 לדוגמה אם התוכנית מופעלת עם 4 מספרים ברשימה , 1 אזי בקובץ 1 יש לפחות 4

מילים בכל שורה).

```
fname=$1
shift
while read x
do
ct=1
for i in $*
 do
  echo $x | awk '{printf "%'$i's",$'$ct'}'
```

```
 ct=$[$ct+1]
 done
echo
done<$fname
```

כתוב תוכנית Script -ב Bash בשם P8.5 שמקבלת כפרמטר שם קובץ (בהמשך נקרא לו קובץ)1
שמכיל מטריצה (לא בהכרח ריבועית) של מספרים (לא בהכרח שלמים).
התוכנית מדפיסה לפלט שורה אחת שמכילה את סכום המספרים בכל עמודה במטריצה כאשר
בין המספרים בשורה ישנו תו רווח אחד בדיוק. ניתן להשתמש ב- awk לביצוע חישוב סכומי
העמודות במטריצה .
ניתן להניח שמספר המספרים שבכל שורה בקובץ 1 זהה .

```
fname=$1
shift
while read x
do
ct=1
for i in $*
 do
   echo $x | awk '{printf "%'$i's",$'$ct'}'
  ct=$[$ct+1]
 done
echo
done<$fname
kaplshak@mars~/lec8/ex8>cat P8.5

c=$(head -1 $1|awk '{print NF}')
for x in $(seq 1 $c)
do
z=$(awk '{printf $'$x'" "}' $1)
z=$(echo $z|tr " " "+")
```

```
#echo $z $x
echo | awk '{printf '$z'" "}'
done
echo
```

מטלה 9:

1. כתוב תוכנית Script -ב sed בשם P9.1 שמקבלת כפרמטר שם קובץ (בהמשך נקרא לו קובץ) התוכנית מדפיסה לפלט את כל השורות בקובץ 1 שמכילות לפחות 3 ספרות כאשר התוכנית משכפלת בשורות אלה את כל הספרות החל מהספרה החמישית והלאה .

```
sed  '/.*[1-9].*[1-9].*[1-9].*/!d' $1 >| tmp
sed  's/[1-9]/&&/5g' tmp
```

כתוב תוכנית Script -ב sed בשם P9.2 שמקבלת כפרמטר שם קובץ ומדפיסה לפלט את כל השורות שבקובץ כאשר כל שורה שאינה מכילה ספרות 5 -ל 0בין מופיעה פעמיים בפלט

```
sed '/[1-5]/!p' $1
```

נגדיר שמטריצה (לא בהכרח ריבועית ולא בהכרח בעלת מספר שווה של מספרים בכל שורה) היא שוות שורות אם סכום המספרים בכל שורה זהה
לדוגמה המטריצה הבאה היא מטריצה שוות שורות: (כי סכום כל שורה הוא)15

```
7 2 6
15
14 1
```

כתוב תוכנית ב - awk בשם P9.3 המקבלת כפרמטרים רשימת שמות קבצים ומדפיסה שורה אחת עבור כל קובץ שמכילה את שם הקובץ, לאחריו תו רווח אחד ולאחריו YES אם הקובץ מיצג מטריצת שוות שורות או NO אם הקובץ אינו מיצג מטריצת שוות שורות. אם הקובץ מיצג מטריצת שוות שורות, אזי השורה של הקובץ תכיל בנוסף תו רווח אחד ולאחריו מספר שמציין את סכום המספרים בכל שורה בקובץ .
בנוסף בשורה האחרונה בפלט יודפס מספר שמציין את סכום המספרים הגדול ביותר בשורה מבין השורות של הקבצים שמיצגים מטריצות שוות שורות .

על סדר שורות הפלט להיות לפי סדר הקבצים ברשימת הפרמטרים לתוכנית .

/הנח הניחי שהקבצים ששמם מועבר בפרמטרים קיימים ותוכנם חוקי (דהינו השורות שלהם

מכילות מספרים עם תו רווח אחד או יותר בין המספרים).

```
#!/bin/awk -f
 {
 s=0
 for (i=1;i<=NF;i++){
  s=s+$i
 }
 X[FILENAME]=FNR
 B[FILENAME,FNR]=s
}
function check_rows(f1){
 bol=1
 for (i=1;i<X[f1];i++)
  if(B[f1,i]!=B[f1,i+1])
    {bol=0}
return bol
}
END{
 for(x=1;x<length(ARGV);x++){
 if(check_rows(ARGV[x])==1){print ARGV[x],"YES",B[ARGV[x],1];MAX[x]=B[ARGV[x],1];max=MAX[x];}
 if(check_rows(ARGV[x])==0){print ARGV[x],"NO"}
 }
 for (i in MAX)
 if(MAX[i]>max) max=MAX[i]
       print max
}
```

נגדיר שמטריצה ריבועית (דהינו מספר המספרים בכל שורה זהה ומספר השורות שווה למספר

העמודות). היא שוות היקף אם כל ארבעת הסכומים הבאים שווים זה לזה :

סכום המספרים בעמודה הראשונה

סכום המספרים בעמודה האחרונה

סכום המספרים בשורה הראשונה

סכום המספרים בשורה האחרונה .

לדוגמה המטריצה הבאה היא מטריצה שוות היקף: (כי כל אחד מארבעת הסכומים הנ"ל שווה ל- 15.

7 6 2

4 5 10

4 8 3

כתוב תוכנית ב - awk בשם P9.4 המקבלת כפרמטרים רשימת שמות קבצים ומדפיסה שורה אחת עבור כל קובץ שמכילה את שם הקובץ, לאחריו תו רווח אחד ולאחריו YES אם הקובץ מיצג מטריצת שוות היקף או NO אם הקובץ אינו מיצג מטריצת שוות היקף .
בנוסף בשורה האחרונה בפלט יודפס מספר שמציין את סכום המספרים הגדול ביותר בקובץ מבין כל הקבצים ששמם מופיע ברשימת הפרמטרים לתוכנית .

על סדר שורות הפלט להיות לפי סדר הקבצים ברשימת הפרמטרים לתוכנית .

/הנח הניחי שהקבצים ששמם מועבר בפרמטרים קיימים ותוכנם חוקי (דהינו השורות שלהם מכילות מספרים עם תו רווח אחד או יותר בין המספרים).

```
#!/bin/awk -f
{
nr[FILENAME]=FNR
for(i=1;i<=NF;i++){
    B[FILENAME,FNR,i]=$i
```

```
 }
sum=0
for(i=1;i<=NF;i++)
  sum=sum+$i
X[FILENAME]=X[FILENAME]+sum
}
function checkRow(f1){
bol=1
sum=0
for(i=1;i<=nr[f1];i++){
 sum=sum+B[f1,1,i]
 }
sum1=0
for(i=1;i<=nr[f1];i++){
 sum1=sum1+B[f1,i,1]
 }
sum2=0
for(i=1;i<=nr[f1];i++){
 sum2=sum2+B[f1,nr[f1],i]
 }
sum3=0
for(i=1;i<=nr[f1];i++){
 sum3=sum3+B[f1,i,nr[f1]]
 }
      if(sum!=sum1 || sum!=sum2 || sum!=sum3) bol=0
      if(bol==1) return 1
      if(bol==0) return 0
}
END{
```

```
max=X[ARGV[1]]
for(x=1;x<length(ARGV);x++){
        if(X[ARGV[x]]>max) max=X[ARGV[x]];
}
for(x=1;x<length(ARGV);x++)
{
  if(checkRow(ARGV[x])==1) print ARGV[x],"YES";
  if(checkRow(ARGV[x])==0) print ARGV[x],"NO";
}
print max
}
```

כתוב/כתבי תוכנית ב - awk בשם P9.5 שמקבלת כפרמטר שם קובץ .
התוכנית מדפיסה לפלט מספר שמציין את אורך המילה הארוכה ביותר בקובץ מבין המילים שאינן מכילות כלל אותיות אנגליות (קטנות או גדולות .)

הערה: כדי לבדוק אם מילה שנמצאת במשתנה s אינה מכילה אותיות אנגליות כלל, ניתן להיעזר בפקודה sub שמחליפה אות אנגלית כלשהי בספרה כלשהי, ואז לבדוק האם ערך המשתנה s לאחר ביצוע הפקודה sub זהה לערך המשתנה לפני ביצוע הפקודה sub .
אפשרות נוספת היא לבדוק את הערך החוזר מהפקודה sub . ל"הנ אם הערך החוזר שווה ל 1 - סימן שבוצעה החלפה ובמילה יש אותיות אנגליות, אחרת הערך החוזר הוא 0 וזה סימן שלא בוצעה החלפה ובמילה אין אותיות אנגליות

```
{
for(i=1;i<=NF;i++){
x=sub("[a-z]","1",$i)
  if(x==0)
  {
  B[NR,i]=length($i)
  max=length($i);
  }
```

```
}
}
END{
for (i in B)
    if(B[i]>max) max=B[i]
print max
}
```

מטלה 10:

כתוב/כתבי תוכנית ב - awk בשם P10.1 שמקבלת כפרמטרים רשימת שמות של קבצים .

התוכנית מדפיסה לפלט עבור כל קובץ שורה המכילה את שם הקובץ לאחר מכן תו רווח אחד ולאחר מכן את אורך השורה הארוכה ביותר בקובץ מבין השורות שאינן מכילות ספרות כלל.

בשורה האחרונה של הפלט התוכנית מדפיסה את רשימת שמות כל הקבצים שאורך השורה הארוכה ביותר שלהם מבין השורות שאינן מכילות ספרות הוא הגדול ביותר .

על סדר הקבצים בפלט (כולל בשורה האחרונה) להיות לפי סדר הופעתם ברשימת הפרמטרים .

```
#!/bin/awk -f
{
x=gsub("[0-9]","11",$0)
  if(x==0 && length($0)>=B[FILENAME])
  {
     B[FILENAME]=length($0)
  }
}
END{
for (z=1;z<length(ARGV);z++){
     if(B[ARGV[z]]=="") B[ARGV[z]]=0
     print ARGV[z],B[ARGV[z]]
     }
max=B[ARGV[1]]
for (z=1;z<length(ARGV);z++){
```

```
        if(B[ARGV[z]]>max) max=B[ARGV[z]]
}
ORS=" "
for (z=1;z<length(ARGV);z++){
        if(B[ARGV[z]]==max) print ARGV[z]
}
ORS="\n"
print ""
}
```

נגדיר שמטריצה ריבועית היא שוות עמודות או שורות אם סכום המספרים בכל עמודה זהה או סכום המספרים בכל שורה זהה .

כתוב/כתבי תוכנית ב - awk בשם P10.2, שמקבלת כפרמטרים רשימת שמות של קבצים שמכילים מטריצות ריבועיות (לא בהכרח כולן באותו גודל .)

התוכנית מדפיסה עבור כל קובץ ששמו הועבר לתוכנית, שורה המכילה את שם הקובץ לאחר מכן תו רווח יחיד ולאחר מכן YES אם הקובץ מכיל מטריצה שוות עמודות או שורות, או NO אם הקובץ אינו מכיל מטריצה שוות עמודות או שורות ב . שורה האחרונה יודפסו שמות הקבצים שמכילים מטריצה שוות עמודות או שורות מופרדים על ידי תו רווח אחד בדיוק .

על סדר הקבצים בפלט (כולל בשורה האחרונה) להיות לפי סדר הופעתם ברשימת הפרמטרים .

/הנח הניחי שהקצבים ששמם מועבר כפרמטר קיימים ותוכנם חוקי (דהינו מכילים מטריצות ריבועיות

```
#!/bin/awk -f
{
for(i=1;i<=NF;i++)
  X[FILENAME,FNR,i]=$i
R[FILENAME]=FNR
C[FILENAME]=NF
}
function checkRow(f1){
```

```
s=0
for(i=1;i<=R[f1];i++)
 {
   s=s+X[f1,1,i]
 }
for(j=2;j<=R[f1];j++){
s1=0
for(i=1;i<=R[f1];i++){
 s1=s1+X[f1,j,i];
 }
if(s!=s1){return "NO" }
}
       return "YES"
}
function checkCol(f1){
s=0
for(i=1;i<=R[f1];i++)
 {
   s=s+X[f1,i,1]
 }
for(j=2;j<=R[f1];j++){
s1=0
for(i=1;i<=R[f1];i++){
 s1=s1+X[f1,i,j];
 }
if(s!=s1){return "NO" }
}
return "YES"
}
```

```
END{
d=1
for(z=1;z<length(ARGV);z++){
if( checkRow(ARGV[z])=="YES" || checkCol(ARGV[z])=="YES")
 {
 print ARGV[z],"YES"
 T[d]=ARGV[z]
 d++
 }
 else
  {
      print ARGV[z],"NO"
  }
 }
#ORS=" "
for(p=1;p<d;p++)
 {
  printf T[p]" "
 }
#ORS="\n"
print ""
}
```

בחברה להשכרת מכוניות שומרים את הנתונים באופן הבא :

לכל דגם מכוניות שהחברה משכירה ישנו קובץ ששמו כשם הדגם (לדוגמה: ניסן-אלמרה ,) כאשר

כל שורה בקובץ מתארת השכרת מכונית מהדגם ומכילה

את תאריך ההשכרה, מספר הרכב ושם השוכר (בפורמט כפי שמתואר בדוגמה

שבהמשך).

כתוב/כתבי תוכנית ב - awk בשם P10.3 שמקבלת כפרמטרים רשימת שמות של דגמי רכבים

(לכל דגם ברשימה ישנו קובץ בתיקיה בה מופעלת התוכנית ששמו כשם הדגם). התוכנית מדפיסה לפלט עבור כל דגם ברשימה שורה שמכילה את שם הדגם, לאחר מכן תו רווח, לאחר מכן מספר המציין את מספר ההשכרות הגדול ביותר בחודש מדגם זה, ולאחר מכן את החודש/חודשים שהיו בהם מספר השכרות גדול ביותר מדגם זה (, אם ישנם מספר חודשים כאלה הם יודפסו כאשר רווח אחד מפריד בין החודשים וסדר החודשים הוא לפי הסדר בשנה, זאת אומרת ינואר, פברואר, מרץ וכו ,)...' לאחר מכן תו רווח, ולבסוף מספר המציין את סך כל ההשכרות שהיו עבור מכוניות מדגם זה .

על סדר הדגמים בפלט להיות לפי סדר הופעתם כפרמטרים לתוכנית .

```
#!/bin/awk -f
BEGIN{
}
{
for(i=1;i<length(ARGV);i++)
    {
    model=ARGV[i]
    while(getline <model){
    if($2 == "January")
    Ct[model,1]++
    if($2 == "February")
    Ct[model,2]++
    if($2 == "March")
    Ct[model,3]++
    if($2 == "April")
    Ct[model,4]++
    if($2 == "May")
    Ct[model,5]++
    if($2 == "June")
    Ct[model,6]++
    if($2 == "July")
```

```
        Ct[model,7]++
        if($2 == "August")
        Ct[model,8]++
        if($2 == "September")
        Ct[model,9]++
        if($2 == "October")
        Ct[model,10]++
        if($2 == "November")
        Ct[model,11]++
        if($2 == "December")
        Ct[model,12]++
        }
        }
}
END{
for(i=1;i<length(ARGV);i++)
{
        Max[ARGV[i]]=0
        for(j=1;j<=12;j++){
        MaxT[ARGV[i]]=MaxT[ARGV[i]]+Ct[ARGV[i],j]
        if(Max[ARGV[i]]<=Ct[ARGV[i],j])
                Max[ARGV[i]]=Ct[ARGV[i],j]
        }
}
for(i=1;i<length(ARGV);i++)
{
printf ARGV[i]" "
        for(j=1;j<=12;j++){
        if(Max[ARGV[i]]==Ct[ARGV[i],j]){
```

```
if(j==1) printf "January "
if(j==2) printf "Febuary "
if(j==3) printf "March "
if(j==4) printf "April "
if(j==5) printf "May "
if(j==6) printf "June "
if(j==7) printf "July "
if(j==8) printf "August "
if(j==9) printf "September "
if(j==10) printf "October "
if(j==11) printf "November "
if(j==12) printf "December "
}
}
printf MaxT[ARGV[i]]" "
print ""
}
}
```

הגדרה: נגדיר את הפרש האלכסונים של מטריצה ריבועית כסכום האיברים שנמצאים באלכסון הראשי של המטריצה פחות סכום האיברים שנמצאים באלכסון המשני של המטריצה .

כתוב/כתבי תוכנית ב - awk בשם P10.4 שמקבלת כפרמטרים רשימת שמות של קבצים שמכילים מטריצות ריבועיות ומדפיסה לפלט שורה אחת עבור כל קובץ שמכילה את שם הקובץ, לאחריו תו רווח יחיד ולאחריו את הפרש האלכסונים של המטריצה שהקובץ מכיל . בשורה האחרונה יופיע שמות הקבצים שהפרש האלכסונים שלהם הוא הקטן ביו . תר על סדר הקבצים בפלט (כולל בשורה האחרונה) להיות לפי סדר הופעתם ברשימת הפרמטרים

```
#!/bin/awk -f
{
Ct[FILENAME]=Ct[FILENAME]+$FNR-$(NF-FNR+1)
```

```
}
END{
for(i=1;i<length(ARGV);i++)
 {
  print ARGV[i],Ct[ARGV[i]]
 if(min>Ct[ARGV[i]]){min=Ct[ARGV[i]]}
 }
for(i=1;i<length(ARGV);i++)
      if(min==Ct[ARGV[i]])
            printf ARGV[i]" "
print ""
}
```

כתוב תוכנית Script ב - sed בשם P0.5 שמקבלת כפרמטרים שם קובץ (בהמשך נקרא לו קובץ) 1 ומחרוזת (בהמשך נקרא לה מחר). 1וזת

התוכנית מדפיסה לפלט את השורות בקובץ ש 1 מכילות את מחרוזת (1 באיזשהו מקום בשורה) כאשר בשורות אלה מתבצע שיכפול של המילה השניי , ה כפי שמודגם בדוגמה שלהלן .

(ניתן להניח שבכל שורה יש לפחות שתי מילים).

```
sed -n '/'$2'/p' $1 >| tmp
sed 's/[^ ]\+/& &/2' tmp
```

מטלה 11:

כתוב/כתבי תוכנית ב - awk ב שם P11.5 שמקבלת כפרמטרים רשימת מספרים (בהמשך נקרא לה רשימה) 1 לאחריה המחרוזת files- ולאחריה רשימת קבצים (בהמשך נקרא לה רשימה)2

ומדפיסה שורה אחת עבור כל קובץ שמכילה את שם הקובץ, לאחריו תו רווח אחד ולאחריו מספר שמציין את סכום המספרים שבקובץ שנמצאים ברשימה . 1 ניתן להניח שהקבצים ברשימה 2 מכילים מספרים בלבד . על סדר הקבצים בפלט להיות לפי סדר הופעתם ברשימה .

להלן שמות קבצים ותוכנם .

A2:

10 30

30 40

B2:

2 3 4

50 4

40 25

C2:

20 200

60 3 70

3

400

K2:

100

200

150

4

E2:

10 20 30 40 30

לאחר הפעלת התוכנית ע"י הפקודה:

P11.5 30 4 10 -files A2 B2 C2 K2 E2

יתקבל הפלט :

A2 70

B2 8

C2 0

K2 4

E2 70

פתרון:

```
#!/bin/awk -f
BEGIN {
```

```
 x=1
 while ( ARGV[x]!="-files" ) {
 NUMS[x]=ARGV[x]
 delete ARGV[x]
 x++
 }
begin_file_list=x+1
delete ARGV[x]
z=ARGV[begin_file_list]
}
{
if(z!=FILENAME){
       print z,ct
       ct=0
       z=FILENAME
       }
for(j=1;j<=NF;j++)
 for(i=1;i<=length(NUMS);i++)
  if( $j==NUMS[i] )
   ct=ct+$j
}
END{
print FILENAME,ct
}
```

דוגמאות של אודי מתגבורים:

כתוב/כתבי תוכנית סקריפט ב - bash בשם P1 שמקבלת כפרמטרים מספר (בהמשך נקרא לו i) ולאחריו רשימת מילים (בהמשך נקרא לה רשימה)1

ומדפיסה לפלט את כל המילים ברשימה 1 שמכילות i חזרות של איזשהו תו

(וללא תווים נוספים) .

לדוגמה, לאחר הפעלת התוכנית על ידי הפקודה :

```
P1 5 aaaaa aaaaaa aaaa bbbbb aaaaac
```

יתקבל הפלט :

```
aaaaa
bbbbb
```

פתרון:

```
y=$[$1-1]
shift
for x in $*
 do
 if [ $(echo $x | egrep -c "^(.)\1{$y}$") -gt 0 ]
 then
 echo $x
 fi
 done
```

כתוב/כתבי תוכנית סקריפט ב - bash בשם P2 שמקבלת כפרמטרים רשימת

מילים (בהמשך נקרא לה רשימה1) ומדפיסה לפלט את כל המילים ברשימה 1

שמורכבות מאוסף תווים שחוזרים בדיוק פעמיים .

לדוגמה, לאחר הפעלת התוכנית על ידי הפקודה :

```
P2 aabbcc abbcca abdabd aaaa aaabbb
```

יתקבל הפלט :

```
aabbcc
abbcca
abdabd
```

פתרון :

```
for x in *$
 do
 echo $x | egrep -o . | sort | uniq -c >| tmp
 flag="OK "
 while read y
 do
 num=$(echo $y | cut -d" " -f1)
 if [ $num -ne 2 ]
 then
 flag="NOT_OK "
 break
 fi
 done<tmp
 if [ $flag = OK ]
 then
 echo $x
 fi
 done
```

כתוב/כתבי תוכנית סקריפט ב - bash בשם P3 שמקבלת כפרמטרים מחרוזת (בהמשך נקרא לה מחרוזת) 1 ותיקיה (בהמשך נקרא לה תיקיה) 1 ומדפיסה לפלט את כל שמות הקבצים (הסופיים, דהינו בלי המסלול אליהם) שנמצאים בתיקיה (1 בעומק כלשהו) ואינם מכילים את מחרוזת . 1 על כל שם קובץ בפלט להופיע בפעם אחת בלבד ובשורה נפרדת. על סדר שמות הקבצים להיות לפי סדר לכסיקוגרפי עולה (בהתאם לפקודה sort ללא אופציות).

לדוגמה, לאחר הפעלת התוכנית על ידי הפקודה :

```
P3 ab ~basicsys/win14/ex7/d2
```

יתקבל הפלט :

A

B

D

E

G

S

הסבר לפלט: השם A מופיע בפלט כי קיים קובץ שהשם הסופי שלו הוא A

שנמצא בתיקיה ~basicsys/win14/ex7/d2 שאינו מכיל את המחרוזת ab . וכן הלאה לגבי שאר השמות שמופיעים בפלט.

פתרון

```
echo -n "" >|tmp1
echo -n "" >|tmp2
string=$1
dir=$2
find $dir -type f >| tmp
while read file
 do
 if [ $(egrep -c "$string" $file) -eq 0 ]
 then
 echo $file >> tmp1
 fi
 done<tmp
```

כתוב/כתבי תוכנית סקריפט ב - bash בשם P4 שמקבלת כפרמטרים מחרוזת

(בהמשך נקרא לה מחרוזת) ושם קובץ מדפיסה את מספר הופעות המחרוזת כמילה בקובץ.

פתרון:

```
string=$1
file=$2
echo -n "" >|tmp3
count=0
for x in $(echo $(cat $file))
 do
 echo $x >> tmp3
 done
egrep -c "^$string$" tmp3
```

כתוב/כתבי תוכנית סקריפט ב - bash בשם P5 שמקבלת כפרמטרים מחרוזת מספר יקרא i ושם תיקיה הסקריפט יחזיר לפלט את מספר הקבצים שעומק כלשהו בתיקיה שהמחרוזת מופיעה בהם i פעמים.

```
string=$1
num=$2
dir=$3
find $dir -type f >| tmp4
count=0
for file in $(echo $(cat tmp4))
 do
 if [ $(P4 $string $file) -eq $num ]
 then
 count=$[$count+1]
 fi
 done
echo $count
```

דוגמא מעולה בפתרון לאיך לחתוך את הקלט הפרמטרים לכמה רשימות לפי מילה מפרידה כלשהי:

כתוב/כתבי תוכנית סקריפט ב - bash בשם P6 שמקבלת כפרמטרים רשימת מחרוזות בהמשך נקרא לה רשימה1 לאחר מכן את המחרוזת nums– לאחריה רשימת מספרים רשימה2 לאחר מכן dirs– ורשימת תיקיות רשימה3.

ומדפיסה לפלט שורה אחת עבור כל צרוף של מחרוזת מרשימה 1, מספר מרשימה 2 ותיקיה מרשימה 3 שמכילה את המחרוזת המספר והתיקייה כשבינם מפריד תו רווח לאחריו מספר שמציין את מספר הקבצים ביקיה שמספר ההופעות בהם של המחרוזת כמילה שווה למספר.

לאחר הפעלת:

```
P6 ab abc -nums 1 2 5 -dirs ~basicsys/win14/ex7
```

יתקבל הפלט :

```
ab 1 /home/cs/segel/basicsys/win14/ex7 2
ab 2 /home/cs/segel/basicsys/win14/ex7 0
ab 5 /home/cs/segel/basicsys/win14/ex7 1
abc 1 /home/cs/segel/basicsys/win14/ex7 7
abc 2 /home/cs/segel/basicsys/win14/ex7 0
abc 5 /home/cs/segel/basicsys/win14/ex7 0
```

הסבר לפלט: השורה הראשונה מתייחסת לצרוף: מחרוזת ab מספר 1, ותיקיה
/home/cs/segel/basicsys/win14/ex7 המספר 2 בסוף השורה הראשונה מציין שיש בדיוק שני קבצים בתיקיה שהמחרוזת ab מופיעה בהם כמילה בדיוק פעם אחת.

<u>פתרון:</u>

```
echo -n "" >| strings
echo -n "" >| nums
echo -n "" >| dirs

for x in $*
 do
 if [ $x = -nums ]
 then
 shift
 break
 fi
```

```
 echo $x >> strings
 shift
 done

for x in $*
 do
 if [ $x = -dirs ]
 then
 shift
 break
 fi
 echo $x >> nums
 shift
 done

for x in $*
 do
 echo $x >> dirs
 done

for str in $(cat strings)
 do
 for dir in $(cat dirs)
 do
 for num in $(cat nums)
 do
 count=$(P5 $str $num $dir)
 echo $str $num $dir $count
 done
```

```
done
done
```

כתוב/כתבי תוכנית ב - awk בשם P1 שמקבלת כפרמטרים מספר (בהמשך נקרא לו i) ולאחריו שם קובץ שמכיל מטריצה (לא בהכרח ריבועית ולא בהכרח בעלת מספר זהה של מספרים בכל שורה) מדפיסה לפלט את סכום המספרים בשורה ה - i של המטריצה .

לדוגמה, נניח שתוכן הקובץ F1 הוא :

```
6 5 4 3 2 1
4 5 6
2 8
13 17 16
```

לאחר הפעלת התוכנית על ידי הפקודה : P1 2 F1 יתקבל הפלט :

15

<u>פתרון:</u>

```
#!/bin/awk -f
BEGIN { row=ARGV[1]; delete ARGV[1] }
{
 for (i=1; i<= NF; i++) {
 A[NR,i]=$i
 }
 B[NR]=NF
}
END {
 for (j=1; j<= B[row]; j++) {
 s=s+A[row,j]
 }
 print s
```

}

כתוב/כתבי תוכנית ב - awk בשם P2 שמקבלת כפרמטרים מספר (בהמשך נקרא לו i) ולאחריו רשימת שמות קבצים שמכילים מטריצות (לא בהכרח ריבועיות ולא בהכרח בעלות מספר זהה של מספרים בכל שורה) ומדפיסה לפלט שורה אחת עבור כל קובץ שמכילה את שם הקובץ, לאחר מכן התו : לאחר מכן תו רווח אחד ולאחר מכן את סכום המספרים בשורה ה - i של המטריצה שהקובץ מכיל. על סדר הקבצים בפלט להיות לפי סדר הופעתם ברשימת הפרמטרים .

לדוגמה, נניח שתוכן הקובץ F1 הוא כמו בדוגמה , 1 תוכן הקובץ F2 הוא :

8 4 5 6

2 3 100

8 6

תוכן הקובץ F3 הוא :

40 18

26 22

8 7

הפלט לפקודה:

P2 2 F1 F3 F2

הוא:

F1: 15

F3: 48

F2: 105

<u>פתרון:</u>

```
#!/bin/awk -f
BEGIN { row=ARGV[1]; delete ARGV[1] }
{
 for (i=1; i<= NF; i++) {
```

```
 A[FILENAME,FNR,i]=$i
 }
 B[FILENAME,FNR]=NF
}
END {
for (x=2; x < length(ARGV); x++) {
 s=0
 for (j=1; j<= B[ARGV[x],row]; j++) {
 s=s+A[ARGV[x],row,j]
 }
 print ARGV[x] ": " s
 } }
```

כתוב תוכנית בשם sed P5 שמקבלת כפרמטר שם קובץ התוכנית מדפיסה לפלט את השורות בקובץ שמכילות לפחות שתי ספרותף כאשר בשורות האלה מתבצעת החלפה בין המילה הראשונה למילה האחרונה.

דוגמא:

```
F6:
one two three
one1 two three
one2 two3 three
12345
11 22 33 444
```

לאחר הפעלת התוכנית P5 F6 יתקבל הפלט:

```
three two3 one2
12345
444 22 33 11
```

<u>פתרון</u>

```
sed '/[0-9].*[0-9]/!d' $1 >|tmp
sed 's/^\([ ]*\)\([^ ]\+\)\(.*\)\([ ]\+\)\([^
]\+\)\([ ]*\)$/\1\5\3\4\2\6/' < tmp
```

תוכנית awk בשם P4 משקבלת כפרמטרים רשימת מספרים (רשימה 1) ואלחריה רשימת שמות קבצים שמיכלים מטריצות ריבעיות(רשימה 2) ומדפיסה לפלט שורה אחת עבור כל קובץ שמכילה את שם הקובץ לאחר מכן רווח ולאחר מכן מספר שמציין את סכום העמודות של הקובץ עבור העמודות שמופיעות ברשימת המספרים (אם מופיעה עמודה שלא נמצאת בקובץ אז העמודה הזאת לא נלקחת בחישוב סכום העמודות בקובץ). יש להניח שכל שם קובץ שנמצא ברשימה1 מכיל תו אחד לפחות שאינו ספרה. על סדר השורות בפלט להיות לפי סדר הקבצים ברשימה 1.

לדוגמא:

F4:

10 20 30 40

1 2 3 4

-1 -2 -3 -4

1 2 3 4

G:

100

H:

6 2 3

8 4 1

1 2 5

F5:

6 2

8 4

הפלט לאחר הפקודה:

P4 1 3 10 F4 G H F5

הוא:

F4 44

G 100

H 24

F5 14

פתרון:

```
#!/bin/awk -f
BEGIN { end_file_list=length(ARGV)-1
 for (x=1; x <= end_file_list; x++) {
 s=ARGV[x]
 if (gsub("[^0-9]","&",s)>0){
 begin_file_list=x
 break
 }
 NUMS[x]=ARGV[x]
 delete ARGV[x]
 }
}
{
 for (i=1; i<= NF; i++) {
 A[FILENAME,FNR,i]=$i
 }
 B[FILENAME]=FNR
}
END {
for (x=begin_file_list; x <= end_file_list; x++)
{
 s=calc_file(ARGV[x])
 print ARGV[x],s
 }
}
```

```
function calc_file(file) {
 sum=0
 for (n in NUMS) {
 sum = sum + sum_col(file,NUMS[n])
 }
 return sum
}
function sum_col(file,n) {
 if (n > B[file]) {return 0}
 sum1=0
 for (j=1;j<=B[file];j++){
 sum1=sum1+A[file,j,n]
 }
 return sum1
}
```

כתוב/כתבי תוכנית ב- awk בשם P2 שמקבלת פרמטר אחד שהינו שם קובץ (בהמשך נקרא לו קובץ 1). הנח/י שהתוכן של קובץ 1 הוא מטריצה של מספרים. המספרים בכל שורה בקובץ 1 מופרדים ע"י תו רווח אחד או יותר. התוכנית תדפיס Yes אם מתקיים **<u>לפחות אחד</u>** משני התנאים הבאים:

(1) סכום המספרים בכל שורה בקובץ הינו קבוע,

(2) סכום המספרים בכל עמודה בקובץ הינו קבוע,אחרת התוכנית תדפיס No.

לדוגמה, נניח שתוכן הקובץ F הוא:

10 30 40

14 26 40

ותוכן הקובץ F2 הוא:

11 9 4 2

1 3 8 10

1 1 1 1

לאחר הפעלת התוכנית ע"י הפקודה: >awk -f P2 F2

יתקבל הפלט:
Yes

<u>הפתרון:</u>

```
{
 for (i=1;i<=NF;i++) {
  col[i]=col[i]+$i
  row[NR]=row[NR]+$i
 }
}
END {
    for (i=1;i<NF;i++)
     if ( col[i]!=col[i+1]){
       for (i=1;i<NR;i++)
         if (row[i]!=row[i+1]) {
           print "No"
           exit
         }
       break
    }
   print "Yes"
 }
```