הרצאה מספר 1

אתר הקורס

אתר הקורס נמצא בכתובת הבאה:

http://mars.netanya.ac.il/~basicsys

הוספה לרשימת תפוצה של הקורס

כדי שתוכלו לקבל הודעות של הקורס למייל שלכם עליכם
לשלוח מייל לכתובת **rotics6@gmail.com**
ולכתוב בנושא ההודעה: הוספה לרשימת תפוצה.

מטרת הקורס

מטרת הקורס להקנות לתלמיד ידע כמשתמש בסביבת **unix**.לצורך כך נלמד לעומק את הפקודות של **Bash** שהיא תוכנית ה- **shell** בשרת המכללה. מערכת ההפעלה של שרת המכללה היא **Linux** (שזהו סוג של מערכת ההפעלה **unix**).

התחברות לשרת המכללה

כדי להתחבר לשרת המכללה יש שתי אפשרויות.

אפשרות 1: באמצעות התקנת putty

מחפשים ב- google את השם putty ומגיעים לאתר בכתובת הבאה:

http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

המסך שמתקבל נראה כך:

PuTTY Download Page

Home | Licence | FAQ | Docs | **Download** | Keys | Links
Mirrors | Updates | Feedback | Changes | Wishlist | Team

Here are the PuTTY files themselves:

- PuTTY (the Telnet and SSH client itself)
- PSCP (an SCP client, i.e. command-line secure file copy)
- PSFTP (an SFTP client, i.e. general file transfer sessions much like FTP)
- PuTTYtel (a Telnet-only client)
- Plink (a command-line interface to the PuTTY back ends)
- Pageant (an SSH authentication agent for PuTTY, PSCP, PSFTP, and Plink)
- PuTTYgen (an RSA and DSA key generation utility).

LEGAL WARNING: Use of PuTTY, PSCP, PSFTP and Plink is illegal in countries where encryption is outlawed. I believe it is legal to use PuTTY, PSCP, PSFTP and Plink in England and Wales and in many other countries, but I am not a lawyer and so if in doubt you should seek legal advice before downloading it. You may find this site useful (it's a survey of cryptography laws in many countries) but I can't vouch for its correctness.

Use of the Telnet-only binary (PuTTYtel) is unrestricted by any cryptography laws.

There are cryptographic signatures available for all the files we offer below. We also supply cryptographically signed lists of checksums. To download our public keys and find out more about our signature policy, visit the Keys page. If you need a Windows program to compute MD5 checksums, you could try the one at this site. (This MD5 program is also cryptographically signed by its author.)

Binaries

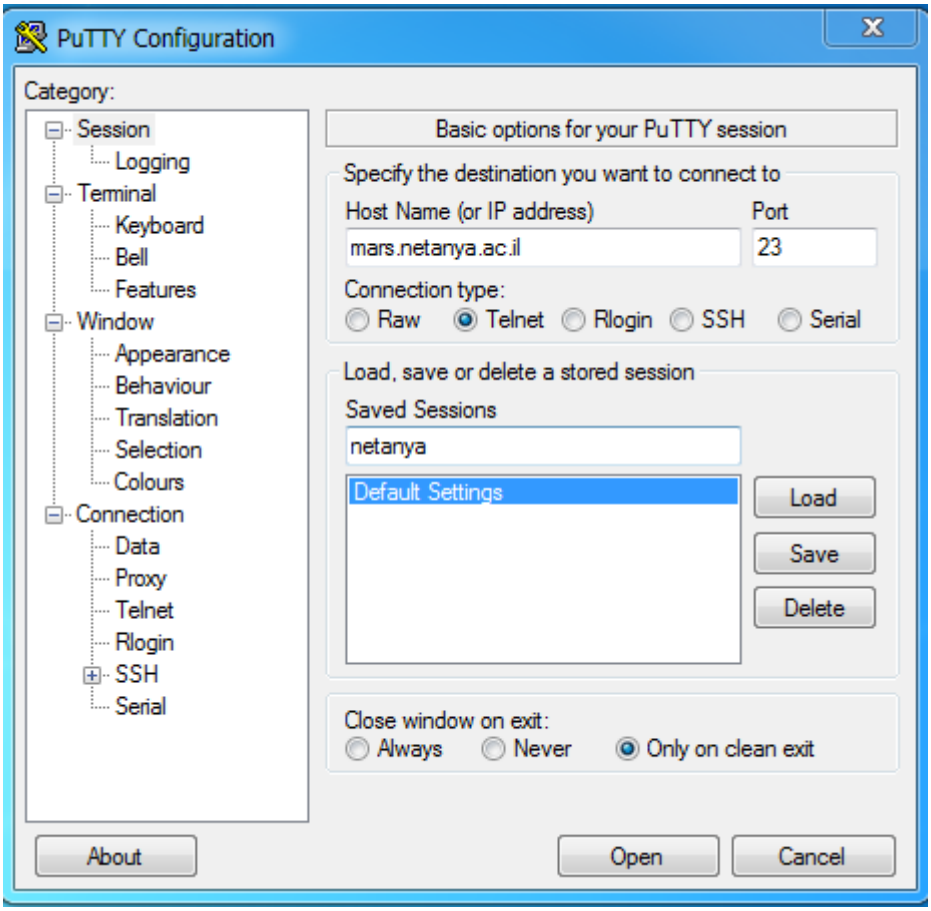
The latest release version (beta 0.63). This will generally be a version I think is reasonably likely to work well. If you have a problem with the release version, it might be worth trying out the latest development snapshot (below) to see if I've already fixed the bug, before reporting it to me.

For Windows on Intel x86

PuTTY:	putty.exe	(or by FTP)	(RSA sig)	(DSA sig)
PuTTYtel:	puttytel.exe	(or by FTP)	(RSA sig)	(DSA sig)
PSCP:	pscp.exe	(or by FTP)	(RSA sig)	(DSA sig)
PSFTP:	psftp.exe	(or by FTP)	(RSA sig)	(DSA sig)
Plink:	plink.exe	(or by FTP)	(RSA sig)	(DSA sig)
Pageant:	pageant.exe	(or by FTP)	(RSA sig)	(DSA sig)
PuTTYgen:	puttygen.exe	(or by FTP)	(RSA sig)	(DSA sig)

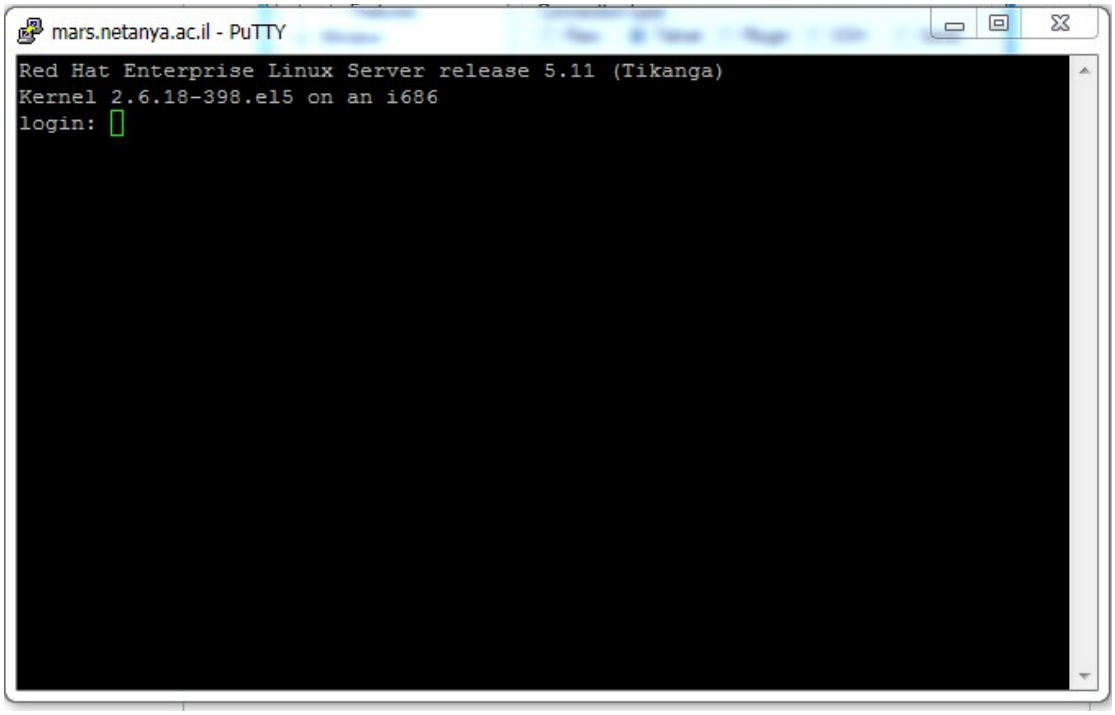
במסך הנ"ל יש ללחוץ על המקש הימני של העכבר (כאשר העכבר נמצא על המקום שמסומן) ולהוריד את putty.exe לשולחן העבודה. אפשר גם על ידי לחיצה כפולה להפעיל את putty.exe אבל אז כל פעם שנרצה להפעיל את putty נצטרך להכנס לאתר הנ"ל.

כעת מפעילים את putty ומקבלים את המסך הבא:



כפי שמוצג במסך הנ"ל יש לרשום mars.netanya.ac.il במקום של ה- host name ויש לדאוג שהכפתור של ה- telnet יהיה דלוק.

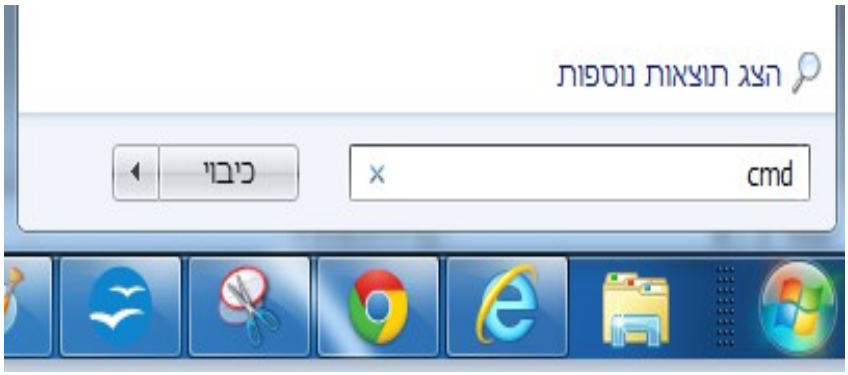
לאחר מכן יש ללחוץ על open ומקבלים את המסך הבא:



במסך הנ"ל מקלידים שם משתמש וסיסמה ונכנסים לחשבון שלכם בשרת של המכללה.

אפשרות 2: באמצעות telnet

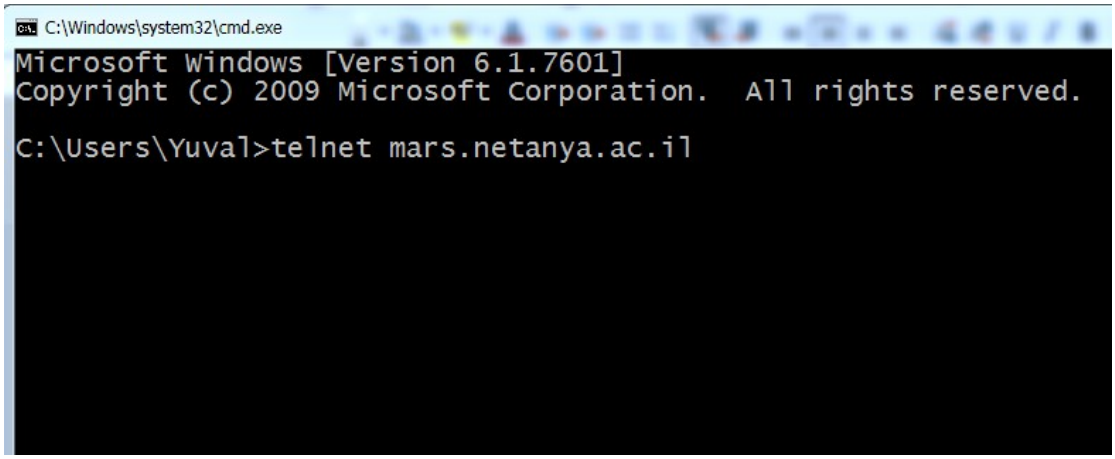
לוחצים על התחל ואז רושמים cmd כפי שמודגם במסך הבא:



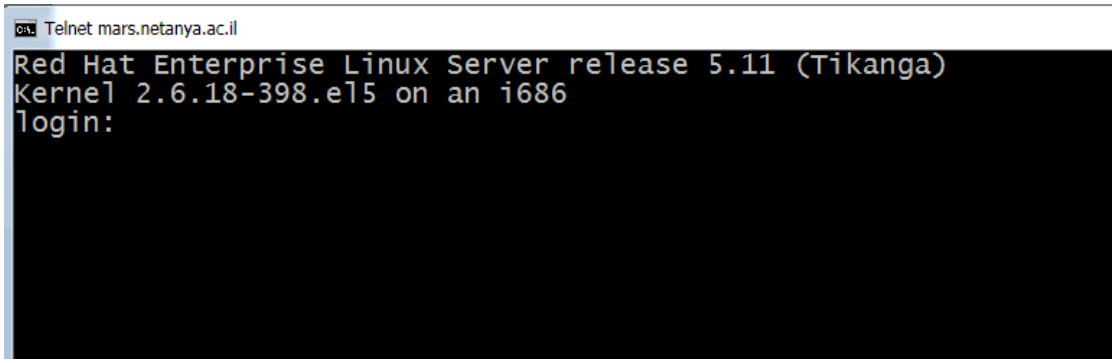
ואז מופיע המסך הבא שבו רושמים:

```
telnet mars.netanya.ac.il
```

ולוחצים על מקש enter (מעכשיו והלאה לא נציין יותר לחיצה על מקש enter).



לאחר הלחיצה על enter (במידה ולא מופיעה שגיאה) יופיע המסך הבא:



במסך הנ"ל מקלידים שם משתמש וסיסמה ונכנסים לחשבון שלכם בשרת של המכללה.

במידה ומופיעה הודעת שגיאה מהסוג:

telnet is not recognized as an internal or external command...

זה בגלל ש- windows 7/8 לא תומך בהפעלת telnet באופן אוטומטי. לפתרון הבעיה מבצעים את ההוראות הבאות:

•Click Start > Control Panel.

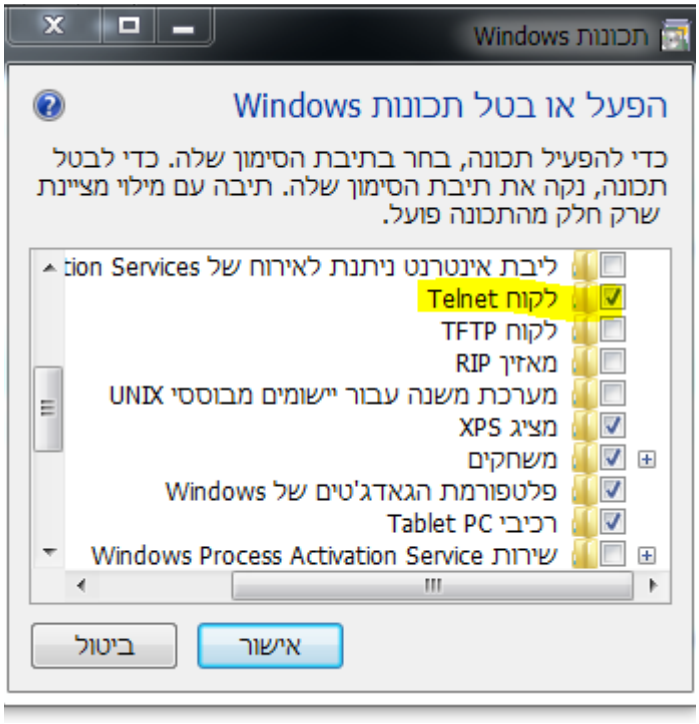
•Click Programs and Features.

•Click Turn Windows features on or off.

•In the Windows Features dialog box, check the Telnet Client check box.

•Click OK. The system installs the appropriate files. This will take a few seconds to a minute.

כשפועלים לפי ההוראות הנ"ל מגיעים למסך הבא,
מדליקים בו את הכפתור של לקוח telnet, ולוחצים על אישור.



לאחר מכן פועלים שוב כפי שתואר בתחילת אפשרות 2.

תיקית הבית

כאשר המשתמש נכנס לחשבון שלו בשרת המערכת מכניסה אותו לתיקיה שנקראת תיקית הבית שלו. לכל משתמש תיקית בית שונה.

הפקודה pwd

הפקודה pwd מציגה את המסלול המלא לתיקיה שבו מופעלת הפקודה.
לדוגמה, לאחר הפעלת הפקודה pwd בתיקית הבית של חשבון הקורס מופיע הפלט הבא:

```
basicsys@mars~>pwd
```

```
/home/cs/segel/basicsys
```

משמעות הפלט היא שתיקית הבית של חשבון הקורס נקראת basicsys ותיקיה זו נמצאת בתוך תיקיה segl שנמצאת בתוך תיקיה cs שנמצאת בתוך תיקיה home שנמצאת בתוך תיקיה /. התיקיה / נקראת התיקיה הראשית (root directory) ובתוכה נמצאים כל התיקיות והקבצים שבשרת.

הפקודה cd

הפקודה: cd <שם תיקיה> גורמת למעבר לתיקיה שהועברה כפרמטר.
הפעלת הפקודת cd ללא פרמטרים גורמת למעבר לתיקית הבית של המשתמש.

לדוגמה:

```
basicsys@mars~>cd lec1

basicsys@mars~/lec1>pwd
/home/cs/segel/basicsys/lec1

basicsys@mars~/lec1>cd

basicsys@mars~>pwd
/home/cs/segel/basicsys
```

תיקה נוכחית

בהמשך נגדיר תיקיה נוכחית כתיקיה שבה מופעלות הפקודות. במערכת תיקיה זו מסומנת על ידי . (נקודה).

לכן הפקודה:

```
cd .
```

לא מבצעת כלום.

במערכת מסמנים ב- .. תקיה אחת מעל התיקיה הנוכחית.

לדוגמה:

```
basicsys@mars~>cd lec1

basicsys@mars~/lec1>pwd
```

```
/home/cs/segel/basicsys/lec1

basicsys@mars~/lec1>cd ..
basicsys@mars~>pwd
/home/cs/segel/basicsys
```

הפקודה ls

הפקודה ls (ללא פרמטרים) מראה את שמות הקבצים שנמצאים בתיקיה בה מופעלת הפקודה. לדוגמה:

```
basicsys@mars~/lec1>ls
F1  F2  last-year  lec1-2014.txt
```

הפקודה cat

הפקודה <שם קובץ> cat מציגה על המסך את תוכן הקובץ.
אופציות: n- הוסף מספרי שורה לקובץ
s- צמצם רצף של שורות רווח לשורת רווח אחת

לדוגמה:

```
basicsys@mars~/lec1>cat F1
abc


def




ghe

basicsys@mars~/lec1>cat -n F1
     1  abc
     2
     3
     4  def
     5
     6
     7
     8
     9  ghe
```

```
basicsys@mars~/lec1>cat -s F1
abc

def

ghe

basicsys@mars~/lec1>cat -s -n F1
     1  abc
     2
     3  def
     4
     5  ghe
```

הערה כללית לגבי פרמטרים לפקודות ב- bash

בדרך כלל אין חשיבות לסדר בין הפרמטרים בפקודות bash. בנוסף לכך, כאשר יש כמה פרמטרים כמו -s -n אפשר לרשום אותם בקיצור בצורה sn-

לדוגמה, כל 4 הפקודות הבאות שקולות.

```
cat -s -n F1

cat -n -s F1

cat -ns F1

cat -sn F1
```

הערה כללית לגבי קבלת עזרה לשימוש פקודות ב- bash

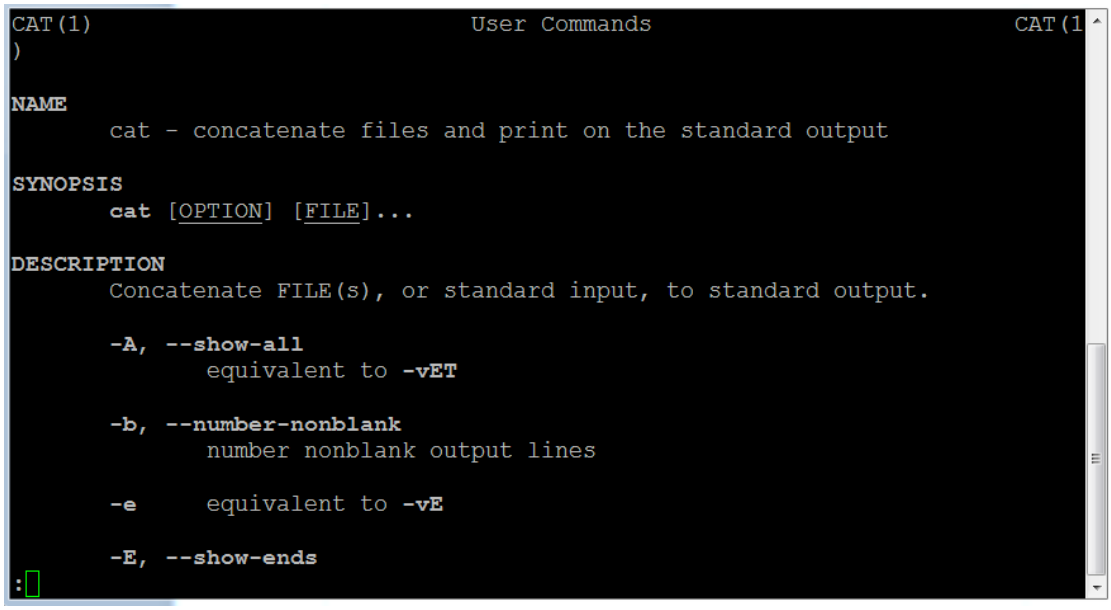
כדי לקבל עזרה לגבי שימוש של פקודות ב- bash אפשר לרשום

```
man <פקודה>
```

לדוגמה כשרושמים:

```
man cat
```

מתקבל המסך הבא:

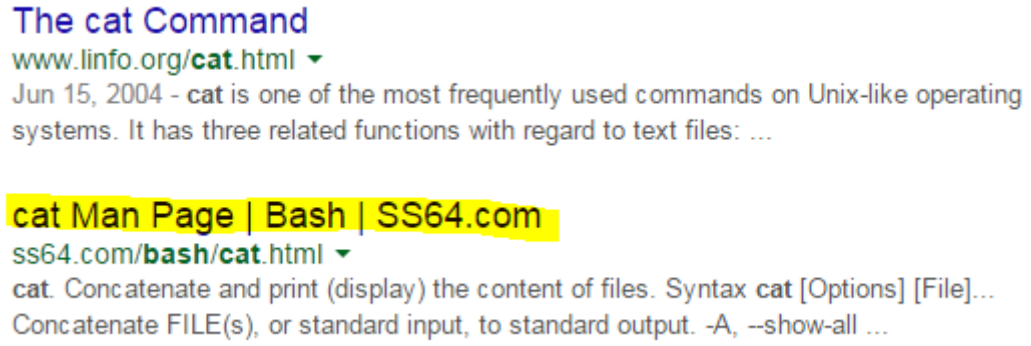


במסך זה יש הסברים על הפקודה ועל האופציות שלה.
כדי להתקדם קדימה לדף הבא יש ללחוץ על מקש רווח.
כדי לצאת מהמסך יש להקליד q

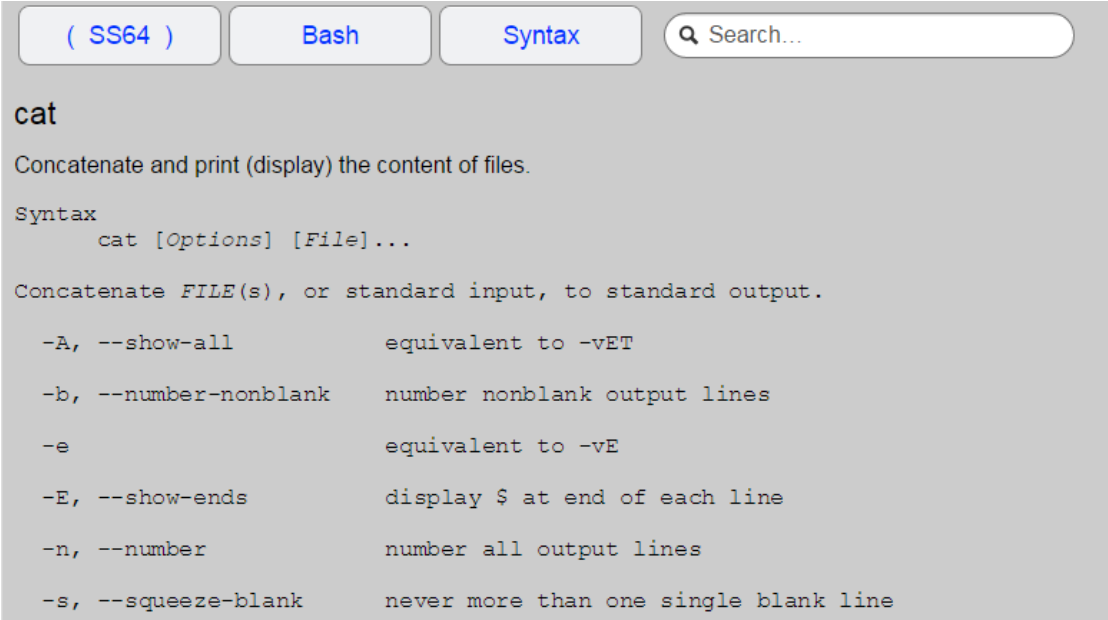
אפשרות נוספת (מומלצת) היא לחפש ב- google
bash <פקודה>
לדוגמה, לאחר חיפוש ב- google של:

bash cat

מקבלים:



נכנסים ללינק השני (זה שמסומן) ומגיעים לדף הבא שמסביר על הפקודה cat:



(SS64) Bash Syntax Search...

cat

Concatenate and print (display) the content of files.

```
Syntax
      cat [Options] [File]...

Concatenate FILE(s), or standard input, to standard output.

  -A, --show-all           equivalent to -vET

  -b, --number-nonblank    number nonblank output lines

  -e                       equivalent to -vE

  -E, --show-ends          display $ at end of each line

  -n, --number             number all output lines

  -s, --squeeze-blank      never more than one single blank line
```

שימוש בעורך pico

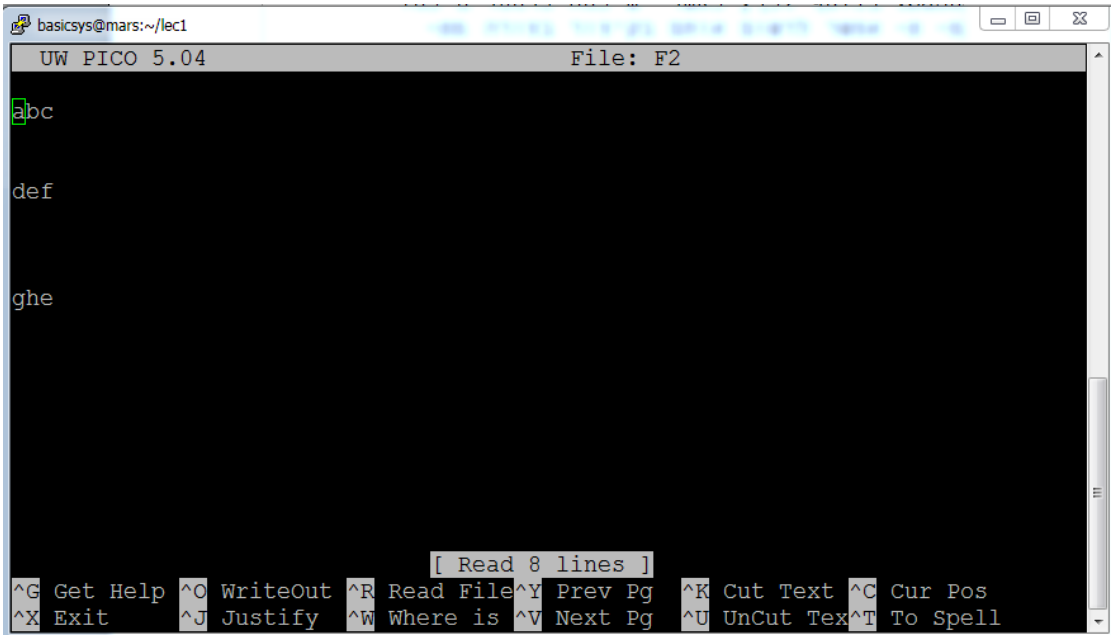
הפקודה pico <שם קובץ>

מאפשרת לערוך את הקובץ שמועבר לה כפרמטר (במידה והקובץ קיים).במידה והקובץ לא קיים נוצר קובץ בשם זה.

לדוגמה לאחר הפעלת הפקודה:

```
basicsys@mars~/lec1>pico F2
```

יתקבל המסך הבא:



פקודות חשובות בעריכת הקובץ:

כאשר רשום ^x הכוונה היא ללחיצה על מקש Ctrl
ועל מקש x (כשהמקש Ctrl לחוץ לוחצים גם על האות x).
באופן דומה לגבי ^O וכו'.

להלן תאור חלק מהאפשרויות של העריכה:

^x - לסיים את העריכה, לאחר בחירת אופציה זו תתקבל
שאלה האם לשמור את הקובץ ובאיזה שם.

^k - למחוק את השורה הנוכחית, (ולשמור אותה בזכרון
במידה ויהיה לאחר מכן שימוש בפקודה ^U).
(על ידי חזרה על ^k ניתן למחוק מספר שורות ברצף,
ואז כל השורות נשמרות בזכרון).

^u - מדביקה את השורה האחרונה שנמחקה על ידי ^k (או
את רצף השורות האחרון שנמחק על ידי ^k).

^o - שומרת את הקובץ (ולא יוצאת מה- pico).

הפקודה echo

הפקודה <טכסט> echo מציגה על המסך את הטכסט.
אופציות:
n- אל תדפיס תו קפוץ שורה בסוף הטכסט
e- אפשר שימוש בתווים מיוחדים כמו:
\n - קפיצת שורה
E- אל תאפשר שימוש בתווים מיוחדים,(זו ברירת המחדל של הפקודה)

לדוגמה:

```
basicsys@mars~/lec1>echo abc
abc

basicsys@mars~/lec1>echo -n abc
abcbasicsys@mars~/lec1>

basicsys@mars~/lec1>echo -e "abc\nde"
abc
de

basicsys@mars~/lec1>echo -E "abc\nde"
abc\nde

basicsys@mars~/lec1>echo "abc\nde"
abc\nde
```

הפקודה date

הפקודה date מדפיסה את הזמן הנוכחי

לדוגמה:

```
basicsys@mars~/lec1>date
Wed Oct 29 11:43:54 IST 2014
```

הרצת תוכנית סקריפט

כדי שתוכלו להריץ תוכניות ב- bash ותהיו מתואמים עם תוכנית הבדיקה האוטומטית, עליכם לבצע את ההוראות הבאות (פעם אחת בלבד):

1. הכנסו לחשבון שלכן בשרת.

2. באמצעות העורך pico הוסיפו את שלוש השורות המתוארות להלן לסוף הקובץ ששמו הוא .bashrc (שימו לב שהנקודה בהתחלה היא חלק משם הקובץ).

דהינו בצעו את הפקודה:

```
pico .bashrc
```

הוסיפו את 3 השורות הבאות (לסוף הקובץ):
(העתיקו את השורות בדיוק כפי שהן ללא תוספת רווחים)

```
PS1="\u@\h\w>"
set -C
PATH=$PATH:.
```

וצאו מ- pico (עם שמירה).

3. בצעו את הפקודה:

```
. .bashrc
```

מעכשיו והלאה תוכלו להריץ תוכניות סקריפט (כפי שיתואר בהמשך), אין צורך לחזור על ההוראות הנ"ל שוב לפני הרצת תוכנית סקריפט.

להרצת תוכנית סקריפט למשל תוכנית שנמצאת בקובץ P1 יש ליצור את הקובץ P1 על ידי ביצוע pico P1 ולהכניס לו סדרת פקודות של bash, כרצונכם ולצאת מ- pico עם שמירה.

למשל נניח שהקובץ P1 מכיל את התוכן הבא כפי שמוצג על ידי הפקודה cat:

```
basicsys@mars~/lec1>cat P1
date
pwd
```

```
echo hi
```
כדי להריץ את התוכנית P1 יש לתת לה הרשאת הרצה.
כדי לעשות זאת משתמשים בפקודה הבאה:

```
basicsys@mars~/lec1>chmod u+x P1
```

משמעות הפקודה היא שמאפשרים למשתמש u (שזה אתם)
הרשאת הרצה (שמסומנת על ידי x) לקובץ P1.
עכשיו להפעלת התוכנית P1 רושמים P1 וסדרת הפקודות
שבתוכנית תתבצע כסדרת פקודות של bash, ואתם תראו על
המסך את התוצאה של הפעלת הפקודות.

לדוגמה,

```
basicsys@mars~/lec1>P1
Wed Oct 29 11:59:17 IST 2014
/home/cs/segel/basicsys/lec1
hi
```

<u>רישום לקורס לצורך הגשת תרגילי בית</u>

על מנת שתוכלו להגיש תרגילי בית בקורס, עליכים לבצע
את הפקודה:

```
~basicsys/register
```

(פעם אחת בלבד) מהחשבון שלכם בשרת (על כל סטודנט
לבצע את הפקודה מהחשבון שלו, אי אפשר לרשום
סטודנטים אחרים מהחשבון שלכם).

לאחר ביצוע הפקודה תתבקשו לרשום מספר תעודת זהות שם
פרטי ושם משפחה (יש לרשום את השם הפרטי ושם המשפחה
באנגלית). שימו לב שמי ששם המשפחה שלו מכיל שתי
מילים כמו למשל ben ayoun צריך לרשום אותו כמילה
אחת עם מקף באמצע כמו למשל ben-ayoun

לדוגמה, כאשר אני מבצע את הפקודה הנ"ל מהחשבון של
הקורס מתקבלת התוצאה הבאה:

```
basicsys@mars~>~basicsys/register
Hello you called the basic systems registeration program
You need to call it in order to be able to submit
exercises.
You need to call this program just once.
```

Please enter your id number (9 digits):

אני מקליד מספר בן 9 ספרות ואז מתקבלת התוצאה הבאה:

Please enter your name (first-name and last-name):

אני מקליד **udi rotics** ומתקבלת התוצאה הבאה:

You entered, id: 111111111 name: udi rotics
OK to update your details: (y/n)?

אני מקליד **y** ומתקבלת התוצאה הבאה:

Your details were sent for update. It will take several days Until your details will be updated, bye.

הסיבה שזה לוקח כמה ימים היא שאני צריך להפעיל תוכנית בחשבון הקורס כדי שהרישום שלכם יקלט. בדרך כלל אני מפעיל אותה אחת לכמה ימים. אתם יכולים לשלוח לי מייל תזכורת כדי שאפעיל את התוכנית והרישום שלכם יקלט.
עד שהרישום שלכם יקלט לא תוכלו להגיש תרגילים בקורס. אחרי שהרישום שלכם יקלט זה ישמש אתכם לכל הקורס ולא תצטרכו לבצע שוב את הפקודה
~basicsys/register

<u>הגשת תרגיל בית ראשון</u>

בתרגיל הבית הראשון תדרשו להכין 4 תוכניות ששמן P1.1 P1.2 P1.3 P1.4 הפרוט של הדרישות שעל התוכניות למלא, יופיע בתרגיל שיועלה לאתר הקורס. (כשהתרגיל יועלה לאתר הקורס, תישלח הודעה למייל של כל מי שביקש להצטרף לרשימת התפוצה).

לאחר שהכנתם את התוכניות עליכם להגיש אותם על ידי ביצוע הפקודה
~basicsys/submit
יש לוודא שהקבצים P1.1-P1.4 נמצאים בתיקיה שבה אתם מפעילים את הפקודה הנ"ל.

לדוגמה, כאשר אני מבצע את הפקודה הנ"ל מחשבון הקורס מתקבלת התוצאה הבאה:
basicsys@mars~>~basicsys/submit
Hello you have called the submit program
Enter the exercise number:

אני מקליד 1 ומתקבלת התוצאה הבאה:

```
Hello you  have called the submit ex1 program.
Note that you can submit files only once.
Do you want to continue (y/n)?
```

אני מקליד y ומתקבלת התוצאה הבאה:

```
Enter the login name of your partner (if you do not
have a partner then enter your login name):
```

אם אתם מגישים בזוגות אז תקלידו את שם המשתמש (דהינו ה-login name) של השותף שלכם (וכדי שזה יצליח השותף שלכם היה צריך לבצע register מהחשבון שלו). אם אתם מגישים לבד תקלידו את שם המשתמש שלכם.
לדוגמה נניח שאני מגיש לבד, אז אני מקליד את שם המשתמש של חשבון הקורס basicsys ומתקבלת התוצאה הבאה:

```
Your user id is: basicsys  your partner user id is:
basicsys OK to continue (y/n)?
```

אני מקליד y ומתקבלת התוצאה הבאה:

```
The files that you have to submit are :
   P1.1 P1.2 P1.3 P1.4
Your file list is: P1.1 P1.2 P1.3 P1.4
Recall that you can submit exercises just once!!!
OK to submit your files (y/n)?
```

אני מקליד y ומתקבלת התוצאה הבאה:

```
Your files were submitted successfully. Bye.
```

בשלב הזה הקבצים נשלחו להגשה ואין אפשרות לשנותם.
במידה וגיליתם טעות בקבצים הדרך היחידה לשנות את ההגשה היא לשלוח אלי מייל ולבקש למחוק את ההגשה שלכם, ואחרי שאאשר לכם שמחקתי להגיש שוב.

הרצאה מספר 2

הפקודה tree

הפקודה <שם תיקיה> tree מציגה על המסך את מבנה התיקיה. כאשר מפעילים את הפקודה ללא פרמטרים מוצג מבנה התיקיה הנוכחית.

לדוגמה

```
basicsys@mars~>cd lec2
```

הפקודה ls ללא פרמטרים מציגה רק את הקבצים/תיקיות שנמצאים רמה אחת מתחת התיקיה הנוכחית.

```
basicsys@mars~/lec2>ls
dir2  dir7  F2  FFF  lec2-2014.txt
```

לעומת זאת, הפקודה tree מציגה את כל הקבצים/תיקיות מתחת התיקיה הנוכחית בעומק כלשהו.

```
basicsys@mars~/lec2>tree
.
|-- F2
|-- FFF
|-- dir2
|   |-- F1
|   `-- dir3
|       |-- F2
|       |-- dir4
|       |   |-- FFF1
|       |   `-- ff
|       `-- ff
|-- dir7
|   |-- FFF1
|   `-- ff
`-- lec2-2014.txt

4 directories, 10 files
```

בדוגמה הנ"ל אפשר לדעת ש dir7 היא תיקיה כי מתחתיה מופיעים עוד קבצים/תיקיות. לעומת זאת אי אפשר לדעת אם FF1 או ff הם קבצים או תיקיות. (יש פקודות אחרות שנראה בהמשך שבעזרתם אפשר לדעת את זה).

```
basicsys@mars~/lec2>tree dir2
dir2
|-- F1
`-- dir3
    |-- F2
    |-- dir4
    |   |-- FFF1
    |   `-- ff
    `-- ff

2 directories, 5 files
```

<u>שם מלא/שם יחסי של קובץ/תיקיה</u>

לכל קובץ/תיקיה אפשר לפנות בשתי צורות:

1. <u>בשם היחסי</u> יחסית לתיקיה הנוכחית.
   שם זה מאופין בכך שאינו מתחיל ב- /.

2. <u>בשם המלא</u> שמתאר את כל המסלול המלא אל הקובץ החל מהתיקיה הראשית השם המלא מאופין בכך שהוא מתחיל ב- /.

לדוגמה, הפקודה הבאה מציגה את תוכן הקובץ F2,
והפניה אל הקובץ היא בשם היחסי שלו.

```
basicsys@mars~/lec2>cat dir2/dir3/F2
def
```

הפקודה הנ"ל תלויה בתיקיה שבה אנו נמצאים ולכן היא לא תצליח אם נהיה בתיקיה אחרת.

כדי לפנות לקובץ בשמו המלא, נברר את המסלול המלא לתיקיה הנוכחית על ידי הפקודה pwd:

```
basicsys@mars~/lec2>pwd
/home/cs/segel/basicsys/lec2
```

הפקודה הבאה מציגה את תוכן הקובץ F2 והפניה אליו היא בשמו המלא. הפקודה הזאת לא תלויה בתיקיה בה אנחנו נמצאים ותצליח בכל תיקיה שבה נהיה.

```
cat /home/cs/segel/basicsys/lec2/dir2/dir3/F2
def
```

נעבור עכשיו לתיקיה אחרת:

```
basicsys@mars~/lec2>cd dir2
```

נבצע שוב את הפקודה שמשתמשת בשם היחסי של F2 ועכשיו הפקודה לא תצליח ונקבל הודעת שגיאה.

```
basicsys@mars~/lec2/dir2>cat dir2/dir3/F2
cat: dir2/dir3/F2: No such file or directory
```

לעומת זאת הפקודה שמשתמשת בשם המלא של F2 תצליח:

```
cat /home/cs/segel/basicsys/lec2/dir2/dir3/F2
def
```

תוכנית הסריקה המקדימה לביצוע פקודות

לפני שהמערכת מבצעת פקודת bash מופעלת תוכנית שנקרא לה בהמשך הסורק שמבצעת שינויים בפקודה (בהתאם לכללים שנלמד בהמשך). לכן הפקודה שהמערכת מקבלת לביצוע היא לא בדיוק הפקודה שרשמתם אלא הפקודה לאחר ביצוע השינויים של הסורק.

להלן שני כללים ראשונים לגבי שינויים שמבצע הסורק בפקודה:

1. הסורק מחליף /~ במסלול המלא לתיקיה הבית של המשתמש.
2. הסורק מחליף מחרוזת~ במסלול המלא לתיקית הבית של המשתמש שהמחרוזת מתארת (המחרוזת אמורה לתאר שם משתמש שקיים במערכת).

לדוגמה כאשר רושמים את הפקודה:

```
basicsys@mars~/lec2/dir2>cat ~/lec2/dir2/dir3/F2
```

הסורק מחליף את ה /~ במסלול המלא לתיקית הבית של המשתמש שבחשבון שלו מתבצעת הפקודה, דהינו לתיקית הבית של basicsys.

לכן הפקודה שהמערכת מקבלת לביצוע היא הפקודה הבאה:

```
cat /home/cs/segel/basicsys/lec2/dir2/dir3/F2
```

ולכן הפקודה הנ"ל למרות שלא התחילה ב- / מתייחסת לשם המלא של הקובץ F2 ותצליח בכל תיקיה.

באופן דומה כאשר רושמים את הפקודה הבאה:

```
basicsys@mars~>cat ~rotics/F1
```

הסורק מחליף את ה ~rotics במסלול המלא לתיקית הבית של המשתמש rotics. לכן הפקודה שהמערכת מקבלת לביצוע היא הפקודה הבאה:

```
basicsys@mars~>cat /home/cs/segel/rotics/F1
```

והתוצאה היא הצגת תוכן הקובץ F1:

```
abc
```

כתיבת מספר פקודות bash בשורה אחת

ניתן לכתוב מספר פקודות bash בשורה אחת, כאשר מפרידים את הפקודות ב- ; המערכת תבצע את הפקודות לפי הסדר משמאל לימין.

לדוגמה:

נניח שמבנה התיקיה dir2 הוא:

```
basicsys@mars~/lec2/dir2>tree
.
|-- F1
`-- dir3
    |-- F2
    |-- dir4
    |   |-- FFF1
    |   `-- ff
    `-- ff

2 directories, 5 files
```

לאחר הפעלת סדרת הפקודות הבאה יוצג על המסך תוכן הקובץ F2 שנמצא בתיקיה dir3.

```
basicsys@mars~/lec2/dir2>cd dir3; cat F2 ; cd ..
def
```

אופציות של הפקודה ls

בפקודה ls אפשר להשתמש באופציות הבאות:

-l מציגה שורה מלאה עבור כל קובץ/תיקיה שמכילה
פרטים נוספים על הקובץ/תיקיה שכוללים:
סוג הקובץ, הרשאות גישה לקובץ, גודל הקובץ,
התאריך האחרון שבו שונה הקובץ, שם המשתמש שהוא
בעל הקובץ.

-a מציגה בנוסף לקבצים הרגילים גם את הקבצים
הנחבאים (hidden files). ב- unix הקבצים הנחבאים
הם אלה שהשם שלהם מתחיל בנקודה, כמו
למשל .login .bashrc. וגם . (שמציין את התיקיה
הנוכחית) ו- .. (שמציין את התיקיה מעל התיקיה
הנוכחית).

-ld מציג שורה מלאה אחת, עבור התיקיה הנוכחית.

לדוגמה ls -l ללא פרמטרים מציג שורה מלאה עבור כל
הקבצים/תיקיות שנמצאים בתיקיה הנוכחית (לא בעומק
כלשהו אלא רק רמה אחת מתחת התיקיה הנוכחית).

```
basicsys@mars~/lec2/dir2>ls -l
total 8
drwx------ 3 basicsys basicsys 4096 Oct 26 10:04 dir3
-rw------- 1 basicsys basicsys   21 Oct 26 10:04 F1
```

בפלט הנ"ל אפשר לראות שהתו השמאלי ביותר מציין את
סוג הקובץ. עבור קובץ רגיל (בדוגמה F1) רשום -
ועבור תיקיה (בדוגמה dir3) רשום d.
ישנם סוגים נוספים שנראה בהמשך.

הפקודה ls -l <שם תיקיה>
מציגה שורה מלאה עבור כל הקבצים/תיקיות בתיקיה רמה
אחת בלבד מתחת התיקיה. לדוגמה:

```
basicsys@mars~/lec2/dir2>ls -l dir3
total 12
drwx------ 2 basicsys basicsys 4096 Oct 26 10:04 dir4
-rw------- 1 basicsys basicsys    4 Oct 26 10:04 F2
-rw------- 1 basicsys basicsys    4 Oct 26 10:04 ff

basicsys@mars~/lec2/dir2>ls -ld
drwx------ 3 basicsys basicsys 4096 Oct 26 10:04 .
```

הרשאות גישה לקבצים/תיקיות

ב- unix מגדירים 3 סוגי משתמשים, ולכל סוג משתמש יש אות שמסמנת אותו:

u- בעל הקובץ, דהינו המשתמש שיצר את הקובץ. בדרך כלל u מציין את שם המשתמש שבחשבון שלו נמצא הקובץ.

g- משתמשים שנמצאים באותה קבוצה (group) של בעל הקובץ (אצלנו במערכת לא משתמשים באפשרות הזאת וכל המשתמשים הם בקבוצה של עצמם בלבד).

o- כל שאר המשתמשים, דהינו משתמשים שאינם בעל הקובץ ואינם בקבוצה של בעל הקובץ.

בנוסף ב- unix מגדירים 3 סוגי הרשאות:

r- הרשאת קריאה של הקובץ
w- הרשאת כתיבה לקובץ
x- הרשאת הרצה של הקובץ

הכללים לגישה לקבצים:

כדי לקרוא/לכתוב/להריץ קובץ צריך הרשאת x/w/r לקובץ (בהתאמה) והרשאת r ו- x לכל התיקיות שנמצאות מסלול המלא אל הקובץ החל מהתיקיה /.

הפקודה ls -l מציגה את ההרשאות לקבצים/תיקיות באופן הבא:

תווים 2-4 מציגים את ההרשאות של u
תווים 5-7 מציגים את ההרשאות של g
תווים 8-10 מציגים את ההרשאות של o

לדוגמה, נבצע את הפקודה:

```
basicsys@mars~/lec2/dir2>ls -l F1
-rw------- 1 basicsys basicsys 21 Oct 26 10:04 F1
```

המשמעות של התוצאה היא שלמשתמש u יש הרשאות r ו-w ולכל שאר המשתמשים אין הרשאות גישה לקובץ F1.

הפקודה chmod

הפקודה chmod מאפשרת לבעל הקובץ/תיקיה לשנות הרשאות של הקובץ/תיקיה.

לדוגמה: הפקודה הבאה מוסיפה הרשאות r ו- x למשתמשים מסוג o לקובץ F1

```
basicsys@mars~/lec2/dir2>chmod o+rx F1
```

ולכן כשנבצע עכשיו את הפקודה ls -l F1 נראה שהתווספו הרשאות r ו- x לתווים 8-10 שמיצגים את המשתמשים מסוג o.

```
basicsys@mars~/lec2/dir2>ls -l F1
-rw----r-x 1 basicsys basicsys 21 Oct 26 10:04 F1
```

הפקודה הבאה מוסיפה הרשאות r,x למשתמשים מסוג g

```
basicsys@mars~/lec2/dir2>chmod g+rx F1

basicsys@mars~/lec2/dir2>ls -l F1
-rw-r-xr-x 1 basicsys basicsys 21 Oct 26 10:04 F1
```

הפקודה הבאה מבטלת את ההרשאות מסוג r,x גם למשתמשים מסוג o וגם למשתמשים מסוג g.

```
basicsys@mars~/lec2/dir2>chmod o-rx,g-rx F1

basicsys@mars~/lec2/dir2>ls -l F1
-rw------- 1 basicsys basicsys 21 Oct 26 10:04 F1
```

הפקודה הבאה משתמשת באופציה a- של ls כדי לראות גם את הקבצים הנחבאים (דהינו הקבצים שהשם שלהם מתחיל בנקודה).

```
basicsys@mars~/lec2/dir2>ls -a
.  ..  dir3  F1

basicsys@mars~/lec2/dir2>ls -la
total 16
drwx------ 3 basicsys basicsys 4096 Oct 26
10:04 .
```

```
drwx------ 4 basicsys basicsys 4096 Oct 26
10:05 ..
drwx------ 3 basicsys basicsys 4096 Oct 26 10:04
dir3
-rw------- 1 basicsys basicsys   21 Oct 26 10:04
F1
```

הפקודה rm

הפקודה rm <רשימת קבצים>

מוחקת את הקבצים שברשימה. במידה ואחד הקבצים שברשימה לא קיים מתקבלת הודעת שגיאה.

הפקודה rm -r <רשימת תיקיות>
מוחקת את כל התיקיות שברשימה. במידה ואחת מהתיקיות לא קיימת מתקבלת הודעת שגיאה.

האופציה i- של הפקודה rm גורמת לכך שלפני המחיקה המשתמש יתבקש לאשר את המחיקה.

האופציה f- גורמת לכך שאם קובץ/תיקיה שברשימת הקבצים/תיקיות לא קיים אז לא תתקבל הודעת שגיאה.

לדוגמה, נניח שבתיקיה temp קיימים שלושה קבצים בשם F1-F3 ושלוש תיקיות בשם d1-d3 כפי שמראה התוצאה של הפקודה ls:

```
basicsys@mars~/temp>ls
d1  d2  d3  F1  F2  F3
```

לאחר ביצוע הפקודה:

```
basicsys@mars~/temp>rm F1 F2
```

ימחקו שני הקבצים F1 ו- F2 והתוצאה של הפקודה ls תהיה:

```
basicsys@mars~/temp>ls
d1  d2  d3  F3
```

לאחר ביצוע הפקודה:

```
basicsys@mars~/temp>rm d1
rm: cannot remove `d1': Is a directory
```

התקבלה הודעת שגיאה כי אסור לבצע את הפקודה rm (ללא התוספת של r-) על תיקיות.

לאחר ביצוע הפקודה:

```
basicsys@mars~/temp>rm -r d1
```

תימחק התיקיה d1 והתוצאה של הפקודה ls תהיה:

```
basicsys@mars~/temp>ls
d2  d3  F3
```

הפקודה הבאה תנסה למחוק קובץ F4 שלא קיים ותקבל הודעת שגיאה:

```
basicsys@mars~/temp>rm F4
rm: cannot remove `F4': No such file or directory
```

הפקודה הבאה תנסה למחוק קובץ F4 ולא תקבל הודעת שגיאה בגלל שימוש באופציה f-

```
basicsys@mars~/temp>rm -f F4
basicsys@mars~/temp>
```

הפקודה הבאה תנסה למחוק את הקובץ F3 אבל בגלל שימוש באופציה i- לפני המשתמש יתבקש לאשר את המחיקה.

```
basicsys@mars~/temp>rm -i F3
rm: remove regular file `F3'?
```

במצב הזה אם המשתמש יקליד y הקובץ ימחק ואם הוא יקליד n הקובץ לא ימחק.

הפקודה cp <קובץ1> <קובץ2>

הפקודה cp <קובץ1> <קובץ2>
יוצרת עותק נוסף של קובץ1 ששמו הוא קובץ2.
במידה וקובץ1 לא קיים מתקבלת הודעת שגיאה.
במידה וקובץ2 כבר קיים הוא ידרס, דהינו תוכנו יוחלף בתוכן של קובץ1.

לדוגמה, נניח שמבנה התיקיה dir2 הוא כפי שמוצג על ידי הפקודה tree הבאה:

```
basicsys@mars~/lec2/dir2>tree
.
|-- F1
`-- dir3
    |-- F2
    |-- dir4
    |   |-- FFF1
    |   `-- ff
    `-- ff

2 directories, 5 files
```

לאחר ביצוע הפקודה הבאה:

```
sicsys@mars~/lec2/dir2>cp F1 F3
```

נוצר עותק נוסף של הקובץ F1 שבתיקיה הנוכחית ששמו F3 הוא כפי שמראה הפקודה tree הבאה:

```
basicsys@mars~/lec2/dir2>tree
.
|-- F1
|-- F3
`-- dir3
    |-- F2
    |-- dir4
    |   |-- FFF1
    |   `-- ff
    `-- ff

2 directories, 6 files
```

נחזיר את המצב לקדמותו על ידי ביצוע הפקודה הבאה:

```
basicsys@mars~/lec2/dir2>rm F3

basicsys@mars~/lec2/dir2>tree
.
|-- F1
`-- dir3
    |-- F2
    |-- dir4
    |   |-- FFF1
    |   `-- ff
    `-- ff
2 directories, 5 files
```

לאחר ביצוע הפקודה הבאה:

```
sicsys@mars~/lec2/dir2>cp dir3/F2 dir3/dir4/F3
```

נוצר עותק נוסף של הקובץ dir3/F2 ששמו הוא F3 והוא נמצא בתיקיה dir3/dir4 כפי שמראה הפקודה tree הבאה:

```
basicsys@mars~/lec2/dir2>tree
.
|-- F1
`-- dir3
    |-- F2
    |-- dir4
    |   |-- F3
    |   |-- FFF1
    |   `-- ff
    `-- ff

2 directories, 6 files
```

כדי להחזיר את המצב לקדמותו נמחק את הקובץ F3 שהוספנו על ידי הפקודה rm הבאה:

```
basicsys@mars~/lec2/dir2>rm dir3/dir4/F3

basicsys@mars~/lec2/dir2>tree
.
|-- F1
`-- dir3
    |-- F2
    |-- dir4
    |   |-- FFF1
    |   `-- ff
    `-- ff

2 directories, 5 files
```

הפקודה <תיקיה> <רשימת קבצים> cp

הפקודה <תיקיה> <רשימת קבצים> cp

הפקודה יוצרת עותקים מכל הקבצים שברשימת הקבצים ומכניסה אותם לתיקיה.
שמות העותקים הם כמו שמות הקבצים שברשימה (אבל הם נמצאים בתוך התיקיה).
במידה ואחד מהקבצים שברשימה לא קיים, או במידה והתיקיה לא קיימת מתקבלת הודעת שגיאה.

במידה ויש בתיקיה קובץ ששמו מופיע ברשימת הקבצים הוא ידרס, דהינו תוכנו יוחלף בתוכן של הקובץ ברשימת הקבצים.

לדוגמה, לאחר ביצוע הפקודה:

```
basicsys@mars~/lec2/dir2>cp F1 dir3
```

יוכנס עותק של הקובץ F1 לתיקיה dir3 כפי שמראה הפקודה tree הבאה:

```
basicsys@mars~/lec2/dir2>tree
.
|-- F1
`-- dir3
    |-- F1
    |-- F2
    |-- dir4
    |   |-- FFF1
    |   `-- ff
    `-- ff
2 directories, 6 files
```

לאחר ביצוע הפקודה:

```
basicsys@mars~/lec2/dir2>cp F1 dir3/F2 dir3/dir4
```

יוכנסו לתיקיה dir3/dir4: עותק של הקובץ F1 ששמו יהיה F1 ועותק של הקובץ dir3/F2 ששמו יהיה F2 כפי שמראה הפקודה tree הבאה:

```
basicsys@mars~/lec2/dir2>tree
.
|-- F1
`-- dir3
    |-- F1
    |-- F2
    |-- dir4
    |   |-- F1
    |   |-- F2
    |   |-- FFF1
    |   `-- ff
    `-- ff

2 directories, 8 files
```

נחזיר את המצב לקדמותו על ידי מחיקת כל הקבצים שהוספנו באמצעות פקודת rm הבאה:

```
lec2/dir2>rm dir3/F1 dir3/dir4/F1 dir3/dir4/F2

basicsys@mars~/lec2/dir2>tree
.
|-- F1
`-- dir3
    |-- F2
    |-- dir4
    |   |-- FFF1
    |   `-- ff
    `-- ff

2 directories, 5 files
```

הפקודה <תיקיה 2> <תיקיה 1> cp -r

הפקודה <תיקיה 2> <תיקיה 1> cp -r

מבצעת אחת משתי האפשרויות הבאות בהתאם לכך אם תיקיה 2 קיימת או לא:

1) אם תיקיה 2 קיימת הפקודה מעתיקה את תיקיה 1 (וכל הקבצים/תיקיות שבתוכה בעומק כלשהו) לתיקיה בשם תיקיה 1 שנמצאת בתוך תיקיה 2.
במקרה שכבר קיימת תיקיה 1 בתוך תיקיה 2 אז התיקיה הזאת תדרס.

2) אם תיקיה 2 לא קיימת הפקודה יוצרת תיקיה חדשה בשם תיקיה 2 שמכילה עותק של כל הקבצים/תיקיות שבתוך תיקיה 1 (בעומק כלשהו).

לדוגמה, הפקודה הבאה מעתיקה את התיקיה dir3 לתיקיה dir5. מאחר והתיקיה dir5 לא קיימת, תוצר תיקיה חדשה בשם dir5 שתכיל עותק של כל הקבצים בתיקיה dir3.

```
basicsys@mars~/lec2/dir2>cp -r dir3 dir5
```

מבנה התיקיה dir2 לאחר ביצוע הפקודה cp הנ"ל מוצג על ידי הפקודה tree הבאה:

```
basicsys@mars~/lec2/dir2>tree
.
|-- F1
|-- dir3
|   |-- F2
|   |-- dir4
|   |   |-- FFF1
|   |   `-- ff
|   `-- ff
`-- dir5
    |-- F2
    |-- dir4
    |   |-- FFF1
    |   `-- ff
    `-- ff

4 directories, 9 files
```

נחזיר את המצב לקדמותו על ידי מחיקת התיקיה שהוספנו באמצעות פקודת rm -r הבאה:

```
basicsys@mars~/lec2/dir2>rm -r dir5

basicsys@mars~/lec2/dir2>tree
.
|-- F1
`-- dir3
    |-- F2
    |-- dir4
    |   |-- FFF1
    |   `-- ff
    `-- ff

2 directories, 5 files
```

הפקודה הבאה מעתיקה את תיקיה dir3/dir4 לתיקיה הנוכחית. מאחר והתיקיה הנוכחית קימת יכנס עותק של התיקיה dir4 לתוך התיקיה הנוכחית.

```
basicsys@mars~/lec2/dir2>cp -r dir3/dir4 .
```

מבנה התיקיה dir2 לאחר ביצוע הפקודה cp הנ"ל מוצג על ידי הפקודה tree הבאה:

```
basicsys@mars~/lec2/dir2>tree
.
|-- F1
|-- dir3
|   |-- F2
|   |-- dir4
|   |   |-- FFF1
|   |   `-- ff
|   `-- ff
`-- dir4
    |-- FFF1
    `-- ff

3 directories, 7 files
```

הפקודה הבאה מעתיקה את תיקיה dir3/dir4 לתיקיה dir4. מאחר והתיקיה הנוכחית קימת יכנס עותק של התיקיה dir3/dir4 לתוך התיקיה dir4.

```
basicsys@mars~/lec2/dir2>cp -r dir3/dir4 dir4

basicsys@mars~/lec2/dir2>tree
.
|-- F1
|-- dir3
|   |-- F2
|   |-- dir4
|   |   |-- FFF1
|   |   `-- ff
|   `-- ff
`-- dir4
    |-- FFF1
    |-- dir4
    |   |-- FFF1
    |   `-- ff
    `-- ff

4 directories, 9 files
```

נחזיר את המצב לקדמותו על ידי מחיקת התיקיה dir4 באמצעות פקודת rm -r הבאה:

```
basicsys@mars~/lec2/dir2>rm -r dir4
```

```
basicsys@mars~/lec2/dir2>tree
.
|-- F1
`-- dir3
    |-- F2
    |-- dir4
    |   |-- FFF1
    |   `-- ff
    `-- ff

2 directories, 5 files
```

הפקודה <קובץ2> <קובץ1> mv

הפקודה <קובץ2> <קובץ1> mv
מעבירה את קובץ1 לקובץ2.
במידה וקובץ1 לא קיים מתקבלת הודעת שגיאה.
במידה וקובץ2 כבר קיים הוא ידרס, דהינו תוכנו יוחלף בתוכן של קובץ1.

לדוגמה, הפקודה הבאה מעבירה את הקובץ F1 לקובץ בשם F3 בתיקיה dir3/dir4

```
basicsys@mars~/lec2/dir2>mv F1 dir3/dir4/F3
```

מבנה התיקיה dir2 לאחר ביצוע הפקודה mv הנ"ל מוצג על ידי הפקודה tree הבאה:

```
basicsys@mars~/lec2/dir2>tree
.
`-- dir3
    |-- F2
    |-- dir4
    |   |-- F3
    |   |-- FFF1
    |   `-- ff
    `-- ff

2 directories, 5 files
```

הפקודה <תיקיה> <רשימת קבצים> mv

הפקודה <תיקיה> <רשימת קבצים> mv

הפקודה מעבירה את כל הקבצים שברשימת הקבצים לתיקיה.

במידה ואחד מהקבצים שברשימה לא קיים, או במידה והתיקיה לא קיימת מתקבלת הודעת שגיאה.
במידה ויש בתיקיה קובץ ששמו מופיע ברשימת הקבצים הוא ידרס, דהינו תוכנו יוחלף בתוכן של הקובץ ברשימת הקבצים.

הפקודה הבאה מעבירה את הקובץ F3 שבתיקיה dir3/dir4 לתיקיה הנוכחית.

```
basicsys@mars~/lec2/dir2>mv dir3/dir4/F3 .

basicsys@mars~/lec2/dir2>tree
.
|-- F3
`-- dir3
    |-- F2
    |-- dir4
    |   |-- FFF1
    |   `-- ff
    `-- ff

2 directories, 5 files
```

נחזיר את התיקיה dir2 למצבה המקורי על ידי פקודת ה-mv הבאה:

```
basicsys@mars~/lec2/dir2>mv F3 F1

basicsys@mars~/lec2/dir2>tree
.
|-- F1
`-- dir3
    |-- F2
    |-- dir4
    |   |-- FFF1
    |   `-- ff
    `-- ff

2 directories, 5 files
```

הפקודה mv <תיקיה1> <תיקיה2>

הפקודה mv <תיקיה1> <תיקיה2>

מבצעת אחת משתי האפשרויות הבאות בהתאם לכך אם תיקיה2 קיימת או לא:

1) אם תיקיה2 קיימת הפקודה מעבירה את תיקיה1 (וכל הקבצים/תיקיות שבתוכה בעומק כלשהו) לתיקיה בשם תיקיה1 שנמצאת בתוך תיקיה2.
במקרה שכבר קיימת תיקיה1 בתוך תיקיה2 אז תתקבל הודעת שגיאה והפקודה לא תתבצע.

2) אם תיקיה2 לא קיימת הפקודה יוצרת תיקיה חדשה בשם תיקיה2 ומעבירה לתוכה כל הקבצים/תיקיות שבתוך תיקיה1 (בעומק כלשהו).

לדוגמה, הפקודה הבאה מעבירה את תיקיה dir3/dir4 לתיקיה הנוכחית. מאחר והתיקיה הנוכחית קיימת הפקודה מעבירה את התיקיה dir3/dir4 לתוך התיקיה הנוכחית לתיקיה בשם dir4.

```
basicsys@mars~/lec2/dir2>mv dir3/dir4 .

basicsys@mars~/lec2/dir2>tree
.
|-- F1
|-- dir3
|   |-- F2
|   `-- ff
`-- dir4
    |-- FFF1
    `-- ff
2 directories, 5 files
```

הפקודה הבאה מעבירה את התיקיה dir4 לתיקיה dir3.
מאחר והתיקיה dir3 קיימת הפקודה מעבירה את התיקיה dir4 לתוך התיקיה dir3 לתיקיה בשם dir4.

```
basicsys@mars~/lec2/dir2>mv dir4 dir3

basicsys@mars~/lec2/dir2>tree
.
|-- F1
`-- dir3
    |-- F2
    |-- dir4
    |   |-- FFF1
    |   `-- ff
    `-- ff
```

```
2 directories, 5 files
```

הפקודה הבאה מעבירה את התיקיה dir3/dir4 לתיקיה בשם dir5 בתוך התיקיה dir3. מאחר והתיקיה dir5 לא קיימת בתוך התיקיה dir3 תיווצר תיקיה חדשה בשם dir5 שתכיל את הקבצים והתיקיות של התיקיה dir3/dir4.

```
basicsys@mars~/lec2/dir2>mv dir3/dir4 dir3/dir5

basicsys@mars~/lec2/dir2>tree
.
|-- F1
`-- dir3
    |-- F2
    |-- dir5
    |   |-- FFF1
    |   `-- ff
    `-- ff

2 directories, 5 files
```

שליחת דואר אלקטרוני

לכל בעל חשבון בשרת יש דואר אלקטרוני שכתובתו היא:
<שם המשתמש>@mars.netanya.ac.il
כדי לשלוח דואר אלקטרוני מהחשבון בשרת יש מספר אפשרויות:

1) על ידי הפעלת הפקודה:

```
mail <כתובת דואר אלקטרוני>
```

לאחר הפעלת הפקודה, מופיעה שורה שמבקשת לרשום נושא לאחר מכן כותבים טקסט כלשהו (לעבור לשורה הבאה לוחצים על enter) ולבסוף לסיום הטקסט כותבים שורה שמכילה רק נקודה בתחילת השורה (זה מסמן שסיימתם את הכנסת הטקסט).
לאחר מכן כותבים כתובת דואר אלקטרוני של מי שאמור לקבל עותק מהמייל, והמייל נשלח.
לדוגמה:

```
basicsys@mars~>mail rotics6@gmail.com
Subject: hello
Hello
asdafasd
.
```

Cc:

2) רושמים את תוכן ההודעה בתוך קובץ (בעזרת pico), ומפעילים את הפקודה הבאה:
mail -s <נושא> <כתובת דואר אלקטרוני> < <שם קובץ>

לדוגמה, נניח שתוכן ההודעה נמצא בקובץ בשם FFF אזי הפקודה הבאה תשלח את התוכן של הקובץ (לא כקובץ מצורף אלא כטקסט) לכתובת הדואר האלקטרוני:
rotics6@gmail.com

mail -s "hello" rotics6@gmail.com < FFF

3) רושמים pine ונכנסים למסך של קורא/שולח מיילים. לומדים את השימוש בו בקלות (כמו pico). (בפעם הראשונה שמפעילים את pine עונים עונים y על השאלות שהוא שואל).

הפנית כל המילים שמגיעים לחשבון בשרת לכתובת מייל נוספת.

כדי להפנות את כל המיילים שמגיעים לשרת לכתובת מייל נוספת יש ליצור בתיקית הבית שלכם (בעזרת pico) קובץ בשם forward. ולהכניס אליו בשורה הראשונה
\<שם המשתמש שלכם>, <כתובת דואר אלקטרוני >
ואז כל המיילים שמגיעים לכתובת שלכם בשרת יגיעו גם לכתובת הנוספת שהכנסתם. אם רושמים רק:
<כתובת דואר אלקטרוני>
אז כל המיילים שמגיעים לכתובת שלכם בשרת יועברו לכתובת המייל שרשמתם ולא ישאר עותק מהם בשרת.

לדוגמה תוכן הקובץ forward. שנמצא בתיקית הבית בחשבון שלי בשרת ששם המשתמש שלו הוא rotics נראה כך:

```
rotics@mars~>cat .forward
\rotics, rotics@netanya.ac.il
```

הפקודה <קובץ> wc

הפקודה <קובץ> wc מציגה פרטים על הקובץ.

בפקודה wc אפשר להשתמש באופציות הבאות:

l- מציגה את מספר השורות בקובץ

w- מציגה את מספר המילים בקובץ (מילה היא רצף תווים ללא תווי רווח או קפיצת שורה)

c- מציגה את מספר התווים בקובץ (גם התו n\ וגם תווי רווח נספרים).

L- מציגה את מספר התווים בשורה הארוכה ביותר בקובץ לא כולל התו n\

wc ללא אופציות מציגה את שלושת המספרים הראשונים ברשימת האופציות הנ"ל לפי הסדר משמאל לימין ולאחר מכן שם הקובץ.

לדוגמה נניח שתוכן הקובץ F1 הוא כפי שמוצג על ידי פקודת ה- cat הבאה:

```
basicsys@mars~/lec2/dir2>cat F1
abcd
efg
123
45
6789  10  20 abcd
```

הפקודה wc הבאה מציגה שבקובץ F1 יש 5 שורות 8 מילים ו- 34 תווים. שימו לב שישנו תו רווח בהתחלת השורה. בהמשך נראה שיטות לצמצום רווחים.

```
basicsys@mars~/lec2/dir2>wc F1
 5  8 34 F1
```

הפקודה wc הבאה מחזירה את מספר השורות בקובץ ולאחר מכן את שם הקובץ:

```
basicsys@mars~/lec2/dir2>wc -l F1
5 F1
```

אם נרצה לקבל רק את מספר השורות בקובץ ללא שם הקובץ אנחנו יכולים להשתמש בצורה הבאה של הפקודה wc שבה

שם הקובץ מועבר מימין לסימן < (בהמשך נסביר למה במקרה זה הפקודה לא מציגה את שם הקובץ).

```
basicsys@mars~/lec2/dir2>wc -l < F1
5
```

באופן דומה נוכל להשתמש באופציות w- ו- c- כפי שמודגם להלן:

```
basicsys@mars~/lec2/dir2>wc -w F1
8 F1

basicsys@mars~/lec2/dir2>wc -w < F1
8

basicsys@mars~/lec2/dir2>wc -c F1
34 F1

basicsys@mars~/lec2/dir2>wc -c < F1
34

basicsys@mars~/lec2/dir2>wc -lw F1
 5  8 F1
```

הפקודה wc הבאה מציגה שאורך השורה הארוכה ביותר בקובץ F1 הוא 17

```
basicsys@mars~/lec2/dir2>wc -L F1
17 F1
```

משתנים ב- bash

משתנה הוא כתובת בזכרון שיש לה שם (שם המשתנה) וערך שהוא התוכן שנמצא בכתובת בזכרון (ערך המשתנה).
ב- bash ערכי המשתנים הם מחרוזות. במידה ולמשתנה לא הוכנס ערך הערך שלו הוא מחרוזת ריקה.

<ערך>=<שם המשתנה>

כדי להציב ערך למשתנה רושמים:

```
<שם המשתנה>=<ערך>
```

הערכים של המשתנים נמחקים כאשר יוצאים מהחשבון.

"<שם משתנה>$"

הסורק מחליף את "<שם משתנה>$" בערך המשתנה.
במידה ולמשתנה לא הוכנס ערך, הערך שלו הוא מחרוזת ריקה.

<שם משתנה>$
הסורק מחליף את <שם משתנה>$ בערך המשתנה לאחר צמצום רצפים של רווחים לרווח אחד וכן צמצום תווי קפוץ שורה והפיכתם לתו רווח.

לדוגמה, בפקודה הבאה הסורק מחליף את "$x" במחרוזת ריקה, מאחר ולמשתנה x לא הוכנס ערך, ולכן הפקודה הבאה תדפיס שורה ריקה.

```
basicsys@mars~/lec2/dir2>echo "$x"
```

נציב ערך 10 למשתנה x על ידי הפקודה הבאה:

```
basicsys@mars~/lec2/dir2>x=10


basicsys@mars~/lec2/dir2>echo x
x

basicsys@mars~/lec2/dir2>echo "$x"
10
```

נציב ל- x מחרוזת שמכילה רצף של רווחים.

```
basicsys@mars~/lec2/dir2>x="ab    cd    ef"
```

בפקודה הבאה הסורק מחליף את "$x" בערך של המשתנה x בדיוק כפי שהוא ללא צמצום רווחים.

```
basicsys@mars~/lec2/dir2>echo "$x"
ab    cd    ef
```

בפקודה הבאה הסורק מחליף את $x בערך של המשתנה x לאחר צמצום כל רצף של רווחים לרווח אחד:

```
basicsys@mars~/lec2/dir2>echo $x
ab cd ef
```

הפקודה <שם משתנה> read

הפקודה <שם משתנה> read מבקשת קלט מהמשתמש (באמצעות המקלדת). המשתמש מקליד מחרוזת וכאשר המשתמש מקליד enter המשתנה מקבל ערך שהוא המחרוזת שהמשתמש הקליד. אם למשתנה כבר היה ערך, הוא נדרס.

לדוגמה, הפקודה הבאה מבקשת משמשתמש להכניס ערך למשתנה ששמו x. המשתמש מקליד 123 והערך שנכנס למשתנה x הוא 123.

```
basicsys@mars~/lec2/dir2>read x
123

basicsys@mars~/lec2/dir2>echo $x
123
```

חישוב ביטויים אריתמטיים

הסורק מחליף [ביטוי אריתמטי]$ בתוצאה של החישוב של הביטוי האריתמטי.
לדוגמה:

```
basicsys@mars~/lec2/dir2>echo $[5+8]
13


basicsys@mars~/lec2/dir2>x=9

basicsys@mars~/lec2/dir2>y=8

basicsys@mars~/lec2/dir2>echo $[$x+$y]

17
```

$(<שורת פקודה>)

הסורק מחליף $(<שורת פקודה>) בפלט של הפקודה.

ולכן כאשר נשתמש למשל במבנה הבא:

<שם משתנה>=$(<שורת פקודה>)

הערך שהמשתנה יקבל יהיה הפלט של הפקודה.

לדוגמה נניח שתוכן הקובץ F1 הוא כפי שמראה הפקודה הבאה:

```
basicsys@mars~/lec2/dir2>cat F1
abcd


efg
6789  10  20 abcd
```

לאחר ביצוע הפקודה הבאה הערך שהמשתנה x יקבל יהיה תוצאת הפעלת הפקודה cat F1 שהוא תוכן הקובץ F1 כולל הרווחים וכולל תווי הקפיצת שורה שבקובץ.

```
basicsys@mars~/lec2/dir2>x=$(cat F1)
```

נציג את הערך ש- x קיבל בדיוק כפי שהוא על ידי שימוש ב- "$x"

```
basicsys@mars~/lec2/dir2>echo "$x"
abcd


efg
6789  10  20 abcd
```

נציג את הערך של x לאחר ביצוע צמצום רצפים של רווחים והפיכת תווי קפיצת שורה לתווי רווח ונקבל:

```
basicsys@mars~/lec2/dir2>echo $x
abcd efg 6789 10 20 abcd
```

<u>קליטת נתונים מקובץ</u>

כאשר משתמשים במבנה הבא:

```
<פקודה> < <שם קובץ>
```

אם הפקודה תבקש קלט, הוא יגיע מהקובץ במקום מהמקלדת.

לדוגמה, הפקודה read x הבאה מבקשת קלט:

```
basicsys@mars~/lec2/dir2>read x
```

נניח שהמשתמש מקליד:

```
100
```

אזי המשתנה x מקבל ערך 100 כפי שמראה הפקודה הבאה:

```
basicsys@mars~/lec2/dir2>echo $x
100
```

מצד שני הפקודה read x<F1 הבאה מבקשת קלט אבל הקלט יגיע מהקובץ F1 (מהשורה הראשונה בקובץ) במקום מהמקלדת. נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec2/dir2>cat F1
abcd


efg
6789  10  20 abcd
```

לכן לאחר ביצוע הפקודה הבאה:

```
basicsys@mars~/lec2/dir2>read x<F1
```

הערך שהמשתנה x מקבל הוא השורה הראשונה בקובץ F1 כפי שמראה הפקודה הבאה:

```
basicsys@mars~/lec2/dir2>echo $x
abcd
```

להלן תוכנית בשם P1 שקולטת שני מספרים ומדפיסה את סכומם.

```
basicsys@mars~/lec2/dir2>cat P1
read x
read y
echo $x+$y=$[$x+$y]
```

כאשר מפעילים את התוכנית באופן הבא:

```
basicsys@mars~/lec2/dir2>P1
```

המשתמש מתבקש להקליד 2 מספרים מהמקלדת, נניח שהקליד:

```
10
20
```

לאחר מכן התוכנית מדפיסה:

```
10+20=30
```

נניח שתוכן הקובץ F2 הוא כפי שמראה הפקודה הבאה:

```
basicsys@mars~/lec2/dir2>cat F2
300
400
```

500

כאשר מפעילים את התוכנית באופן הבא:

```
basicsys@mars~/lec2/dir2>P1<F2
```

המשתמש לא מתבקש להקליד 2 מספרים אלא המספרים מגיעים משתי השורות הראשונות בקובץ F2, ולכן התוכנית מדפיסה:

```
300+400=700
```

אם נבצע את הפקודה שוב, נקבל את אותה התוצאה כי בסיום הפקודה הקודמת הקובץ F2 נסגר, ועכשיו כשנבצע שוב את הפקודה הקובץ F2 יפתח מחדש לקריאה ושוב תקראנה שתי השורות הראשונות בקובץ.

```
basicsys@mars~/lec2/dir2>P1<F2
300+400=700
```

הרצאה מספר 3

הפקודה seq

הפקודה seq מדפיסה סדרה של מספרים על המסך.

בשימוש בפקודה בצורה:

seq <מספר>

תוצג סדרת מספרים החל מ- 1 ועד המספר.

בשימוש בפקודה בצורה:

seq <מספר 1> <מספר 2>

תוצג סדרת מספרים החל ממספר 1 ועד מספר 2 בקפיצות של 1.

בשימוש בפקודה בצורה:

seq <מספר 1> <מספר 2> <מספר 3>

תוצג סדרת מספרים החל ממספר 1 ועד מספר 3 בקפיצות של מספר 2.

האופציה "<מחרוזת>"s- גורמת לכך שבין המספרים שיודפסו תוצג המחרוזת. ברירת המחדל היא \n ולכן אם לא משתמשים באופציה s- אז בין המספרים בסדרה תהיה ירידת שורה.

לדוגמה:

```
basicsys@mars~/lec3>seq 4
1
2
3
4

basicsys@mars~/lec3>seq -s" " 4
1 2 3 4

basicsys@mars~/lec3>seq -s" " 4
1 2 3 4

basicsys@mars~/lec3>seq -s" " 4 8
4 5 6 7 8
```

```
basicsys@mars~/lec3>seq -s" " 4 2 11
4 6 8 10

basicsys@mars~/lec3>seq -s" " 4 0.5 6
4 4.5 5 5.5 6

basicsys@mars~/lec3>seq -s" " 4 0.2 6
4 4.2 4.4 4.6 4.8 5 5.2 5.4 5.6 5.8
```

הפניות קלט/פלט

ברירת המחדל ב unix היא:

הקלט, שנקרא standard input מגיע מהמקלדת.

הפלט הרגיל שנקרא standard output מופנה למסך.

פלט השגיאות שנקרא standard error מופנה למסך.

כדי לשנות את ברירת המחדל של הקלט משתמשים במבנה הבא:

<שם קובץ> < פקודה/תכנית

כתוצאה מכך, הקלט לפקודה/תכנית מגיע מהקובץ במקום מהמקלדת.

לדוגמה, נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec3>cat F1
20
30
```

בפקודה:

```
basicsys@mars~/lec3>read x < F1
```

הקלט לפקודה read x מגיע מהקובץ F1 במקום מהמקלדת. ולכן הערך שהמשתנה x מקבל הוא השורה הראשונה בקובץ F1 כפי שמראה הפקודה הבאה:

```
basicsys@mars~/lec3>echo $x
20
```

כדי לשנות את ברירת המחדל של הפלט הרגיל משתמשים במבנה הבא:

<u><שם קובץ> |< פקודה/תכנית></u>

כתוצאה מכך הפלט הרגיל של הפקודה/תכנית מופנה לקובץ במקום למסך. אם הקובץ לא קיים אז הוא נוצר. אם הקובץ קיים אז התוכן הישן שלו נדרס ומוחלף בפלט של הפקודה/תכנית.

לדוגמה, לאחר הפעלת הפקודה הבאה:

```
basicsys@mars~/lec3>seq 3 >|F1
```

על המסך לא נראה כלום כי הפלט של הפקודה seq 3 נכנס לקובץ במקום להופיע על המסך.
לכן לאחר ביצוע הפקודה הנ"ל התוכן של הקובץ F1 הוא:

```
basicsys@mars~/lec3>cat F1
1
2
3
```

ניתן לשנות את ברירת המחדל של הפלט הרגיל גם על ידי שימוש במבנה הבא:

<u><שם קובץ> < פקודה/תכנית></u>

כתוצאה מכך הפלט הרגיל של הפקודה/תכנית מופנה לקובץ במקום למסך. במידה והקובץ לא קיים אז הוא נוצר.
במידה והקובץ קיים יש שתי אפשרויות:
אם האופציה no-clobber בתוקף אז תתקבל הודעת שגיאה "אסור לדרוס את הקובץ".
אם האופציה no-clobber לא בתוקף אז התוכן הישן של הקובץ נדרס ומוחלף בפלט של הפקודה/תכנית.

הפקודה:

```
set -o noclobber
```

וגם הפקודה:

```
set -C
```

גורמת לכך שהאופציה no-clobber תהיה בתוקף.

הפקודה:

```
set +o noclobber
```

וגם הפקודה:

```
set +C
```

גורמת לכך שהאופציה no-clobber לא תהיה בתוקף.

לדוגמה,

כשנבצע את הפקודה:

```
basicsys@mars~/lec3>echo abc > F1
```

מאחר והקובץ F1 קיים והאופציה no-clobber בתוקף נקבל את הודעת השגיאה:

```
-bash: F1: cannot overwrite existing file
```

נשנה את האופציה no-clobber כך שלא תהיה בתוקף על ידי הפקודה:

```
basicsys@mars~/lec3>set +o noclobber
```

עכשיו נבצע שוב את הפקודה:

```
basicsys@mars~/lec3>echo abc > F1
```

ולא נקבל הודעת שגיאה. התוכן של הקובץ F1 אחרי ביצוע הפקודה הנ"ל הוא:

```
basicsys@mars~/lec3>cat F1
abc
```

נשנה את האופציה no-clobber כך שתהיה בתוקף על ידי הפקודה:

```
basicsys@mars~/lec3>set -o noclobber
```

נבצע את הפקודה:

```
basicsys@mars~/lec3>echo edf > F1
```

ונקבל הודעת שגיאה:

```
-bash: F1: cannot overwrite existing file
```

נשנה את האופציה no-clobber כך שלא תהיה בתוקף על ידי הפקודה:

```
basicsys@mars~/lec3>set +C
```

עכשיו נבצע שוב את הפקודה:

```
basicsys@mars~/lec3>echo def > F1
```

ולא נקבל הודעת שגיאה. התוכן של הקובץ F1 אחרי ביצוע הפקודה הנ"ל הוא:

```
basicsys@mars~/lec3>cat F1
def
```

ב- unix הפלט הרגיל מיוצג על יד המספר 1 ופלט השגיאות מיוצג על ידי המספר 2

המבנה:

<שם קובץ> |< פקודה/תכנית

שקול למבנה:

<שם קובץ> |<1 פקודה/תכנית

והמבנה:

<שם קובץ> < פקודה/תכנית

שקול למבנה:

<שם קובץ> <1 פקודה/תכנית

כדי לשנות את ברירת המחדל של פלט השגיאות ניתן להשתמש במבנים הבאים:

<שם קובץ> |<2 פקודה/תכנית

כתוצאה מכך פלט השגיאות של הפקודה/תכנית יופנה לקובץ. אם הקובץ לא קיים אז הוא נוצר. אם הקובץ קיים תוכנו ידרס ויוחלף בפלט השגיאות של הפקודה/תכנית.

<שם קובץ> <2 פקודה/תכנית

כתוצאה מכך פלט השגיאות של הפקודה/תכנית יופנה לקובץ. אם הקובץ לא קיים אז הוא נוצר. אם הקובץ קיים והאופציה no-clobbr בתוקף תתקבל הודעת שגיאה. אם הקובץ קיים והאופציה no-clobber לא בתוקף תוכנו של הקובץ ידרס ויוחלף בפלט השגיאות של הפקודה/תכנית.

<שם קובץ2> |<2 <שם קובץ1> |< פקודה/תכנית

כתוצאה מכך הפלט הרגיל יופנה לקובץ 1 (וידרוס את תוכנו אם הוא קיים) ופלט השגיאות יופנה לקובץ 2 (וידרוס את תוכנו אם הוא קיים).

<u>1&>2 <שם קובץ1> |< פקודה/תכנית</u>

כתוצאה מכך גם הפלט הרגיל וגם פלט השגיאות יופנו לקובץ 1 (וידרסו את תוכנו, אם הוא קיים).

לדוגמה, נפעיל את הפקודה הבאה:

```
basicsys@mars~/lec3>echo1 ddd
```

מאחר והפקודה לא חוקית נקבל על המסך את ההודעה הבאה שמתקבלת מפלט השגיאות:

```
-bash: echo1: command not found
```

נפעיל את הפקודה הבאה שבה פלט השגיאות מופנה לקובץ F1:

```
basicsys@mars~/lec3>echo1 ddd 2>|F1
```

לאחר הפעלת הפקודה הנ"ל לא נראה כלום על המסך כי ההודעה של השגיאה הופנתה לקובץ F1 (ודרסה את תוכנו הקודם), לכן התוכן של F1 הוא:

```
basicsys@mars~/lec3>cat F1
-bash: echo1: command not found
```

נפעיל את אותה הפקודה כמו קודם אבל עם שימוש ב > במקום >| ,

```
basicsys@mars~/lec3>echo1 ddd 2>F1
```

מאחר והקובץ F1 קיים והאופציה no-clobber בתוקף נקבל את הודעת השגיאה הבאה:

```
-bash: F1: cannot overwrite existing file
```

ניצור תכנית בשם P1 שתוכנה הוא:

```
basicsys@mars~/lec3>cat P1
echo hi
echo1 abab
echo bye
```

```
basicsys@mars~/lec3>chmod u+x P1
```

בפקודה הבאה נפעיל את התכנית P1 ונקבל על המסך גם את הפלט הרגיל וגם את פלט השגיאות:

```
basicsys@mars~/lec3>P1
hi
./P1: line 2: echo1: command not found
bye
```

בפקודה הבאה נפעיל את התכנית P1 ונפנה את הפלט הרגיל לקובץ F1. לכן על המסך נראה רק את פלט השגיאות:

```
basicsys@mars~/lec3>P1 >| F1
./P1: line 2: echo1: command not found
```

ולתוך הקובץ F1 יכנס הפלט הרגיל (וידרוס את תוכנו הקודם):

```
basicsys@mars~/lec3>cat F1
hi
bye
```

בפקודה הבאה נפעיל את התכנית P1 ונפנה את פלט השגיאות לקובץ F1. לכן על המסך נראה רק את הפלט הרגיל:

```
basicsys@mars~/lec3>P1 2>| F1
hi
bye
```

ולתוך הקובץ F1 יכנס פלט השגיאות (וידרוס את תוכנו הקודם):

```
basicsys@mars~/lec3>cat F1
./P1: line 2: echo1: command not found
```

בפקודה הבאה נפעיל את התכנית P1 ונפנה את הפלט הרגיל לקובץ F1 ואת פלט השגיאות לקובץ F2. לכן על המסך לא נראה כלום:

```
basicsys@mars~/lec3>P1 1>|F1 2>|F2
```

תוכן הקובץ F1 הוא:

```
basicsys@mars~/lec3>cat F1
hi
bye
```

תוכן הקובץ F2 הוא:

```
basicsys@mars~/lec3>cat F2
```

```
./P1: line 2: echo1: command not found
```

בפקודה הבאה נפעיל את התכנית P1 ונפנה את גם את הפלט הרגיל וגם את פלט השגיאות לקובץ F1. לכן על המסך לא נראה כלום.

```
basicsys@mars~/lec3>P1 1>|F1 2>&1
```

תוכן הקובץ F1 הוא:

```
basicsys@mars~/lec3>cat F1
hi
./P1: line 2: echo1: command not found
bye
```

ניתן גם להשתמש במבנים הבאים:

<u><שם קובץ> << פקודה/תכנית</u>

כתוצאה מכך הפלט הרגיל של הפקודה/תכנית יופנה לסוף הקובץ. (תוכן הקובץ לא נדרס אלא מתווסף לו הפלט הרגיל של הפקודה/תכנית).

<u><שם קובץ> <<2 פקודה/תכנית</u>

כתוצאה מכך פלט השגיאות של הפקודה/תכנית יופנה לסוף הקובץ.

<u><שם קובץ2> <<2 <שם קובץ1> << פקודה/תכנית</u>

כתוצאה מכך הפלט הרגיל יופנה לסוף קובץ1 ופלט השגיאות יופנה לסוף קובץ2.

<u>2>&1 <שם קובץ> << פקודה/תכנית</u>

כתוצאה מכך גם הפלט הרגיל וגם פלט השגיאות של הפקודה/תכנית יופנה לסוף הקובץ.

לדוגמה, נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec3>cat F1
abc
```

בפקודה הבאה, נפנה את הפלט הרגיל לסוף קובץ F1. לכן על המסך לא נראה כלום.

```
basicsys@mars~/lec3>echo def >> F1
```

ותוכן הקובץ F1 הוא:

```
basicsys@mars~/lec3>cat F1
abc
def
```

בפקודה הבאה, נפנה את פלט השגיאות לסוף קובץ F1. לכן על המסך לא נראה כלום.

```
basicsys@mars~/lec3>echo def 2>> F1
```

ותוכן הקובץ F1 הוא:

```
basicsys@mars~/lec3>cat F1
abc
def
-bash: echo1: command not found
```

ניתן גם לשלב שימוש ב- << וב- |< באותה פקודה.

לדוגמה, נניח שתוכן הקובץ F2 הוא:

```
basicsys@mars~/lec3>cat F2
20
30
```

בפקודה הבאה הפלט הרגיל של התכנית P1 מופנה לקובץ F1 ודורס את תוכנו ופלט השגיאות של התכנית מופנה לקובץ F2 לסוף הקובץ.

```
basicsys@mars~/lec3>P1 >|F1 2>>F2
```

תוכן הקובץ F1 הוא:

```
basicsys@mars~/lec3>cat F1
hi
bye
```

תוכן הקובץ F2 הוא:

```
basicsys@mars~/lec3>cat F2
20
30
./P1: line 2: echo1: command not found
```

<u>הפקודה <שם קובץ> more</u>

הפקודה <שם קובץ> more מציגה את הקובץ על המסך כך שאם מספר השורות בקובץ גדול ממספר השורות במסך תוצג

התחלת תוכן הקובץ (שנכנסת למסך), והמשך התוכן של
הקובץ יוצג רק לאחר שהמשתמש יקליד רווח.
המשתמש יכול להקליד Enter ואז התוכן יתקדם בשורה
המשתמש יכול להקליד מספר ואז Enter והתוכן יתקדם
מספר שורות. לדוגמה, אם המשתמש מקליד 3 Enter
אז התוכן מתקדם 3 שורות.
לדוגמה,
נניח שהקובץ F1 מכיל את סדרת המספרים 1...100 כפי
שמתקבל על ידי הפקודה הבאה:

```
basicsys@mars~/lec3>seq 100 >| F1
```

לאחר ביצוע הפקודה:

```
basicsys@mars~/lec3>more F1
```

יתקבל המסך הבא:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
--More--(20%)
```

כעת אם נקליד רווח נקבל את המשך תוכן הקובץ ואם
נקליד enter המסך הנ"ל יתקדם לשורה הבאה בקובץ.
ניתן להקליד Ctrl+c כדי לצאת מהפקודה (לפני שהציגה
את כל תוכן הקובץ).

התכנית הבאה מבקשת מהמשתמש להקליד שתי שורות שמציגות שמות של קבצים ומדפיסה את סכום השורות של שני הקבצים.

```
basicsys@mars~/lec3>cat P1
read x
read y
z=$(wc -l < $x)
t=$(wc -l < $y)
echo $[$z+$t]
```

נניח שהקובץ F1 מכיל 2 שורות והקובץ F2 מכיל שורה אחת.

לאחר הפעלת התכנית על ידי הפקודה:

```
basicsys@mars~/lec3>P1
```

המשתמש מתבקש להכניס שתי שורות קלט מהמקלדת, נניח שהמשתמש הקליד:

```
F1
F2
```

מאחר וסכום השורות של הקבצים F1 ו- F2 הוא 3 הפלט של התכנית יהיה:

```
3
```

הפקודה <תנאי> test

הפקודה <תנאי> test בודקת נכונות של התנאי. בדרך כלל משתמשים בה לבדיקת תנאים בתוך פקודות אחרות (כמו if ו- while).

המבנה:

[<תנאי>]

שקול לפקודה <תנאי> test

(חשוב להקפיד על תו רווח אחד לפחות אחרי הסוגר השמאלי ולפני הסוגר הימני).

להלן סיכום סוגי התנאים שנלמד.

להשוואה בין מספרים משתמשים בתנאים הבאים:

-gt	גדול
-ge	גדול או שווה
-lt	קטן

-le קטן או שווה
-ne שונה
-eq שווה

להשוואה בין מחרוזות משתמשים בתנאים הבאים:

= שווה
!= שונה
> גדול (אבל צריך להשתמש בזה בתוך גרשיים כפולים דהינו ">" כדי שהסורק לא יחשוב שזו הפנית פלט)
< קטן (אבל צריך להשתמש בזה בתוך גרשיים כפולים דהינו "<" כדי שהסורק לא יחשוב שזו הפנית קלט)

לבדיקת סוגי קבצים משתמשים בתנאים הבאים:

-f קובץ רגיל
-d תיקיה
-e קובץ או תיקיה

לדוגמה, התכנית הבאה קולטת מספר מהמשתמש ומדפיסה הודעה אם המספר גדול מ- 5 או לא.

```
basicsys@mars~/lec3>cat P1
read x
if [ $x -gt 5 ]
then
 echo "The number you entered is greater than 5"
else
 echo "The number you entered is less than or  equal 5"
fi
```

לאחר הפעלת התכנית על ידי הפקודה:

```
basicsys@mars~/lec3>P1
```

נניח שהמשתמש מקליד:

```
8
```

הפלט של התכנית הוא:

```
The number you entered is greater than 5
```

לאחר הפעלת התכנית על ידי הפקודה:

```
basicsys@mars~/lec3>P1
```

נניח שהמשתמש מקליד:

```
3
```

הפלט של התכנית הוא:

```
The number you entered is less than or equal 5
```

התכנית הבאה קולטת מספר מהמשתמש ומדפיסה סדרת יורדת של מספרים החל מהמספר ועד המספר 3.

```
basicsys@mars~/lec3>cat P1
read x
while [ $x -gt 2 ]
do
 echo $x
 x=$[$x-1]
done
```

לאחר הפעלת התכנית על ידי הפקודה:

```
basicsys@mars~/lec3>P1
```

נניח שהמשתמש מקליד:

```
6
```

הפלט של התכנית הוא:

```
6
5
4
3
```

התכנית הבאה קולטת מספרים מהמשתמש, עד שהמשתמש מקליד Ctrl+d. אם המשתמש מקליד מספר שגדול מ- 4 המספר מודפס. כאשר המשתמש מקליד Ctrl+d פקודת ה- read x שבתוך ה- while מחזירה false והלולאה מסתיימת.

```
basicsys@mars~/lec4>cat P1
while read x
do
 if [ $x -gt 4 ]
  then
   echo $x
  fi
done
```

לאחר הפעלת התכנית על ידי הפקודה:

```
basicsys@mars~/lec3>P1
```

נניח שהמשתמש מקליד:

```
70
```

הפלט של התכנית הוא:

```
70
```

התכנית מבקשת לקלוט מספר נוסף מהמשתמש, נניח שהמשתמש מקליד:

```
2
```

```
     מאחר ו- 2 אינו גדול מ- 4 תנאי ה- if לא מתקיים
      והתוכנית לא מדפיסה 2 אלא חוזרת לתחילת הלולאה
      ומבקשת לקלוט מספר נוסף מהמשתמש. נניח שהמשתמש
                                        מקליד:
4
     מאחר ו- 4 אינו גדול מ- 4 תנאי ה- if לא מתקיים
     והתוכנית לא מדפיסה  4 אלא חוזרת לתחילת הלולאה
                ומבקשת לקלוט מספר נוסף מהמשתמש.

  נניח שהמשתמש מקליד: Ctrl+d. כתוצאה מכך ה- read x
     מחזירה false, ולכן התנאי של לולאת ה- while לא
      מתקיים והלולאה מסתימת. מאחר ואחרי הלולאה אין
                  פקודות נוספות התוכנית מסתימת.

  הפקודה הבאה מכניסה לקובץ F1 את סדרת המספרים 8-1:

basicsys@mars~/lec3>seq 8 >| F1

        לאחר ביצוע הפקודה הנ"ל תוכן הקובץ F1 הוא:

basicsys@mars~/lec3>cat F1
1
2
3
4
5
6
7
8
   הפקודה הבאה מפעילה את התכנית P1 כך שבמקום שתקבל
 קלט מהמשתמש היא תקבל אותו מהקובץ F1. במקרה הזה כל
      פקודת read x שבתוך לולאת ה- while תקלוט שורה
 נוספת מהקובץ. כאשר יגיע סוף הקובץ פקודת ה- read x
               תחזיר false ולולאת ה- while תסתיים.

basicsys@mars~/lec3>P1<F1
5
6
7
8
```

הרצאה מספר 4

הפקודה cut

בפקודה cut אפשר להשתמש בכמה צורות כפי שיתואר בהמשך.

cut -c<תחום> <קובץ>

הפקודה cut -c<תחום> <קובץ> מציגה את השורות בקובץ כאשר מכל שורה מוצגים רק התווים שנמצאים בתחום. מספור התווים מתחיל מ- 1.

כאשר מבקשים בתחום תו שלא נמצאת בשורה היא לא מוצגת. לדוגמה כאשר מבקשים תו 10 בשורה שיש בה רק 8 תווים התו לא מוצג.

דוגמאות לתחום תווים:

2 תו מספר 2
2-4 תווים 2 3 ו- 4
3- החל מתו מספר 3 ועד סוף השורה
3,5-7,9- תו מספר 3 תווים 5 6 7 והחל מתו מספר 9 ועד סוף השורה

לדוגמה, נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec4>cat F1
1234567891234567 89
abcdefghiabcdefghi
987654321
```

לאחר ביצוע הפקודה:

```
basicsys@mars~/lec4>cut -c2 F1
```

מתקבל הפלט:

```
2
b
8
```

לאחר ביצוע הפקודה:

```
basicsys@mars~/lec4>cut -c5- F1
```

מתקבל הפלט:

```
56789123456789
efghiabcdefghi
54321
```

לאחר ביצוע הפקודה:

```
basicsys@mars~/lec4>cut -c2,5- F1
                                        מתקבל הפלט:
256789123456789
befghiabcdefghi
854321
```

cut -d"<תו מפריד>" -f<תחום> <קובץ>

הפקודה cut -d"<תו מפריד>" -f<תחום> <קובץ> מציגה את השורות בקובץ כאשר מכל שורה מוצגים רק השדות שנמצאים בתחום, כאשר שדה הוא רצף של תווים שאינו מכיל את התו המפריד. מספור השדות מתחיל מ- 1.

כאשר מבקשים בתחום שדה שלא נמצא בשורה, השדה לא מוצג.

כאשר התו המפריד לא נמצא בשורה מוצגת כל השורה, אלא אם נעשה שימוש באופציה s- שגורמת לכך ששורות שלא נמצא בהן התו המפריד לא מוצגות.

בין השדות בשורת הפלט מוצג התו המפריד, אלא אם נעשה שימוש באופציה: "--output-delimiter="מחרוזת שגורמת לכך שבין השדות בשורת הפלט תוצג המחרוזת.

השימוש בתחום הוא בדומה לתחום של עמודות, לדוגמה:

```
2  שדה מספר 2
2-4 שדות 2 3 ו- 4
-3  החל משדה מספר 3 ועד סוף השורה
3,5-7,9- שדה מספר 3 שדות 5 6 7 והחל משדה מספר 9
                        ועד סוף השורה
```

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec4>cat F1
a:b:c:d:e:f:g:h:i
1234:5678:3:44:55:66:77
A   :  B  :   C
```

לאחר ביצוע הפקודה:

```
basicsys@mars~/lec4>cut -d":" -f3 F1
                                        מתקבל הפלט:
c
3
   C
```

לאחר ביצוע הפקודה:

```
basicsys@mars~/lec4>cut -d":" -f1,3-5 F1
```

מתקבל הפלט:

```
a:c:d:e
1234:3:44:55
A   :   C
```

השורה האחרונה מראה שכאשר מבקשים שדות 5-4 שלא נמצאים בשורה הם לא מודפסים.

לאחר ביצוע הפקודה:

```
cut -d":" -f1,3-5 --output-delimiter="xyz" F1
```

מתקבל הפלט:

```
axyzcxyzdxyze
1234xyz3xyz44xyz55
A   xyz   C
```

בדוגמה הנ"ל השימוש באופציה --output-delimiter גורם לכך שבין השדות שבפלט במקום התו המפריד : מופיעה המחרוזת xyz.

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec4>cat F1
a:b:c:d:e:f:g:h:i
1234:5678:3:44:55:66:77
A   :  B  :   C
gg 1234 567 89
```

לאחר ביצוע הפקודה:

```
basicsys@mars~/lec4>cut -d" " -f3 F1
```

מתקבל הפלט:

```
a:b:c:d:e:f:g:h:i
1234:5678:3:44:55:66:77

567
```

בדוגמה הנ"ל שתי השורות הראשונות מודפסות במלואן כי התו המפריד רווח לא נמצא בשורות אלה. השורה השלישית ריקה כי השדה השלישי בשורה זו, שנמצא בין תו הרווח השני לתו הרווח השלישי, הוא מחרוזת ריקה.

לאחר ביצוע הפקודה:

```
basicsys@mars~/lec4>cut -s -d" " -f3 F1
```

מתקבל הפלט:

```
567
```

בדוגמה הנ"ל שתי השורות הראשונות לא מוצגות כי התו המפריד לא מופיע בהן והשתמשנו באופציה s-

חשוב לשים לב שהתו המפריד בין השדות חייב להיות תו בודד ולכן לאחר הפעלת הפקודה הבאה:

```
basicsys@mars~/lec4>cut -d":a" -f3 F1
```

מתקבלת הודעת השגיאה:

```
cut: the delimiter must be a single character
Try `cut --help' for more information.
```

התכנית הבאה קולטת מהמשתמש שם קובץ ומדפיסה את מספר השורות בקובץ, תוך שימוש בפקודה cut וללא שימוש בצורה wc -l < <קובץ>

```
basicsys@mars~/lec4>cat P1
read x
y=$(wc -l $x)
echo $y >|F2
cut -d" " -f1 F2
```

לאחר ביצוע הפקודה:

```
basicsys@mars~/lec4>P1
```

המשתמש מתבקש להקליד מחרוזת. נניח שהמשתמש מקליד:

```
F1
```

הפלט של התכנית הוא:

```
3
```

כי מספר השורות בקובץ F1 הוא 3.

<u>הפקודה head <רשימת קבצים></u>

הפקודה head <רשימת קבצים> מדפיסה את 10 השורות הראשונות בכל אחד מהקבצים. אם יש יותר מקובץ אחד ברשימה אז לפני כל קובץ מופיעה כותרת עם שמו. אם בקובץ יש פחות מ- 10 שורות יוצג כל תוכן הקובץ.

אופציות:
<מספר>- הצג רק <מספר> שורות ראשונות מהקובץ

q- כאשר יש יותר מקובץ אחד אל תציג את הכותרות של שמות הקבצים

לדוגמה, נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec4>cat F1
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

לאחר ביצוע הפקודה:

```
basicsys@mars~/lec4>head F1
```

יתקבל הפלט:

```
1
2
3
4
5
6
7
8
9
10
```

לאחר ביצוע הפקודה:

```
basicsys@mars~/lec4>head -3 F1
```

יתקבל הפלט:

```
1
2
```

```
3
```

נניח שתוכן הקובץ F2 הוא:

```
basicsys@mars~/lec4>cat F2
100
200
300
400
```

לאחר ביצוע הפקודה:

```
basicsys@mars~/lec4>head -3 F1 F2
```

יתקבל הפלט:

```
==> F1 <==
1
2
3

==> F2 <==
100
200
300
```

לאחר ביצוע הפקודה:

```
basicsys@mars~/lec4>head -3 -q F1 F2
```

יתקבל הפלט:

```
1
2
3
100
200
300
```

הפקודה <רשימת קבצים> tail

הפקודה <רשימת קבצים> tail מדפיסה את 10 השורות האחרונות בכל אחד מהקבצים. אם יש יותר מקובץ אחד ברשימה אז לפני כל קובץ מופיעה כותרת עם שמו. אם בקובץ יש פחות מ- 10 שורות יוצג כל תוכן הקובץ.

אופציות:

<מספר>- הצג רק <מספר> שורות אחרונות מהקובץ

q- כאשר יש יותר מקובץ אחד אל תציג את הכותרות של שמות הקבצים

לאחר ביצוע הפקודה:

```
basicsys@mars~/lec4>tail F1
```

יתקבל הפלט:

```
6
7
8
9
10
11
12
13
14
15
```

לאחר ביצוע הפקודה:

```
basicsys@mars~/lec4>tail -3 F1
```

יתקבל הפלט:

```
13
14
15
```

התכנית הבאה קולטת מהמשתמש שם קובץ ומספר (בהמשך נקרא להם קובץ1 ומספר1) ומדפיסה לפלט את שורה מספר 1 בקובץ1.

```
basicsys@mars~/lec4>cat P1
read x
read y
head -$y $x >| tmp
tail -1 tmp
```

נניח שלאחר הפעלת התכנית P1 המשתמש הקליד:

```
F1
5
```

אז הפלט של התכנית הוא השורה החמישית בקובץ F1:

```
5
```

התכנית הנ"ל עובדת נכון אם מספר1 קטן או שווה למספר השורות בקובץ1. נשפר את התכנית כך שבמקרה שמספר1 גדול ממספר השורות בקובץ התכנית תדפיס הודעת שגיאה.

```
basicsys@mars~/lec4>cat P1
read x
read y
n=$(wc -l < $x)
if [ $y -gt $n ]
then
  echo "You requested line $y but the file has $n lines"
  exit
fi
head -$y $x >| tmp
tail -1 tmp
```

ננית שלאחר הפעלת התכנית P1 המשתמש הקליד:

```
F1
50
```

אז הפלט של התכנית הוא:

```
You requested line 50 but the file has 15 lines
```

התכנית הנ"ל משתמשת בפקודה exit.

<u>הפקודה exit</u>

הפקודה exit גורמת לסיום התכנית.

התכנית הבאה מבקשת מהמשתמש שם קובץ שמכיל מספרים ומדפיסה לפלט את סכום המספרים בקובץ:

```
basicsys@mars~/lec4>cat P1
read x
n=$(wc -l < $x)
sum=0
while [ $n -ge 1 ]
do
 head -$n $x >| tmp
 z=$(tail -1 tmp)
 sum=$[$sum+$z]
 n=$[$n-1]
done
echo $sum
```

ננית שתוכן הקובץ F2 הוא:

```
100
200
300
400
```

נניח שלאחר הפעלת התכנית P1 המשתמש מקליד:

```
F2
```

אז הפלט של התכנית הוא:

```
1000
```

המבנה <קובץ> < done בסוף לולאת while

כאשר משתמשים ב <קובץ> < done בסוף לולאת while אז הקלט לפקודות (שמבקשות קלט) בתוך לולאת ה- while מגיע מהקובץ במקום מהמקלדת.

לדוגמה, התכנית הבאה מדפיסה את 3 השורות הראשונות בקובץ F2

```
basicsys@mars~/lec4>cat P1
n=1
while [ $n -le 3 ]
 do
   read x
   echo $x
   n=$[$n+1]
 done<F2
```

לאחר הפעלת התכנית P1 מתקבל הפלט:

```
100
200
300
```

התכנית הבאה משתמשת במבנה הנ"ל כדי לבצע את המשימה של התכנית הקודמת, דהינו התכנית מבקשת מהמשתמש שם קובץ שמכיל מספרים ומדפיסה לפלט את סכום המספרים בקובץ:

```
basicsys@mars~/lec4>cat P1
read x
sum=0
while read z
do
 sum=$[$sum+$z]
done< $x
echo $sum
```

נניח שלאחר הפעלת התכנית P1 המשתמש מקליד:

```
F2
```

אז הפלט של התכנית הוא:

```
1000
```

הפקודה continue

בפקודה continue משתמשים בתוך לולאה והיא גורמת לכך שהתכנית תחזור לתחילת הלולאה שבתוכה הופעלה הפקודה continue.

לדוגמה, התכנית הבאה קולטת מספרים מהמשתמש (עד שהמשתמש מקליד Ctrl+d) ומדפיסה את סכום כל המספרים שהמשתמש הקליד שאינם שווים למספר 30.

```
basicsys@mars~/lec4>cat P1
sum=0
while read x
do
 if [ $x -eq 30 ]
  then
   continue
 fi
 sum=$[$sum+$x]
done
echo $sum
```

נניח שלאחר הפעלת התכנית המשתמש הקליד את סדרת המספרים הבאה ולאחריה Ctrl+d:

```
10
30
20
30
5
```

אז הפלט של התכנית הוא:

```
35
```

הפקודה break

בפקודה break משתמשים בתוך לולאה והיא גורמת לכך שהתכנית תצא מהלולאה שבתוכה הופעלה הפקודה break.

במידה והלולאה הפנימית שבתוכה הופעלה הפקודה break נמצאת בתוך לולאה חיצונית הפקודה יוצאת רק מהלולאה הפנימית ולא יוצאת מהלולאה החיצונית.

לדוגמה, התכנית הבאה קולטת מספרים מהמשתמש עד שהמשתמש מקליד Ctrl+d או שהמשתמש מקליד את המספר 30

ומדפיסה את סכום כל המספרים שהמשתמש הקליד (לא כולל המספר 30).

```
basicsys@mars~/lec4>cat P1
sum=0
while read x
do
 if [ $x -eq 30 ]
  then
   break
 fi
 sum=$[$sum+$x]
done
echo $sum
```

נניח שלאחר הפעלת התכנית המשתמש הקליד את סדרת המספרים הבאה:

```
10
15
30
```

אז הפלט של התכנית הוא:

```
25
```

התכנית הבאה מדפיסה שורה אחת עבור כל מספר שמופיע בקובץ F1 שמכילה את המספר, לאחריו תו רווח ולאחריו מספר שמציין את מספר החזרות של המספר בקובץ F2.

```
basicsys@mars~/lec4>cat P1
while read x
do
 m=0
 while read y
 do
  if [ $x -eq $y ]
  then
   m=$[$m+1]
  fi
 done<F2
 echo "$x $m"
done<F1
```

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec4>cat F1
10
20
```

```
15
```

וננית שתוכן הקובץ F2 הוא:

```
basicsys@mars~/lec4>cat F2
15
10
15
10
10
```

לאחר הפעלת התכנית P1 מתקבל הפלט:

```
10 3
20 0
15 2
```

לולאת for

המבנה של לולאת for הוא:

```
for  <שם משתנה>  in  <רשימה>
   do
     סדרת פקודות

   done
```

בסיבוב הראשון בלולאה הערך שהמשתנה מקבל הוא האיבר הראשון ברשימה, בסיבוב השני בלולאה הערך שהמשתנה מקבל הוא האיבר השני ברשימה וכן הלאה.. עד שהרשימה מסתיימת.

לדוגמה, בתכנית P1 הבאה, בסיבוב הראשון בלולאה הערך של המשתנה x הוא aa בסיבוב השני הערך שלו הוא bb וכן הלאה...

```
basicsys@mars~/lec4>cat P1
for x in aa bb cc
 do
  echo $x $x
 done
```

לאחר הפעלת התכנית P1 מתקבל הפלט:

```
aa aa
bb bb
cc cc
```

התכנית הבאה שקולה לתכנית הקודמת כי הסורק מצמצם את הרווחים ומחליף את aa bb cc ב- aa bb cc

```
basicsys@mars~/lec4>cat P1
for x in aa    bb     cc
 do
  echo $x $x
 done
```

לאחר הפעלת התכנית P1 מתקבל הפלט:

```
aa aa
bb bb
cc cc
```

התכנית הבאה מקבלת רשימה בעלת איבר אחד (הגרשיים הכפולים גורמים לכך שהסורק לא משנה את המחרוזת ומשאיר את הרווחים) ולכן התכנית נכנסת ללולאה רק פעם אחת ומדפיסה רק שורה אחת.

```
basicsys@mars~/lec4>cat P1
for x in "aa  bb        cc"
 do
  echo $x $x
 done
```

לאחר הפעלת התכנית P1 מתקבל הפלט:

```
aa bb cc aa bb cc
```

בשורה הנ"ל יש צמצום רווחים בגלל שהסורק מחליף את $x בערך המשתנה x לאחר צמצום רווחים.

התכנית הבאה זהה לקודמת פרט לשימוש ב- "$x" במקום ב- $x.

```
basicsys@mars~/lec4>cat P1
for x in "aa  bb        cc"
 do
  echo "$x" "$x"
 done
```

לאחר הפעלת התכנית P1 מתקבל הפלט:

```
aa  bb        cc aa  bb        cc
```

התכנית הבאה זהה לקודמת פרט להוספת רווחים בין ה- "$x" הראשון לשני. מאחר והסורק מצמצם את הרווחים האלה הפלט של התכנית הבאה זהה לפלט של התכנית הקודמת.

```
basicsys@mars~/lec4>cat P1
for x in "aa  bb        cc"
 do
  echo "$x"      "$x"
 done
```

לאחר הפעלת התכנית P1 מתקבל הפלט:

```
aa  bb        cc aa  bb        cc
```

בתכנית הבאה הגרשיים הכפולים גורמים לכך שהסורק לא מצמצם רווחים.

```
basicsys@mars~/lec4>cat P1
for x in "aa  bb        cc"
 do
  echo "$x      $x"
 done
```

לאחר הפעלת התכנית P1 מתקבל הפלט:

```
aa  bb        cc      aa  bb        cc
```

<u>שרשור מחרוזות</u>

נניח שיש לנו שתי מחרוזות, אחת ב- <משתנה1> והשניה ב- <משתנה2> כדי לקבל את השרשור של שתי המחרוזות נשתמש במבנה הבא: "<משתנה2><משתנה1>$"

לדוגמה, לאחר הפעלת סדרת הפקודות הבאה:

```
basicsys@mars~/lec4>x=abc

basicsys@mars~/lec4>y=de

basicsys@mars~/lec4>z="$x$y"

basicsys@mars~/lec4>echo "$z"
```

יתקבל הפלט:

```
abcde
```

לאחר הפעלת סדרת הפקודות הבאה:

```
basicsys@mars~/lec4>x="abcd     ef"

basicsys@mars~/lec4>y="12ab  g"

basicsys@mars~/lec4>z="$x$y"

basicsys@mars~/lec4>echo "$z"
```

יתקבל הפלט:

```
abcd     ef12ab  g
```

שימוש בלולאת for לקריאת שורות מקובץ

התכנית הבאה, מבצעת את המשימה של התכנית שהוצגה קודם, בעזרת שימוש בלולאת for במקום לולאת while.

```
basicsys@mars~/lec4>cat P1
for x in $(cat F1)
do
 m=0
 for y in $(cat F2)
 do
  if [ $x -eq $y ]
  then
   m=$[$m+1]
  fi
 done
 echo "$x $m"
done
```

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec4>cat F1

10
20
15
```

ונניח שתוכן הקובץ F2 הוא:

```
basicsys@mars~/lec4>cat F2
15
10
15
10
10
```

לאחר הפעלת התכנית P1 מתקבל הפלט:

```
10 3
20 0
15 2
```

בדוגמה הנ"ל הסורק מחליף את $(cat F1) ברשימה שמורכבת מתוכן הקובץ F1. מאחר ואין חשיבות לשמירה על מבנה השורות בקובץ במקרה זה התכנית עובדת.

נניח שאנחנו רוצים להדפיס את תוכן הקובץ F1 בעזרת התכנית הבאה:

```
basicsys@mars~/lec4>cat P1
c=0
for x in $(cat F1)
  do
   echo "$x"
   c=$[$c+1]
  done
echo $c
```

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec4>cat F1
10  20    30
10
40 50
```

לאחר הפעלת התכנית P1 מתקבל הפלט:

```
10
20
30
10
40
50
6
```

מאחר והסורק מחליף את $(cat F1) ברשימה בעלת 6 איברים 50 40 10 30 20 10 התכנית תיכנס ללולאה 6 פעמים ותדפיס כל פעם מספר אחד מהרשימה, והתוצאה אינה זהה לתוכן הקובץ.

נסתכל על התכנית הבאה:

```
basicsys@mars~/lec4>cat P1
c=0
for x in "$(cat F1)"
  do
   echo "$x"
   c=$[$c+1]
  done
echo $c
```

לאחר הפעלת התכנית P1 מתקבל הפלט:

```
10  20    30
10
40 50
1
```

התכנית הנ"ל אכן מציגה את תוכן הקובץ אבל המספר 1 שמודפס בסוף מציין שהתכנית נכנסה ללולאה פעם אחת בלבד. לתכנית הנ"ל עדין יש חסרון שהיא לא מאפשרת לטפל בנפרד בכל סיבוב בלולאה בשורה מהקובץ.
לפתרון הבעיה נשתמש בתכנית הבאה שבכל סיבוב בלולאה מכניסה שורה מהקובץ למשתנה x.

```
basicsys@mars~/lec4>cat P1
c=$(wc -l < F1)
for i in $(seq $c)
  do
   head -$i F1 >| F6
   x=$(tail -1 F6)
   echo "$x"
  done
```

לאחר הפעלת התכנית P1 מתקבל הפלט:

```
10  20    30
10
40 50
```

הפקודה tr

הפקודה <מחרוזת2> <מחרוזת1> tr

מבקשת קלט, ומדפיסה לפלט כל שורת קלט שהיא מקבלת לאחר ביצוע החלפה של כל תו במחרוזת1 בתו המתאים לו במחרוזת2.

אם מחרוזת 2 קצרה ממחרוזת1, החלפת התווים מתבצעת כאילו התו האחרון במחרוזת2 משוכפל כך ששתי המחרוזות שוות בגודלן. במילים אחרות כל תו במחרוזת1 שאין לו תו מתאים במחרוזת2 יוחלף בתו האחרון שבמחרוזת2.

אם מחרוזת2 ארוכה ממחרוזת1 מתעלמים מהתווים שבמחרוזת2 שאין להם תו מתאים במחרוזת1.

אופציות:

s- לאחר החלפת התווים שבמחרוזת1 בתווים המתאימים להם במחרוזת2 בצע צמצום של רצפים של תווים שמופיעים במחרוזת2. במידה ומחרוזת2 לא קיימת צמצם רצפים של תווים שמופיעים במחרוזת1.

d- הדפס לפלט את כל התווים בקלט שלא מופיעים במחרוזת1 (במקרה זה הפקודה מקבלת רק פרמטר אחד).

c- החלף את כל התווים שלא מופיעים במחרוזת1

לדוגמה, לאחר הפעלת הפקודה הבאה:

```
basicsys@mars~/lec4>tr ab xy
```

המשתמש מתבקש להקליד שורת קלט, נניח שהמשתמש מקליד:
```
abcdab
```
אז מתקבל הפלט:
```
xycdxy
```

והמשתמש מתבקש להקליד שורה נוספת, נניח שהוא מקליד:
```
12aabba34xxyy22ab
```
אז מתקבל הפלט:
```
12xxyyx34xxyy22xy
```
והמשתמש מתבקש להקליד שורה נוספת, נניח שהוא מקליד Ctrl+d ואז הפקודה מסתיימת.

הפקודה הבאה מחליפה אותיות אנגליות קטנות בגדולות, לאחר הפעלת הפקודה:
```
basicsys@mars~/lec4>tr a-z A-Z
```

נניח שהמשתמש מקליד:
```
ssasadas
```
אז הפלט שמתקבל הוא:
```
SSASADAS
```

בפקודה הבאה הקלט מגיע מהקובץ F1 במקום מהמקלדת.

נניח שתוכן הקובץ F1 הוא:
```
asfdsa  asd
afd 123
dafd  adsa
```
לאחר הפעלת הפקודה:
```
basicsys@mars~/lec4>tr a-z A-Z <F1
```
מתקבל הפלט:
```
ASFDSA  ASD
AFD 123
DAFD  ADSA
```

לאחר הפעלת הפקודה הבאה הקובץ F2 מכיל את התוכן של הקובץ F1 לאחר הפיכת אותיות קטנות לגדולות:

```
basicsys@mars~/lec4>tr a-z A-Z <F1 >|F2
```

התוכן של הקובץ F2 לאחר הפעלת הפקודה הנ"ל הוא:

```
basicsys@mars~/lec4>cat F2
ASFDSA  ASD
AFD 123
DAFD  ADSA
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec4>tr a-zA-Z0-9 "*"
```

נניח שהמשתמש מקליד:

```
assasa1234fsdkfa
```

אז הפלט שמתקבל הוא:

```
**********
```

והמשתמש מתבקש להקליד שורה נוספת, נניח שהוא מקליד:

```
afsadf   sadfsakfsa
```

אז הפלט שמתקבל הוא:

```
******   **********
```

הפקודה הבאה מראה שכאשר רוצים להשתמש בתו מיוחד כמו למשל ~ צריך להקדים לו \.

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec4>tr "\~" "*"
```

נניח שהמשתמש מקליד:

```
~~~~~~
```

אז הפלט שמתקבל הוא:

```
******
```

הפקודה הבאה משתמשת באופציה s- לצמצום רצפים של תווים. לאחר הפעלת הפקודה:

```
basicsys@mars~/lec4>tr -s ab
```

נניח שהמשתמש מקליד:

```
aaabbbbaaaabbbb
```

אז הפלט שמתקבל הוא:

```
abab
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec4>tr -s ab 12
```

נניח שהמשתמש מקליד:

```
a11112222bbb111122221112222
```

אז הפלט שמתקבל הוא:

```
121212
```

בפקודה הבאה מחליפים את כל התווים שלא מופיעים במחרוזת 1 בתו *.

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec4>tr -c "a-zA-Z0-9\n" "*"
```

נניח שהמשתמש מקליד:

```
abc&&@@xy12$#@!cd
```

אז הפלט שמתקבל הוא:

```
abc****xy12****cd
```

בפקודה הבאה מוחקים את כל התווים שמופיעים במחרוזת 1:

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec4>tr -d abc
```

נניח שהמשתמש מקליד:

```
aaabbbcccabcxaaabbbcccy
```

אז הפלט שמתקבל הוא:

```
xy
```

הפקודה הבאה מוחקת את כל התווים שלא מופיעים במחרוזת 1.

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec4>tr -dc "a-zA-Z0-9\n"
```

נניח שהמשתמש מקליד:

```
ab%^$#@!cd^%$12@@@!!!   $$$$ef
```

אז הפלט שמתקבל הוא:

```
abcd12ef
```

נניח שתוכן הקובץ F1 הוא:

```
asfdsa   asd
afd 123
dafd   adsa
```

הפקודה הבאה מציגה את תוכן הקובץ F1 לאחר החלפת כל תוו רווח בתו \n.

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec4>tr " " "\n" <F1
```

מתקבל הפלט:

```
asfdsa


asd
afd
123
dafd


adsa
```

הפקודה הבאה מציגה את תוכן הקובץ F1 לאחר צמצום רצפים של רווחים והחלפת כל תוו רווח בתו \n. אם אין רווחים בתחילת השורה הראשונה בקובץ הפקודה מדפיסה את המילים של הקובץ כך שכל מילה מופיעה בשורה נפרדת.

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec4>tr -s " " "\n" <F1
```

מתקבל הפלט:

```
asfdsa
asd
afd
123
dafd
adsa
```

נניח שתוכן הקובץ F1 הוא:

```
  asfdsa   asd
afd 123
  dafd   adsa
```

לאחר הפעלת הפקודה הקודמת:

```
basicsys@mars~/lec4>tr -s " " "\n" <F1
```

מתקבל הפלט:

```
asfdsa
asd
afd
123
dafd
adsa
```

בפלט הנ"ל ישנה שורה ריקה אחת בהתחלה.

<u>הדפסת מילים בקובץ כאשר כל מילה מופיעה בשורה נפרדת</u>

התכנית הבאה מציגה את המילים שבקובץ F1 כאשר כל מילה מופיעה בשורה נפרדת והיא עובדת נכון גם אם יש רווחים בתחילת השורה הראשונה בקובץ.

```
basicsys@mars~/lec4>cat P1
echo " " >| tmp
cat F1 >> tmp
tr -s " " "\n" <tmp >| tmp1
n=$(wc -l < tmp1)
n=$[$n-1]
tail -$n tmp1
```

לאחר הפעלת התכנית P1 מתקבל הפלט:

```
asfdsa
asd
afd
123
dafd
adsa
```

גם התכנית הבאה מציגה את המילים שבקובץ F1 כאשר כל מילה בשורה נפרדת והיא עובדת נכון גם אם יש רווחים בתחילת השורה בקובץ.

```
basicsys@mars~/lec4>cat P1
echo $(cat F1) >| tmp
tr " " "\n" < tmp
```

לאחר הפעלת התכנית P1 מתקבל הפלט:

```
asfdsa
asd
afd
123
dafd
adsa
```

הדפסת תוכן של קובץ לאחר צמצום הרווחים שבשורות הקובץ

נניח שתוכן הקובץ F1 הוא:

```
  asfdsa   asd
afd 123
  dafd   adsa
```

התכנית הבאה מציגה את תוכן הקובץ F1 לאחר צמצום הרווים שבשורות הקובץ.

```
basicsys@mars~/lec4>cat P1
while read x
do
 echo $x
done<F1
```

לאחר הפעלת התכנית P1 מתקבל הפלט:

```
asfdsa asd
afd 123
dafd adsa
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec4>P1>|F2
```

תוכן הקובץ F2 הוא:

```
basicsys@mars~/lec4>cat F2
asfdsa asd
afd 123
dafd adsa
```

ולכן אפשר להשתמש בשיטה הנ"ל כדי לקבל קובץ F2 שמכיל את התוכן של הקובץ F1 לאחר צמצום הרווחים שבשורות הקובץ.

הרצאה מספר 5

הפעלת פקודות ללא שם קובץ כפרמטר

חלק גדול מהפקודות שמצפות לשם קובץ כפרמטר, ניתנות להפעלה גם ללא קבלת שם הקובץ כפרמטר. במקרה זה תוכן הקובץ שעליו הפקודות פועלות מגיע מהקלט.

לדוגמה, לאחר הפעלת הפקודה wc -l ללא שם קובץ:

```
basicsys@mars~/lec5>wc -l
```

המשתמש מתבקש להקליד קלט, נניח שהמשתמש מקליד את שלוש השורות הבאות ולאחר מכן מקליד Ctrl+d:

```
abcd
aa
bb
```

אז הפלט של התכנית הוא:

```
3
```

הפקודה wc -l פעלה על הקובץ שאת תוכנו המשתמש הקליד ומאחר והמשתמש הקליד 3 שורות התוצאה של הפקודה היא 3. הפקודה לא מדפיסה את שם הקובץ שעליו היא פועלת כי היא לא קבלה שם כזה והיא פעלה על הקלט.

נניח שתוכן הקובץ F1 הוא:

```
abc   aa   abc
abc  bb    aa
bb xyz    aa
bb
```

נפעיל את הפקודה הנ"ל כך שהקלט יגיע מהקובץ F1 במקום מהמקלדת.

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec5>wc -l < F1
```

מתקבל הפלט:

```
4
```

מאחר והפקודה wc -l לא מקבלת את הקובץ F1 כפרמטר שם הקובץ F1 לא מודפס.

לעומת זאת, לאחר הפעלת הפקודה:

```
basicsys@mars~/lec5>wc -l F1
```

מתקבל הפלט:

```
4 F1
```

כי בפקודה הנ"ל שם הקובץ F1 מועבר כפרמטר

לפקודה wc -l.

הפקודה הבאה מפעילה את הפקודה cut ללא שם קובץ כפרמטר.

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec5>cut -c2,4
```

המשתמש מתבקש להקליד קלט, נניח שהמשתמש מקליד את שתי השורות הבאות ולאחר מכן מקליד Ctrl+d:

```
abcdefg
123456
```

אז הפלט של התכנית הוא:

```
bd
24
```

הפקודה cut -c2,4 לא מקבלת שם קובץ כפרמטר ולכן היא פועלת על הקובץ שאת תוכנו המשתמש מקליד.

הפקודה הבאה מפעילה את הפקודה cat ללא שם קובץ כפרמטר.

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec5>cat
```

המשתמש מתבקש להקליד שורת קלט, נניח שהמשתמש מקליד:

```
abcde
```

אז הפקודה מדפיסה לפלט:

```
abcde
```

והמשתמש מתבקש להקליד שורת קלט נוספת, נניח שהמשתמש מקליד:

```
xyz
```

אז הפקודה מדפיסה לפלט:

```
xyz
```

והמשתמש מתבקש להקליד שורת קלט נוספת, נניח שהמשתמש מקליד Ctrl+d והפקודה מסתיימת.

הפקודה cat ללא שם קובץ פועלת באופן שונה מהפקודות wc ו- cut בכך שהיא מדפיסה לפלט את השורות שהמשתמש מקליד מיד בלי לחכות שהמשתמש יקליד את כל שורות הקובץ.

הפקודה הבאה מכניסה את כל הפלט של הפקודה cat ללא פרמטרים לתוך המשתנה x ולכן לא נראה את הפלט של הפקודה cat על המסך.

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec5>x=$(cat)
```

נניח שהמשתמש מקליד את 3 השורות הבאות ולאחר מכן המשתמש מקליד Ctrl+d

```
adsfa
123456
aabbcc
```

אז לפקודה אין פלט והתוכן של המשתנה x הוא:

```
basicsys@mars~/lec5>echo "$x"
adsfa
123456
aabbcc
```

<u>שימוש ב - כשם קובץ שמועבר כפרמטר</u>

כאשר פקודה מקבלת כפרמטר שם קובץ - הקובץ שאותו היא מיצג הוא הקלט הסטנדרטי.

לדוגמה, נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec5>cat F1
ab
cd
12
```

ונניח שתוכן הקובץ F2 הוא:

```
basicsys@mars~/lec5>cat F2
ef
gh
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec5>cat F1 - F2
```

יוצג תוכן הקובץ F1 לאחר מכן המשתמש יתבקש להקליד קלט והקלט שהמשתמש יקליד יודפס עד שהמשתמש יקליד Ctrl+d ולאחר מכן יודפס תוכן הקובץ F2.

לכן לאחר הפעלת הפקודה הנ"ל מתקבל הפלט (שהוא תוכן הקובץ F1):

```
ab
cd
12
```

והמשתמש מתבקש להקליד קלט, נניח שהמשתמש מקליד את השורה:

```
xy
```

אז מתקבל הפלט:

```
xy
```

והמשתמש מתבקש להקליד קלט נוסף, נניח שהמשתמש מקליד Ctrl+d אז מתקבל הפלט (שהוא תוכן הקובץ F2):

```
ef
gh
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec5>wc -l F1 - F2
```

מתקבל הפלט:

```
      3 F1
```

והמשתמש מתבקש להקליד קלט, נניח שהמשתמש מקליד את 4 השורות הבאות ולאחר מכן Ctrl+d:

```
11
22
33
44
```

אז מתקבל הפלט:

```
      4 -
      2 F2
      9 total
```

נניח שהקובץ F3 מכיל 3 שורות, לאחר הפעלת הפקודה:

```
basicsys@mars~/lec5>wc -l F1 - F2 < F3
```

מתקבל הפלט:

```
 3 F1
 3 -
 2 F2
 8 total
```

<u>הפקודה sort</u>

הפקודה <שם קובץ> sort מדפיסה לפלט את תוכן הקובץ לאחר מיון השורות.

אופציות:

n- מיין את לפי ערך מספרי
r- מיין בסדר הפוך
u- צמצם שורות זהות

לדוגמה, נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec5>cat F1
10
300
5
2000
```

לאחר ביצוע הפקודה:

```
basicsys@mars~/lec5>sort -n F1
```

יתקבל הפלט:

```
5
10
300
2000
```

לאחר ביצוע הפקודה:

```
basicsys@mars~/lec5>sort -n -r F1
```

יתקבל הפלט:

```
2000
300
10
5
```

נניח שתוכן הקובץ F2 הוא:

```
basicsys@mars~/lec5>cat F2
30
20
10
30
40
10
10
100
```

לאחר ביצוע הפקודה:

```
basicsys@mars~/lec5>sort -n -u F2
```

יתקבל הפלט:

```
10
20
30
40
100
```

ברירת המחדל של sort היא למיין לפי הערך של התווים בשורה ולא לפי ערך מספרי. ערכי התווים נקבעים לפי הקוד שלהם במערכת. אם המערכת עובדת לפי קוד ascii אז הערך לפי הקוד הזה.
אצלנו במערכת המשתנה שקובע את הקוד של התווים נקרא LANG והערך ההתחלי שלו הוא:

```
basicsys@mars~/lec5>echo $LANG
en_US.UTF-8
```

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec5>cat F1
AAB
cdd
CDD
aab
```

לאחר ביצוע הפקודה sort ללא פרמטרים:

```
basicsys@mars~/lec5>sort F1
```

מתקבל הפלט:

```
aab
AAB
cdd
CDD
```

הפקודה הבאה משנה את הערך של המשתנה LANG כך שקידוד התווים יהיה לפי קוד ה- ascii שלהם:

```
basicsys@mars~/lec5>LANG=ascii
```

נבצע שוב את הפקודה sort ללא פרמטרים:

```
basicsys@mars~/lec5>sort F1
```

הפלט שמתקבל עכשיו הוא:

```
AAB
CDD
aab
cdd
```

הסיבה לפלט הנ"ל היא שלפי קוד ascii הקידוד של אותיות גדולות באנגלית קטן יותר מהקידוד של אותיות קטנות באנגלית. לדוגמה הקידוד של התו A לפי קוד ascii הוא: 10 והקידוד של התו a לפי קוד ascii

הוא: 97. לכן במיון לפי קוד ascii התו A מופיע לפני התו a.

הכללים הבאים מתקיימים בכל הקידודים:
בספרות 0-9 מתקיים:
0 < 1 < 2 < ... < 9
באותיות אנגליות קטנות מתקיים:
a < b < c < ... < z
באותיות אנגליות גדולות מתקיים:
A < B < C < ... < Z

כמו רוב הפקודות, גם הפקודה sort, כאשר מבצעים אותה ללא שם קובץ, היא פועלת על הקובץ שמגיע מהקלט.

לדוגמה לאחר הפעלת הפקודה:
```
basicsys@mars~/lec5>sort
   :(Ctrl+d ננית שהמשתמש מקליד את הקלט הבא (ובסיום
10
20
100
20
3
```
אז הפלט המתקבל הוא:
```
10
100
20
20
3
```

לאחר הפעלת הפקודה:
```
basicsys@mars~/lec5>sort -n
   :(Ctrl+d ננית שהמשתמש מקליד את הקלט הבא (ובסיום
10
20
100
20
3
```
אז הפלט המתקבל הוא:
```
3
10
20
20
100
```

pipeline

כאשר משתמשים במבנה הבא (שנקרא pipeline):

<פקודה 1> | <פקודה 2>

המשמעות היא שהפלט הרגיל של פקודה 1 הוא הקלט של פקודה 2 (ולכן לא נראה אותו על המסך).

במקרה זה פלט השגיאות של פקודה 1 מופנה למסך.

לכן המבנה הנ"ל שקול לשתי הפקודות הבאות:

```
<פקודה 1> >| tmp
<פקודה 2> < tmp
```

כאשר משתמשים במבנה הבא:

<פקודה 1> 2>&1 | <פקודה 2>

המשמעות היא שהפלט הרגיל ופלט השגיאות של פקודה 1 הם הקלט של פקודה 2 (ולכן לא נראה אותם על המסך).

לכן המבנה הנ"ל שקול לשתי הפקודות הבאות:

```
<פקודה 1> >| tmp  2>&1
<פקודה 2> < tmp
```

לדוגמה, נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec5>cat F1
3000
10
300
5
2000
```

לאחר ביצוע שתי הפקודות הבאות:

```
basicsys@mars~/lec5>sort -n F1 >| temp

basicsys@mars~/lec5>tail -1 < tmp
```

מתקבל הפלט:

```
3000
```

ניתן להשיג את אותה תוצאה ללא שימוש בקובץ ביניים,
על ידי שימוש ב- pipeline.

לאחר הפעלת הפקודה הבאה:

```
basicsys@mars~/lec5>sort -n F1 | tail -1
```

מתקבל הפלט:

```
3000
```

נניח שהמשתנה x מכיל מחרוזת:

```
basicsys@mars~/lec5>x="ab cd ef gh"
```

שתי הפקודות הבאות מדפיסות את המילה הראשונה
והשלישית במחרוזת שבמשתנה x.

לאחר הפעלת שתי הפקודות הבאות:

```
basicsys@mars~/lec5>echo "$x" >| tmp

basicsys@mars~/lec5>cut -d" " -f1,3 < tmp
```

מתקבל הפלט:

```
ab ef
```

ניתן להשיג את אותה תוצאה ללא שימוש בקובץ ביניים,
על ידי שימוש ב- pipeline.

לאחר הפעלת הפקודה הבאה:

```
basicsys@mars~/lec5>echo "$x" |  cut -d" " -f1,3
```

מתקבל הפלט:

```
ab ef
```

נניח שתוכן הקובץ P1 הוא:

```
basicsys@mars~/lec5>cat P1
echo hi
echo1 abc
echo bye
echo2 def
```

לאחר הפעלת הפקודה הבאה:

```
basicsys@mars~/lec5>P1
```

מתקבל הפלט:

```
hi
./P1: line 2: echo1: command not found
bye
./P1: line 4: echo2: command not found
```

לאחר הפעלת הפקודה הבאה:

```
basicsys@mars~/lec5>P1 | wc -l
```

מתקבל הפלט:

```
./P1: line 2: echo1: command not found
./P1: line 4: echo2: command not found
2
```

הסיבה לפלט הנ"ל היא שהפקודה wc -l קיבלה כקלט שתי שורות שהן הפלט הרגיל של התכנית P1 והדפיסה לפלט את מספר השורות שהיא קיבלה, ולכן הדפיסה 2.
פלט השגיאות של התכנית P1 לא הגיע לפקודה wc -l כקלט והופנה למסך ולכן בתחילת הפלט הופיעו שתי השורות של פלט השגיאות של התכנית P1.

לאחר הפעלת הפקודה הבאה:

```
basicsys@mars~/lec5>P1 2>&1 | wc -l
```

מתקבל הפלט:

```
4
```

הסיבה לפלט הנ"ל היא שהפקודה wc -l קיבלה כקלט גם את הפלט הרגיל וגם את פלט השגיאות של התכנית P1 ולכן הדפיסה 4.

הפקודה הבאה מדפיסה את מספר השורות בקובץ הגדול ביותר מבין כל הקבצים בתיקיה הנוכחית שמתחילים באות F.

```
wc -l F* | cut -d" " -f2 | sort -n |  tail -1
```

כדי להבין איך הפקודה הנ"ל עובדת נפרק אותה לשלבים.
נניח שבתיקיה הנוכחית הקבצים שמתחילים באות F הם F1,F2,F3 שמכילים 5,8,2 שורות בהתאמה.

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec5>wc -l F*
```

מתקבל הפלט:

```
 5 F1
 8 F2
 2 F3
15 total
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec5>wc -l F* | cut -d" " -f2
```

מתקבל הפלט:

```
5
8
2
total
```

לאחר הפעלת הפקודה:

```
@mars~/lec5>wc -l F* | cut -d" " -f2 | sort -n
```

מתקבל הפלט:

```
total
2
5
8
```

לאחר הפעלת הפקודה:

```
wc -l F* | cut -d" " -f2 | sort -n |  tail -1
```

מתקבל הפלט:

```
8
```

לתכנית הנ"ל יש שני חסרונות:

1) היא מסתמכת על כך שבפלט של הפקודה sort -n המילה total תופיע לפני כל המספרים.

2) היא לא תעבוד נכון במקרים רבים כמו למשל בדוגמה הבאה: נניח שבתיקיה הנוכחית שמות הקבצים שמתחילים באות F הם F1,F2,F3 ומספר השורות שלהם הוא: 50,100,60 בהתאמה.

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec5>wc -l F*
```

מתקבל הפלט:

```
 50 F1
100 F2
 60 F3
210 total
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec5>wc -l F* | cut -d" " -f2
```

מתקבל הפלט:

```
50
F2
60
total
```

בסדרת המספרים הנ"ל מופיע F2 במקום המספר 100 שהוא גודל הקובץ F2. ולכן לאחר הפעלת הפקודה:

```
wc -l F* | cut -d" " -f2 | sort -n |  tail -1
```

מתקבל הפלט:

```
60
```

הפלט הנ"ל אינו נכון כי הפלט הנכון היה צריך להיות 100.

התכנית הבאה פותרת את הבעיה הנ"ל ומחשבת נכון את מספר השורות של הקובץ הגדול ביותר מבין הקבצים שנמצאים בתיקיה הנוכחית ומתחילים באות F:

```
basicsys@mars~/lec5>cat P1

x=$[$(echo $(wc -l F*) | wc -w)-2]
y=$(seq 1 2 $x)
z=$(echo $y | tr " " ",")
echo $(wc -l F*) | cut -d" " -f$z |  \
tr " " "\n" | sort -n | tail -1
```

בתכנית הנ"ל נעשה שימוש בסימן \ לציון שהמשך השורה מופיע בשורה הבאה.

נסביר את התכנית הנ"ל בשלבים.

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec5>echo $(wc -l F*)
```

מתקבל הפלט:

```
50 F1 100 F2 60 F3 210 total
```

בפלט הנ"ל מופיעים זוגות של מספר שורות ושם קובץ ובסוף מופיע הזוג: סה"כ מספר שורות והמילה total.

הפקודה הבאה מכניסה למשתנה x את מספר המילים ברשימה פחות 2:

```
x=$[$(echo $(wc -l F*) | wc -w)-2]
```

לאחר הפעלת הפקודה הנ"ל הערך של המשתנה x הוא 6.

```
basicsys@mars~/lec5>echo $x

6
```

הפקודה הבאה מכניסה למשתנה y סדרה של מספרים אי זוגיים החל מ- 1 ועד הערך של x. (מאחר והערך של x הוא זוגי אז הערך של x לא יכלל ברשימה).

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec5>y=$(seq 1 2 $x)
```

הערך של המשתנה y הוא:

```
basicsys@mars~/lec5>echo $y

1 3 5
```

לאחר הפעלת הפקודה הבאה:

```
basicsys@mars~/lec5>z=$(echo $y | tr " " ",")
```

הערך של המשתנה z הוא:

```
basicsys@mars~/lec5>echo $z

1,3,5
```

לאחר הפעלת הפקודה הבאה:

```
@mars~/lec5>echo $(wc -l F*) | cut -d" " -f$z
```

מתקבל הפלט:

```
5 8 2
```

הפלט הנ"ל מכיל את המספרים שמתאימים למספרי השורות בקבצים שמתחילים באות F.

לאחר הפעלת הפקודה הבאה:

```
echo $(wc -l F*) | cut -d" " -f$z | tr " " "\n" \
| sort -n
```

מתקבל הפלט:

```
2
5
8
```

הפלט הנ"ל מכיל את המספרים שמתאימים למספרי השורות בקבצים שמתחילים באות F ממוינים לפי סדר מספרי עולה.

לאחר הפעלת הפקודה הבאה:

```
echo $(wc -l F*) | cut -d" " -f$z | tr " " "\n" \
| sort -n | tail -1
```

מתקבל הפלט:

```
8
```

הפלט הנ"ל מתאר את מספר השורות הגדול ביותר מבין מספרי השורות של הקבצים שמתחילים באות F.

הפקודה uniq

הפקודה <שם קובץ> uniq מדפיסה לפלט את שורות הקובץ לאחר צמצום כל רצף שורות זהות לשורה אחת.

אופציות:

u- הצג רק שורות שלא מופיעות ברצף

d- הצג רק שורות שמופיעות ברצף של יותר משורה אחת

c- לפני כל שורה הצג את מספר ההופעות הרצופות שלה

לדוגמה, נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec5>cat F1
10
10
10
29
29
ab
ab
88
ab
ab
10
10
88
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec5>uniq F1
```

מתקבל הפלט:

```
10
29
ab
88
ab
10
88
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec5>uniq -u F1
```

מתקבל הפלט:

```
88
88
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec5>uniq -d F1
```

מתקבל הפלט:

```
10
29
ab
ab
10
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec5>uniq -c F1
```

מתקבל הפלט:

```
      3 10
      2 29
      2 ab
      1 88
      2 ab
      2 10
      1 88
```

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec5>cat F1
abc   aa   abc
abc  bb    aa
bb xyz    aa
bb
```

הפקודה הבאה מחשבת את מספר החזרות של כל מילה בקובץ F1.

לאחר הפעלת הפקודה:

```
cat F1 | tr -s " " "\n" | sort | uniq -c
```

מתקבל הפלט:

```
      3 aa
      3 abc
      3 bb
      1 xyz
```

הפקודה הנ"ל עובדת נכון אם בקובץ F1 אין רווחים בתחילת השורה הראשונה.

לדוגמה, נניח שתוכן הקובץ F2 הוא:

```
basicsys@mars~/lec5>cat F2
   abc   aa   abc
abc  bb    aa
bb xyz    aa
bb
```

לאחר הפעלת הפקודה הנ"ל על הקובץ F2:

```
cat F2 | tr -s " " "\n" | sort | uniq -c
```

יתקבל הפלט:

```
      1
      3 aa
      3 abc
      3 bb
      1 xyz
```

כדי להתגבר על הבעיה הנ"ל נשתמש בפקודה הבאה, שמסתמכת על השיטה שהוצגה בהרצאה 4 לפרק את הקובץ למילים כך שכל מילה מוצגת בשורה נפרדת.

לאחר הפעלת הפקודה:

```
echo $(cat F2) | tr " " "\n" | sort | uniq -c
```

מתקבל הפלט:

```
      3 aa
      3 abc
      3 bb
      1 xyz
```

<u>תנאים מורכבים</u>

ניתן ליצור תנאים מורכבים בעזרת השימוש במבנים הבאים.

```
<תנאי 1> -a <תנאי 2>
```

התנאי הנ"ל מתקיים אם ורק אם מתקיימים שני התנאים תנאי 1 ותנאי 2.

```
<תנאי 1> -o <תנאי 2>
```

התנאי הנ"ל מתקיים אם ורק אם מתקיים לפחות אחד מהתנאים תנאי 1 ותנאי 2.

```
! <תנאי 1>
```

התנאי הנ"ל מתקיים אם ורק אם תנאי 1 לא מתקיים.

ניתן להשתמש בסוגריים בהרכבת תנאים. כאשר משתמשים בסוגרים, צריך להשתמש ב- \(עבור סוגר שמאלי וב- \) עבור סוגר ימני, וצריך להוסיף רווח אחד לפחות מימין לסוגר ומשמאל ל- \.

לדוגמה,

\(<תנאי 1> -o <תנאי 2> \) -a \(<תנאי 3> -o <תנאי 4> \)

התנאי הנ"ל יהיה נכון אם מתקיים לפחות אחד מהתנאים תנאי 1 ותנאי 2 וגם מתקיים לפחות אחד מהתנאים תנאי 3 ותנאי 4.

התכנית הבאה בודקת את קיום התנאי:

```
(5 > 5) || (5 = 5)

basicsys@mars~/lec5>cat P1

if [ 5 -gt 5 -o 5 -eq 5 ]
then
 echo "(5 > 5) || (5 = 5)"
else
 echo "It is not true that: (5 > 5) || (5 = 5)"
fi
```

לאחר הפעלת התכנית הנ"ל מתקבל הפלט:

```
(5 > 5) || (5 = 5)
```

התכנית הבאה בודקת את קיום התנאי:

```
(5 > 5) && (5 = 5)

basicsys@mars~/lec5>cat P1

if [ 5 -gt 5 -a 5 -eq 5 ]
then
 echo "(5 > 5) && (5 = 5)"
else
 echo "It is not true that: (5 > 5) && (5 = 5)"
fi
```

לאחר הפעלת התכנית הנ"ל מתקבל הפלט:

```
It is not true that: (5 > 5) && (5 = 5)
```

התכנית הבאה בודקת את קיום התנאי:

```
((x > y) || (y > z)) && (( x > 0 ) || ( x <= 8 ))
```

```
basicsys@mars~/lec5>cat P1
```

```
x=10
y=5
z=8
if [ \( $x -gt $y -o $y -gt $z \) \
    -a \( $x -gt 0 -o $x -le 8 \) ]
then
 echo "((x > y) || (y > z)) && (( x > 0 ) || ( x
<= 8 ))"
else
 echo " It is not true that :"
 echo "((x > y) || (y > z)) && (( x > 0 ) || ( x
<= 8 ))"
fi
```

לאחר הפעלת התכנית הנ"ל מתקבל הפלט:

```
((x > y) || (y > z)) && (( x > 0 ) || ( x <= 8 ))
```

התכנית הבאה בודקת את קיום התנאי:

```
( (x > y) || (y > z) ) && !( x > 0 )
```

```
basicsys@mars~/lec5>cat P1
```

```
x=10
y=5
z=8
if [ \( $x -gt $y -o $y -gt $z \) \
    -a ! \( $x -gt 0 \) ]
then
 echo "( (x > y) || (y > z) ) && !( x > 0 )"
else
 echo "It is not true that:"
```

```
 echo "( (x > y) || (y > z) ) && !( x > 0 )"
fi
```

לאחר הפעלת התכנית הנ"ל מתקבל הפלט:

```
It is not true that:
( (x > y) || (y > z) ) && !( x > 0 )
```

ביטוים בסגנון glob

בכל הפקודות של bash שמקבלות רשימת קבצים/תיקיות כפרמטרים, כמו הפקודות ls,cat,cut,wc,head,tail וכו' משתמשים בביטוי בסגנון glob לתיאור רשימת הקבצים/תיקיות, והסורק מחליף את הביטוי ברשימת שמות הקבצים/תיקיות שמתאימים לביטוי.

לדוגמה, בפקודה:

```
ls F*
```

F* הוא ביטוי בסגנון glob. בדוגמה זו הסורק מחליף את הביטוי F* ברשימת כל הקבצים והתיקיות שנמצאים בתיקיה הנוכחית והשמות שלהם מתאימים לביטוי F*, דהינו מתחילים באות האנגלית F.

להלן פרוט החוקים של ביטוים בסגנון glob.

החוקים של ביטויים בסגנון גלוב (glob style)

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

קבוצת תווים בתוך סוגריים מרובעים מתאימה לתו בודד מתוך הקבוצה.

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

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

* מתאימה לרצף תווים כלשהו (כולל מחרוזת ריקה).

<u>חשוב לזכור</u>: לפי החוקים של ביטויים בסגנון גלוב, ההתאמה חיבת להיות לכל המחרוזת (התאמה לחלק מהמחרוזת אינה מספיקה). לדוגמה הביטוי a מתאים למחרוזת a בלבד (הוא לא מתאים למשל למחרוזת ab).

לדוגמה, נניח שלאחר הפעלת הפקודה:

```
basicsys@mars~/lec5/example1>ls
                                        מתקבל הפלט:
7e  8  ae18  ae19

                                    לאחר הפעלת הפקודה:

basicsys@mars~/lec5/example1>ls a*

                                        מתקבל הפלט:
ae18  ae19

                                    לאחר הפעלת הפקודה:

basicsys@mars~/lec5/example1>ls a

                                        מתקבל הפלט:
ls: a: No such file or directory

                                    לאחר הפעלת הפקודה:

basicsys@mars~/lec5/example1>ls ??

                                        מתקבל הפלט:
7e

                                    לאחר הפעלת הפקודה:

basicsys@mars~/lec5/example1>ls ???

                                        מתקבל הפלט:

                                    לאחר הפעלת הפקודה:

basicsys@mars~/lec5/example1>ls ???

                                        מתקבל הפלט:
ls: ???: No such file or directory

                                    לאחר הפעלת הפקודה:

basicsys@mars~/lec5/example1>ls ????
```

מתקבל הפלט:

```
ae18  ae19
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec5/example1>ls ae*
```

מתקבל הפלט:

```
ae18  ae19
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec5/example1>ls ae*8
```

מתקבל הפלט:

```
ae18
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec5/example1>ls 7e*
```

מתקבל הפלט:

```
7e
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec5/example1>ls 7e?
```

מתקבל הפלט:

```
ls: 7e?: No such file or directory
```

נניח שלאחר הפעלת הפקודה:

```
basicsys@mars~/lec5/example2>ls
```

מתקבל הפלט:

```
aaa  aaa111  abc  abcd  eb
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec5/example2>ls [a-e][a-e]
```

מתקבל הפלט:

```
eb
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec5/example2>ls [a-e][a-e][a-e]
```

מתקבל הפלט:

```
aaa  abc
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec5/example2>ls [a-e][a-e][a-e]*
```

מתקבל הפלט:

```
aaa  aaa111  abc  abcd
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec5/example2>ls [a-e][a-e][a-e]?
```

מתקבל הפלט:

```
abcd
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec5/example2>ls ab*
```

מתקבל הפלט:

```
abc  abcd
```

נניח שתוכן הקובץ abc הוא:

```
basicsys@mars~/lec5/example2>cat abc
I am file abc
```

ונניח שתוכן הקובץ abcd הוא:

```
basicsys@mars~/lec5/example2>cat abcd
I am file abcd
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec5/example2>cat ab*
```

הסורק מחליף את ab* ב- abc abcd ולכן הפקודה מציגה
את תוכן הקבצים abc ו- abcd ומתקבל הפלט:

```
I am file abc
I am file abcd
```

נניח שמבנה הקבצים בתיקיה lec5 הוא כפי שמתואר על
ידי הפעלת הפקודה tree הבאה:

```
basicsys@mars~/lec5>tree
.
|-- example1
|   |-- 7e
|   |-- 8
|   |-- ae18
|   `-- ae19
|-- example2
|   |-- aaa
|   |-- aaa111
|   |-- abc
|   |-- abcd
|   `-- eb
`-- lec5-2014.txt
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec5>ls ex*/a*
```

מתקבל הפלט:

```
example1/ae18  example2/aaa     example2/abc
example1/ae19  example2/aaa111  example2/abcd
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec5>ls ex*/??
```

מתקבל הפלט:

```
example1/7e  example2/eb
```

הרצאה מספר 6

הפקודה egrep

הפקודה egrep <ביטוי רגולארי> <קובץ>

מדפיסה לפלט את כל השורות בקובץ שמתאימות לביטוי הרגולארי לפי מערכת חוקים של ביטוים רגולאריים מורחבים.

לדוגמה, נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec6>cat F1

Jim Alchin 21 Seattle aef
Bill Gates 21 Seattle aef
Steve Jobs 21 Nevada aaa
Scott Neally 85 Los Angeles
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep 21 F1
```

מתקבל הפלט:

```
Jim Alchin 21 Seattle aef
Bill Gates 21 Seattle aef
Steve Jobs 21 Nevada aaa
```

הסיבה לפלט הנ"ל היא שכל השורות שהודפסו מתאימות לביטוי הרגולארי 21 שמשמעותו: המחרוזת 21 מופיעה באיזשהו מקום בשורה.

להלן תיאור מפורט של החוקים של התאמת שורות לביטוים רגולאריים מורחבים.

החוקים של ביטויים רגולאריים מורחבים (Extended Regular Expressions)

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

קבוצת תווים בתוך סוגריים מרובעים מתאימה לתו בודד מתוך הקבוצה.

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

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.

<u>חשוב לזכור</u>: 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 <u>מה שלפני</u> ה- *

+ מציין 1 או יותר חזרות של <u>מה שלפני</u> ה- +

? מציין 0 או 1 חזרות של <u>מה שלפני</u> ה- ?

{n} מציין בדיוק n חזרות של <u>מה שלפני</u> ה- {n}

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

{n,} מציין מספר חזרות גדול שווה n של <u>מה שלפני</u> ה- {n,}

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

בכל הכללים הנ"ל <u>מה שלפני</u> 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 ^ab$ | cd יתאים למחרוזת ab או למחרוזת שמכילה את הרצף cd באיזשהו מקום.

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

החוקים של ביטויים רגולאריים בסיסיים (Basic Regular Expressions)

הפקודה grep משתמשת בביטויים רגולאריים בסיסיים כאשר ההבדל בינה לבין egrep הוא שלסימנים + } {) (| ? צריך להקדים \ כדי שהם יעבדו עם הפקודה grep.

לדוגמה, הפקודה:

```
egrep "[0-9]{3}"
```

שקולה לפקודה:

```
grep  "[0-9]\{3\}"
```

כאשר כותבים grep -E זה שקול ל- egrep ולכן שתי הפקודות הנ"ל שקולות לפקודה:

```
grep -E "[0-9]{3}"
```

להלן דוגמאות של שימוש בביטויים רגולאריים מורחבים בפקודה egrep.

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec6>cat F1

Jim Alchin 21 Seattle aef
Bill Gates 21 Seattle aef
Steve Jobs 21 Nevada aaa
Scott Neally 85 Los Angeles
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep "21 S" F1
```

מתקבל הפלט:

```
Jim Alchin 21 Seattle aef
Bill Gates 21 Seattle aef
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep "21 s" F1
```

מתקבל פלט ריק (שלא מכיל כלום, אפילו לא שורה ריקה)
מאחר ובאף שורה בקובץ F1 לא מופיעה המחרוזת:

```
21 s
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep [a-d][a-d] F1
```

מתקבל הפלט:

```
Steve Jobs 21 Nevada aaa
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep [^a-d]ea F1
```

מתקבל הפלט:

```
Jim Alchin 21 Seattle aef
Bill Gates 21 Seattle aef
Scott Neally 85 Los Angeles
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep [^Ait]l F1
```

מתקבל הפלט:

```
Bill Gates 21 Seattle aef
Scott Neally 85 Los Angeles
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep [^Aitl]l F1
```

מתקבל הפלט:

```
Scott Neally 85 Los Angeles
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep ^[abc] F1
```

מתקבל פלט ריק (שאינו מכיל אף שורה)

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep ^[^S] F1
```

מתקבל הפלט:

```
Jim Alchin 21 Seattle aef
Bill Gates 21 Seattle aef
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep ^[^SB] F1
```

מתקבל הפלט:

```
Jim Alchin 21 Seattle aef
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep [as]$ F1
```

מתקבל הפלט:

```
Steve Jobs 21 Nevada aaa
Scott Neally 85 Los Angeles
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep [^as]$ F1
```

מתקבל הפלט:

```
Jim Alchin 21 Seattle aef
Bill Gates 21 Seattle aef
```

ננית שתוכן הקובץ F2 הוא:

```
basicsys@mars~/lec6>cat F2
J
B
S
S
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep ^[Bill]$ F1
```

מתקבל פלט ריק.

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep ^[Bill]$ F2
B
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>(echo JJ; tail -3 F2) >| F3
```

תוכן הקובץ F3 הוא:

```
basicsys@mars~/lec6>cat F3
JJ
B
S
S
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep ^[JB]$ F3
```

מתקבל הפלט:

```
B
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep ^.$ F3
```

מתקבל הפלט:

```
B
S
S
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep ^..$ F3
```

מתקבל הפלט:

```
JJ
```

נניח שתוכן הקובץ F4 הוא:

```
basicsys@mars~/lec6>cat F4
abb8a
2abb8a
abba
abbba
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep "^ab.[^a]" F4
```

מתקבל הפלט:

```
abb8a
abbba
```

ננית שתוכן הקובץ F5 הוא:

```
basicsys@mars~/lec6>cat F5
a8b
a88b
a8
8
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep "[0-9]$" F5
```

מתקבל הפלט:

```
a8
8
```

ננית שתוכן הקובץ F6 הוא:

```
basicsys@mars~/lec6>cat F6
88
8
aa
a
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep "^[^0-9]$" F6
```

מתקבל הפלט:

```
a
```

ננית שתוכן הקובץ F7 הוא:

```
basicsys@mars~/lec6>cat F7
caca
caac
caa
ca
c
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep "^ca*$" F7
```

מתקבל הפלט:

```
caa
ca
c
```

נניח שתוכן הקובץ F8 הוא:

```
basicsys@mars~/lec6>cat F8
caa

caca
cacaa
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep "^(ca)*$" F8
```

מתקבל הפלט:

```
caca
```

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec6>cat F1

8
8w
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep "(ca)*" F1
```

מתקבל הפלט:

```
8
8w
```

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec6>cat F1

8
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep "c*d*" F1
```

מתקבל הפלט:

```
8
```

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec6>cat F1
8
a
8a
a8
ababab
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep "[a-c]*[^a-c]" F1
```

מתקבל הפלט:

```
8
8a
a8
```

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec6>cat F1
005
00157
a0a00b7
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep "0.*0.*7" F1
```

מתקבל הפלט:

```
00157
a0a00b7
```

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec6>cat F1
000000001
0000000010
123456789
0123456789
a123456789
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep "^[0-9]{9}$" F1
```

מתקבל הפלט:

```
000000001
123456789
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep "^[0-9]{9}" F1
```

מתקבל הפלט:

```
000000001
0000000010
123456789
0123456789
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>grep "^[0-9]{9}$" F1
```

מתקבל פלט ריק.

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>grep "^[0-9]\{9\}$" F1
```

מתקבל הפלט:

```
000000001
123456789
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>grep -E "^[0-9]{9}$" F1
```

מתקבל הפלט:

```
000000001
123456789
```

ננית שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec6>cat F1
acdat
acatb
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep "cat|dog|cow" F1
```

מתקבל הפלט:

```
acatb
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep "c.*a.*t|dog|cow" F1
```

מתקבל הפלט:

```
acdat
acatb
```

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec6>cat F1
adogb
acat
acow
catb
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep "^cat|dog|cow$" F1
```

מתקבל הפלט:

```
adogb
acow
catb
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep "^(cat|dog|cow)$" F1
```

מתקבל פלט ריק.

אופציות של grep/egrep

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

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec6>cat F1
     abc shalom  de
     abc   de shalom1
abc de     shalom 123
shalom
aaashalom
aaaa bbbb shalom
   shalom
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep "\<shalom\>" F1
```

מתקבל הפלט:

```
     abc shalom  de
abc de     shalom 123
shalom
aaaa bbbb shalom
   shalom
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep -w "shalom" F1
```

מתקבל הפלט:

```
     abc shalom  de
abc de     shalom 123
shalom
aaaa bbbb shalom
   shalom
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep -w "shalom.*d" F1
```

מתקבל פלט ריק.

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep "\<shalom\>.*d" F1
```

מתקבל הפלט:

```
     abc shalom  de
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep "shalom" F1
```

מתקבל הפלט:

```
     abc shalom  de
     abc   de shalom1
abc de     shalom 123
shalom
aaashalom
aaaa bbbb shalom
   shalom
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>echo abcd@ | egrep -w abcd
```

מתקבל הפלט:

```
abcd@
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>echo abcd1 | egrep -w abcd
```

מתקבל פלט ריק.

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>echo abcd@ | egrep "\<abcd\>"
```

מתקבל הפלט:

```
abcd@
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>echo abcd1 | egrep "\<abcd\>"
```

מתקבל פלט ריק.

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec6>cat F1

     abc   de shalom
     abc   de shalom1
abc de     shalom 123
```

הפקודה הבאה מציגה לפלט את כל השורות בקובץ F1 שהשורה השלישית שלהם מכילה את המילה shalom.

לאחר הפעלת הפקודה:

```
egrep "^[ ]*[^ ]+[ ]+[^ ]+[ ]+shalom($|[ ])" F1
```

מתקבל הפלט:

```
     abc   de shalom
abc de     shalom 123
```

שימוש ב- backtracking בביטוי רגולארי

\1 מתאים לחלק של המחרוזת שהתאים לסוגריים הראשונים בביטוי

\2 מתאים לחלק של המחרוזת שהתאים לסוגריים השניים בביטוי

\i מתאים 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...

לדוגמה:

```
1      2    2            3     4   5   5 4 3 1

( ab + ( cd )+ ef [2-4]gh( ijk ( lm( n ) ) ) )
```

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec6>cat F1
sdf7assssy8sa
66
afasfarrr   998
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep "(.)\1\1" F1
```

מתקבל הפלט:

```
sdf7assssy8sa
afasfarrr   998
```

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec6>cat F1
abccba
aasddsa123321 888
dasfasd
```

```
123 321
 555   123321  999999
12321
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep "(.)(.)(.)\3\2\1" F1
```

מתקבל הפלט:

```
abccba
aasddsa123321 888
 555   123321  999999
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep -o "(.)(.)(.)\3\2\1" F1
```

מתקבל הפלט:

```
abccba
asddsa
123321
123321
999999
```

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec6>cat F1
abcdeabcde
123abc123abc
abab
aa
aaaaaa
abcd
abcdd
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep "(.+)\1" F1
```

מתקבל הפלט:

```
abcdeabcde
123abc123abc
abab
```

```
aa
aaaaaa
abcdd
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep "^(.+)\1$" F1
```

מתקבל הפלט:

```
abcdeabcde
123abc123abc
abab
aa
aaaaaa
```

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec6>cat F1
abcdd   adfdafda
b98sdfsad09sa9b
dsaudsoas
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep "^([a-z]).*\1$" F1
```

מתקבל הפלט:

```
abcdd   adfdafda
b98sdfsad09sa9b
```

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec6>cat F1
abcdabcda
123412341
1234112341
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep "^((.).+)\1\2$" F1
```

מתקבל הפלט:

```
abcdabcda
123412341
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>echo -n aabbaba | egrep -o a
```

מתקבל הפלט:

```
a
a
a
a
```

לאחר הפעלת הפקודה:

```
basicsy~/lec6>echo -n aabbaba |egrep -o a | wc -l
```

מתקבל הפלט:

```
4
```

לאחר הפעלת הפקודה:

```
basicsys~/lec6>echo -n aabbaba |tr a "\n" | wc -l
```

מתקבל הפלט:

```
4
```

הפקודה הבאה מציגה את התווים שבמחרוזת כך שכל תו מופיע בשורה נפרדת.

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec8>echo abcde | egrep -o .
```

מתקבל הפלט:

```
a
b
c
d
e
```

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec6>cat F1

abcdef
```

```
12334ab
abcd
yyy
```

ננית שתוכן הקובץ F2 הוא:

```
basicsys@mars~/lec6>cat F2
shalom hi hello
yossi rami
1234
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep -c ab F1 F2
```

מתקבל הפלט:

```
F1:3
F2:0
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep -c -h ab F1 F2
```

מתקבל הפלט:

```
3
0
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep ab F1 F2
```

מתקבל הפלט:

```
F1:abcdef
F1:12334ab
F1:abcd
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep AB F1 F2
```

מתקבל פלט ריק.

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep -i AB F1 F2
```

מתקבל הפלט:

```
F1:abcdef
F1:12334ab
F1:abcd
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep -i -l AB F1 F2
```

מתקבל הפלט:

```
F1
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep -i  "AB|1" F1 F2
```

מתקבל הפלט:

```
F1:abcdef
F1:12334ab
F1:abcd
F2:1234
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep -i -l "AB|1" F1 F2
```

מתקבל הפלט:

```
F1
F2
```

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec6>cat F1
adogb
acat
acow
catb
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>egrep -v "cat" F1
```

מתקבל הפלט:

```
adogb
acow
```

העברת פרמטרים לתכנית סקריפט

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

$# - מספר הפרמטרים לתכנית

$* - 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.

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec6>cat P1

echo "the first parameter is $1"
echo "the second parameter is $2"
echo "the number of parameters $#"
echo "all the parameters are: $*"
echo "all the parameters are: $@"
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>P1 10 ab cd 20 gh
```

מתקבל הפלט:

```
the first parameter is 10
the second parameter is ab
the number of parameters 5
all the parameters are: 10 ab cd 20 gh
all the parameters are: 10 ab cd 20 gh
```

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec6>cat P1
for x in $*
 do
  echo "$x"
 done
for x in $@
 do
  echo "$x"
 done
for x in "$*"
 do
  echo "$x"
 done
for x in "$@"
 do
  echo "$x"
 done
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>P1 "ab   cd"  "ef   gh"
```

מתקבל הפלט:

```
ab
cd
ef
gh
ab
cd
ef
gh
ab   cd ef   gh
ab   cd
ef   gh
```

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec6>cat P1

while read x
 do
 if [ $(echo "$x" | egrep -c "$1") -gt 0 ]
 then
  echo $(echo "$x" | egrep -o "$1")
 fi
 done < $2
```

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec6>cat F1
abccba
aasddsa123321 888
dasfasd
123 321
 555   123321  999999
12321
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>P1 "(.)(.)(.)\3\2\1" F1
```

מתקבל הפלט:

```
abccba
asddsa 123321
123321 999999
```

<u>הפקודה shift</u>

מבצעת הזזה שמאלה של הפרמטרים לתכנית (כאשר הפרמטר הראשון נמחק). לדוגמה, אם הפרמטרים לתכנית הם

```
10 20 30 40
```

אחרי ביצוע הפקודה shift הפרמטרים לתכנית הופכים להיות:

```
20 30 40
```

לדוגמה, נניח שתוכן התכנית P2 הוא:

```
basicsys@mars~/lec6>cat P2
echo $*
echo $#
shift
echo $*
echo $#
shift
echo $*
echo $#
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>P2 10 20 30 40
```

מתקבל הפלט:

```
10 20 30 40
4
20 30 40
3
30 40
2
```

נניח שתוכן התכנית P3 הוא:

```
basicsys@mars~/lec6>cat P3
s=0
while [ $# -ge 1 ]
 do
  s=$[$s+$1]
  shift
 done
echo $s
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>P3 10 20 30
```

מתקבל הפלט:

```
60
```

מבנה משפט case

```
case   מחרוזת  in

1 ביטוי)

     1 סדרת פקודות ;;

2 ביטוי)

     2 סדרת פקודות ;;

   ...

esac
```

בודקים התאמה בין המחרוזת לביטוי הראשון, אם יש התאמה מתבצעת סדרת הפקודות הראשונה, אם אין התאמה בודקים התאמה לביטוי השני וכן הלאה.. הביטויים הם בסגנון glob בתוספת הסימן | שמציין or לוגי.

לדוגמה, נניח שתוכן התכנית P16 הוא:

```
basicsys@mars~/lec6>cat P16

read x
case $x in
 a)
  echo "you typed a";;
 b|c)
  echo "you typed b or c";;
 g*)
  echo "you typed a string that starts with g";;
 f??)
  echo "you typed a string that starts with f"
  echo "followed by exactly two characters";;
 *)
  echo "you typed nothing of the first options";;
esac
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>P16
```

המשתמש מתבקש להקליד קלט, נניח שהמשתמש מקליד:

```
yyyy
```

אז מתקבל הפלט:

```
you typed nothing of the first options
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>P16
```

המשתמש מתבקש להקליד קלט, נניח שהמשתמש מקליד:

```
a
```

אז מתקבל הפלט:

```
you typed a
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>P16
```

המשתמש מתבקש להקליד קלט, נניח שהמשתמש מקליד:

```
b
```

אז מתקבל הפלט:

```
you typed b or c
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>P16
```

המשתמש מתבקש להקליד קלט, נניח שהמשתמש מקליד:

```
fabc
```

אז מתקבל הפלט:

```
you typed nothing of the first options
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>P16
```

המשתמש מתבקש להקליד קלט, נניח שהמשתמש מקליד:

```
f1u
```

אז מתקבל הפלט:

```
you typed a string that starts with f
followed by exactly two characters
```

הרצאה מספר 7

הפקודה find

מבנה הפקודה:

find <תיקיה/תיקיות> <תנאים> <פקודות לביצוע>

הפקודה עוברת על כל הקבצים/תיקיות בתיקיות שהועברו לה כפרמטרים.

עבור כל קובץ/תיקיה בודקת את כל התנאים (לפי הסדר משמאל לימין) אם כל התנאים מתקיימים אז מבוצעות הפקודות לביצוע. התנאים והפקודות לביצוע הם במבנה 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.

התנאים של הפקודה find

בכל התנאים שבודקים ערך מספרי מתקיים הכלל הבא:

n מציין שווה בדיוק ל- n

n+ מציין גדול מ- n

n- מציין קטן מ- n

גודל הקובץ/תיקיה -size

לדוגמה, -size 4c הקובץ/תיקיה מכיל 4 תווים בדיוק.

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

סוג הקובץ/תיקיה -type

האפשרויות: d - תיקיה, f - קובץ

מספר -mmin מתייחס לזמן (בדקות) של העדכון האחרון של הקובץ. לדוגמה -100 -mmin אומר שהקובץ עודכן לאחרונה לפני פחות מ- 100 דקות.

הפקודות לביצוע של הפקודה find

נדגיש שוב שהפקודות לביצוע הן רק עבור הפקודה find והן מבוצעות עבור כל קובץ/תיקיה שה- find מגיע אליו בתנאי שהקובץ/תיקיה מקיים את כל התנאים שמופיעים לפני הפקודה לביצוע.

-print הדפס את שם הקובץ/תיקיה שאתה נמצא בו כרגע (כולל תו \n). שם הקובץ שיודפס יכלול את המסלול אל הקובץ/תיקיה החל מהתיקייה שהועברה כפרמטר לפקודה find. הפקודה -print היא ברירת המחדל של הפקודה find.

-print0 הדפס את שם הקובץ/תיקיה שאתה נמצא בו כרגע ללא תו \n.

-exec <פקודה> \; בצע את הפקודה (כפקודה של bash). אם בפקודה מופיע הסימן {} החלף אותו בשם הקובץ שאתה נמצא בו כרגע, לפני ביצוע הפקודה.

-exec <פקודה> {} + החלף את הסוגריים המסולסלים בשמות הקבצים שמקימים את התנאים שלפני ה- exec ורק אחרי ההחלפה של כל השמות, בצע את הפקודה.

חשוב לשים לב שהפקודות של ה- exec- מתבצעות בתיקיה שבה הופעלה פקודת ה- find ולא בתיקיות ששמן הועבר כפרמטר לפקודת ה- find

נניח שמבנה התיקיה t/d1 הנו כפי שמתואר על ידי
הפקודה tree הבאה:

```
basicsys@mars~/lec7>tree t/d1
t/d1
|-- d3
|   |-- a1
|   |-- f2
|   `-- f3
`-- f4

2 directories, 3 files
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec7>find t/d1 -type f
```

מתקבל הפלט:

```
t/d1/f4
t/d1/d3/f2
t/d1/d3/f3
```

בפלט הנ"ל מופיעים כל שמות הקבצים שנמצאים
בתיקיה t/d1. מאחר ולפקודה find אין פקודות לביצוע,
הפקודה שמתבצעת היא ברירת המחדל שהיא הפקודה print-
שמדפיסה (עבור כל קובץ שמקיים את התנאים
של ה- find) את המסלול אל הקובץ החל מ- t/d1.

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec7>find t/d1 -type d
```

מתקבל הפלט:

```
t/d1
t/d1/d3
t/d1/d3/a1
```

נניח שמבנה התיקיה t הנו כפי שמתואר על ידי הפקודה tree הבאה:

```
basicsys@mars~/lec7>tree t
t
|-- d1
|   |-- d3
|   |   |-- a1
|   |   |-- f2
|   |   `-- f3
|   `-- f4
|-- d2
|   `-- d3
|       `-- f5
`-- f1

5 directories, 5 files
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec7>find t -name f5 -print
```

מתקבל הפלט:

```
t/d2/d3/f5
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec7>find t -name f* -print
```

מתקבל הפלט:

```
t/d1/f4
t/d1/d3/f2
t/d1/d3/f3
t/f1
t/d2/d3/f5
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec7>find t -name f* -print -print
```

מתקבל הפלט:

```
t/d1/f4
t/d1/f4
```

```
t/d1/d3/f2
t/d1/d3/f2
t/d1/d3/f3
t/d1/d3/f3
t/f1
t/f1
t/d2/d3/f5
t/d2/d3/f5
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec7>find t -name f*
```

מתקבל הפלט:

```
t/d1/f4
t/d1/d3/f2
t/d1/d3/f3
t/f1
t/d2/d3/f5
```

נניח שלאחר הפעלת הפקודה:

```
basicsys@mars~/lec7>ls -l t
```

מתקבל הפלט:

```
total 12
drwx------ 3 basicsys basicsys 4096 Dec  9 14:10 d1
drwx------ 3 basicsys basicsys 4096 Dec  9 14:10 d2
-rw------- 1 basicsys basicsys   37 Dec  9 14:10 f1
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec7>find t -size 37c
```

מתקבל הפלט:

```
t/f1
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec7>find t -size +37c
```

מתקבל הפלט:

```
t
t/d1
```

```
t/d1/d3
t/d1/d3/f2
t/d1/d3/f3
t/d1/d3/a1
t/d2
t/d2/d3
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec7>find t -size -37c
```

מתקבל הפלט:

```
t/d1/f4
t/d2/d3/f5
```

נניח שלאחר הפעלת הפקודה:

```
basicsys@mars~/lec7>ls -l t/d1/f4 t/d2/d3/f5
```

מתקבל הפלט:

```
-rw------- 1 basicsys basicsys 32 Dec  9 14:10 t/d1/f4
-rw------- 1 basicsys basicsys 36 Dec  9 14:10 t/d2/d3/f5
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec7>find -name a* -o -size 37c
```

מתקבל הפלט:

```
./t/d1/d3/a1
./t/f1
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec7>find t -size -38c -a -name 'f*'
```

מתקבל הפלט:

```
t/d1/f4
t/f1
t/d2/d3/f5
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec7>find t -size -38c -name f*
```

מתקבל הפלט:

```
t/d1/f4
t/f1
t/d2/d3/f5
```

לאחר הפעלת הפקודה:

```
find t \( -name a* -o -size 37c \) -mmin -100
```

מתקבל פלט ריק.

לאחר הפעלת הפקודה:

```
find t \( -name a* -o -size 37c \) -mmin +100
```

מתקבל הפלט:

```
t/d1/d3/a1
t/f1
```

נניח שמבנה התיקיה t/d1 הנו כפי שמתואר על ידי הפקודה tree הבאה:

```
asicsys@mars~/lec7>tree t/d1
t/d1
|-- d3
|   |-- a1
|   |-- f2
|   `-- f3
`-- f4

2 directories, 3 files
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec7>find t/d1 -exec echo hi \;
```

מתקבל הפלט:

```
hi
hi
hi
hi
hi
hi
```

לאחר הפעלת הפקודה:

```
find t/d1 -name f* -exec echo hi \;
```

מתקבל הפלט:

```
hi
hi
hi
```

לאחר הפעלת הפקודה:

```
find t/d1 -name f* -exec echo hi \; -print
```

מתקבל הפלט:

```
hi
t/d1/f4
hi
t/d1/d3/f2
hi
t/d1/d3/f3
```

לאחר הפעלת הפקודה:

```
find t/d1 -name f* -exec echo -n "hi " \; -print
```

מתקבל הפלט:

```
hi t/d1/f4
hi t/d1/d3/f2
hi t/d1/d3/f3
```

לאחר הפעלת הפקודה:

```
find t/d1 -name f* -exec echo -n "hi " \; \
-print0 -exec echo " bye" \;
```

מתקבל הפלט:

```
hi t/d1/f4 bye
hi t/d1/d3/f2 bye
hi t/d1/d3/f3 bye
```

לאחר הפעלת הפקודה:

```
find t/d1 -name f* -print -exec head -2 {} \;
```

מתקבל הפלט:

```
t/d1/f4
I am file f4 indirectory t/d1
a
t/d1/d3/f2
I am file f2 at directory t/d1/d3
5ab
t/d1/d3/f3
I am file f3 at directory t/d1/d3
dd
```

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec7>cat P1
head -2 $1 | tail -1
```

לאחר הפעלת הפקודה:

```
find t/d1 -name f* -exec P1 {} \;
```

מתקבל הפלט:

```
a
5ab
dd
```

נניח שתוכן התכנית finder הוא:

```
basicsys@mars~/lec7>cat finder
find $1 -type f -exec egrep -l $2 {} \;
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec7>finder t ab
```

מתקבל הפלט:

```
t/d1/d3/f2
t/d1/d3/f3
```

נניח שתוכן הקובץ t/d1/d3/f2 הוא:

```
basicsys@mars~/lec7>cat t/d1/d3/f2
I am file f2 at directory t/d1/d3
5ab
```

3

ננית שתוכן הקובץ t/d1/d3/f3 הוא:

```
basicsys@mars~/lec7>cat t/d1/d3/f3
I am file f3 at directory t/d1/d3
dd
1ab
```

ננית שתוכן התכנית get_file_name הוא:

```
basicsys@mars~/lec7>cat get_file_name
echo $1 | tr "/" "\n" | tail -1
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec7>get_file_name ab/cd/ef
```

מתקבל הפלט:

```
ef
```

ננית שמבנה התיקיה t/d1 הנו כפי שמתואר על ידי הפקודה tree הבאה:

```
basicsys@mars~/lec7>tree t1
t1
|-- d1
|   |-- d3
|   |   |-- a1
|   |   |-- f2
|   |   `-- f3
|   |-- f3
|   `-- f4
|-- d2
|   `-- d3
|       `-- f5
|-- f1
`-- f2
```

ננית שתוכן התכנית get_file_name הוא:

```
basicsys@mars~/lec7>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
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec7>remove_duplicates t1
```

מתקבל הפלט הבא שבו המשתמש מתבקש להקליד y או n על כל אחת מהשאלות, וניח שהמשתמש עונה n לכל השאלות:

```
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
```

לאחר הפעלת הפקודה:

```
find t/d1 -type f -exec echo hi {} +
```

מתקבל הפלט:

```
hi t/d1/f4 t/d1/d3/f2 t/d1/d3/f3
```

לאחר הפעלת הפקודה:

```
find t/d1 -type f -exec echo hi {} + -print
```

מתקבל הפלט:

```
t/d1/f4
t/d1/d3/f2
t/d1/d3/f3
hi t/d1/f4 t/d1/d3/f2 t/d1/d3/f3
```

לאחר הפעלת הפקודה:

```
find t/d1 -type f -exec echo hi {} + -print \
-exec echo bye {} +
```

מתקבל הפלט:

```
t/d1/f4
t/d1/d3/f2
t/d1/d3/f3
hi t/d1/f4 t/d1/d3/f2 t/d1/d3/f3
bye t/d1/f4 t/d1/d3/f2 t/d1/d3/f3
```

הפקודה xargs

הפקודה <מחרוזת> xargs

משרשרת את הקלט הסטנדרטי למחרוזת ואת התוצאה (לאחר השרשור) מבצעת כפקודת bash.

אופציות:

<קובץ> a- שרשר למחרוזת את תוכן הקובץ במקום לשרשר את הקלט הסטנדרטי

n- שרשר n מילים מהקלט למחרוזת ובצע את המחרוזת (לאחר השרשור) כפקודת bash, לאחר מכן שרשר n מילים נוספות מהקלט למחרוזת ובצע את המחרוזת (לאחר השרשור) כפקודת bash, וכן הלאה עד שמסתיים הקלט.

התכנית הבאה יוצרת קובץ בשם FF שמכיל שמות קבצים F1-F10

```
basicsys@mars~/lec7>cat C1

echo -n >| FF
for i in $(seq 10)
 do
  echo F$i >> FF
 done
```

לאחר הפעלת התכנית C1 נוצר קובץ בשם FF שתכנו הוא:

```
basicsys@mars~/lec7>cat FF

F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec7>cat F[0-9]*
```

מתקבל הפלט:

```
I am file F1
I am file F10
I am file F2
I am file F3
I am file F4
I am file F5
I am file F6
I am file F7
I am file F8
I am file F9
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec7>ls F[0-9]*
```

מתקבל הפלט:

```
F1  F10  F2  F3  F4  F5  F6  F7  F8  F9
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec7>ls F[0-9]* | xargs cat
```

מתקבל הפלט:

```
I am file F1
I am file F10
I am file F2
I am file F3
I am file F4
I am file F5
I am file F6
I am file F7
I am file F8
I am file F9
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec7>xargs cat
```

המשתמש מתבקש להקליד קלט, נניח שהמשתמש מקליד את שתי השורות הבאות ולאחר מכן מקליד Ctrl+d

```
F1 F5
F7 F8 F2
```

אז מתקבל הפלט:

```
I am file F1
I am file F5
I am file F7
I am file F8
I am file F2
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec7>xargs echo
```

המשתמש מתבקש להקליד קלט, נניח שהמשתמש מקליד את שתי השורות הבאות ולאחר מכן מקליד Ctrl+d

```
abc
def ghe
```

אז מתקבל הפלט:

```
abc def ghe
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec7>xargs echo -n
```

המשתמש מתבקש להקליד קלט, נניח שהמשתמש מקליד את שתי השורות הבאות ולאחר מכן מקליד Ctrl+d

```
abc
def
```

אז מתקבל הפלט:

```
abc defbasicsys@mars~/lec7>
```

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec7>cat F1
abc def
gh
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec7>xargs -a F1 echo
```

מתקבל הפלט:

```
abc def gh
```

לאחר הפעלת הפקודה:

```
echo 1 2 3 4 5 6 7 8 9 10  | xargs -n 3 echo
```

מתקבל הפלט:

```
1 2 3
4 5 6
7 8 9
10
```

התכנית הבאה, מראה את החיסכון בזמן שנובע משימוש בפקודה xargs בשילוב הפקודה find במקום להשתמש באופציה exec- של find.

תוכן התכנית G2 הוא:

```
basicsys@mars~/lec7>cat G2
echo "using xargs"
date
find ~/win12 -type f | xargs egrep -l abc >|/dev/null
date
echo "using -exec "
date
find ~/win12 -type f -exec egrep -l abc {} \; >|/dev/null
date
```

לאחר הפעלת התכנית G2:

```
basicsys@mars~/lec7>G2
```

מתקבל הפלט:

```
using xargs
Fri Jan  6 13:20:03 IST 2012
Fri Jan  6 13:20:03 IST 2012
using -exec
Fri Jan  6 13:20:03 IST 2012
Fri Jan  6 13:20:26 IST 2012
```

מהפלט הנ"ל רואים שזמן הביצוע של הפקודה:

```
find ~/win12 -type f | xargs egrep -l abc >|/dev/null
```

היה פחות משניה.

בעוד שזמן הביצוע של הפקודה:

```
find ~/win12 -type f -exec egrep -l abc {} \; >|/dev/null
```

היה 23 שניות.

הרצאה מספר 8

awk

awk היא שפה לניתוח טקסט ולהפקת דוחות. השפה הזו ניתנת להפעלה בכל shell של unix כולל bash.

נתחיל עם מספר דוגמאות פשוטות להפעלת השפה.

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec8>cat F1

yyy moshe1
moshe2
no mosh
abc
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec8>awk '/moshe/{print;print}' F1
```

מתקבל הפלט:

```
yyy moshe1
yyy moshe1
moshe2
moshe2
```

בפקודה הנ"ל נבדק התנאי /moshe/ על כל אחת מהשורות של הקובץ F1. משמעות התנאי היא שהמחרוזת moshe מופיעה באיזה שהוא מקום בשורה. עבור כל שורה שמקיימת את התנאי מופעלת סדרת הפקודות
{print;print}

שמבצעת שתי פקודות print. הפקודה print היא פקודה בשפה של awk שמשמעותה: הדפס את כל השורה הנוכחית (כולל תו n\). לכן בפלט הנ"ל כל שורה שמקימת את התנאי /moshe/ מודפסת פעמיים.

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec8>awk '{print;print}' F1
```

מתקבל הפלט:

```
yyy moshe1
yyy moshe1
moshe2
moshe2
no mosh
no mosh
abc
abc
```

בפקודה הנ"ל לא הופיע תנאי לפני סדרת הפקודות {print;print}
ולכן סדרת הפקודות הופעלה על כל השורות של הקובץ F1. במילים אחרות, אם לפני סדרת פקודות בתוך סוגריים מסולסלים לא מופיע תנאי אז ברירת המחדל היא כאילו מופיע לפניהן תנאי True ולכן הסדרה מופעלת על כל השורות.

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 לתכנית (בדרך כלל זו רשימת הקבצים שמועברים בקריאה לתכנית ה- awk), כאשר

ARGV[0] מכיל את המחרוזת awk

ARGV[1] מכיל את הפרמטר הראשון לתכנית

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

ARGC מכיל את מספר הפרמטרים לתכנית 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):

מספר1 > מספר2 מספר1 <= מספר2

מספר1 < מספר2 מספר1 >= מספר2

מספר1 == מספר2 מספר1 =! מספר2

מחרוזת1 == מחרוזת2 מחרוזת1 =! מחרוזת2

תנאי1 && תנאי2 תנאי1 || תנאי2

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

כמו למשל: (תנאי 4 || תנאי 3) && (תנאי 2 || תנאי 1)

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 ב- awk

הפקודה:

print מחרוזת1,מחרוזת2...

מדפיסה את המחרוזות כאשר בין המחרוזות מודפס הערך של משתנה המערכת OFS (ברירת מחדל: תו רווח אחד), בנוסף מודפס בסוף הפלט הערך של משתנה המערכת ORS (ברירת מחדל: \n).

כאשר הפרמטרים לפקודה print לא מופרדים על ידי פסיקים כמו במבנה הבא:

print מחרוזת1 מחרוזת2...

המחרוזות יודפסו ברצף (ללא תווי הפרדה בין המחרוזות) ובסוף יודפס ערך משתנה המערכת ORS.

נניח שתוכן הקובץ F2 הוא:

```
basicsys@mars~/lec8>cat F2

aa   bb     cc
dd   ee   fff gg
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec8>awk '{print $3,$1}' F2
```

מתקבל הפלט:

```
cc aa
fff dd
```

בפקודה הנ"ל הועברו לפקודה print של awk שני פרמטרים $3 ו- $1 כאשר פסיק מפריד בין הפרמטרים. הפקודה מדפיסה את הפרמטרים שהועברו לה בתוספת רווח אחד בין הפרמטרים.

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec8>awk '{print $3 $1}' F2
```

מתקבל הפלט:

```
ccaa
fffdd
```

בדוגמה הבאה מועבר לפקודה print פרמטר אחד שהוא המחרוזת שמורכבת מהשדה השלישי בשורה,לאחריה תו רווח ולאחריה השדה הראשון בשורה.

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec8>awk '{print $3" "$1}' F2
```

מתקבל הפלט:

```
cc aa
fff dd
```

ב- awk להבדיל מ- bash לא מוסיפים $ לפני המשתנה כדי לקבל את ערכו. ולכן כאשר נעביר לפקודה print את המשתנה NF כפרמטר היא תדפיס את ערכו, שהוא מספר השדות בשורה, כפי שמראה הדוגמה הבאה:

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec8>awk '{print $0,NF}' F2
```

מתקבל הפלט:

```
aa   bb     cc 3
dd   ee   fff gg 4
```

כאשר משתמשים ב- $i עבור משתנה i הוא מוחלף בשדה ה- i בשורה. ולכן כאשר נשתמש ב- $NF כפרמטר לפקודה

print יודפס השדה האחרון בשורה, כפי שמראה הדוגמה הבאה:

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec8>awk '{print $0,$NF}' F2
```

מתקבל הפלט:

```
aa   bb     cc cc
dd   ee   fff gg gg
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec8>awk '{print $NF}' F2
```

מתקבל הפלט:

```
cc
gg
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec8>awk '{print NR,$0}' F1 F2
```

מתקבל הפלט:

```
1 yyy moshe1
2 moshe2
3 no mosh
4 abc
5 aa   bb     cc
6 dd   ee   fff gg
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec8>awk '{print FNR,$0}' F1 F2
```

מתקבל הפלט:

```
1 yyy moshe1
2 moshe2
3 no mosh
4 abc
1 aa   bb     cc
2 dd   ee   fff gg
```

כפי שמראה הדוגמה הבאה ניתן להשתמש בתנאי השוואה
בין מספרים כמו למשל: מספר2 < מספר1 וכו'

נניח שתוכן הקובץ F4 הוא:

```
basicsys@mars~/lec8>cat F4

aa 8 2
bb  7   6
cc  2   4
```

לאחר הפעלת הפקודה:

```
sys@mars~/lec8>awk '$3>2 {print $0,$2*$3}' F4
```

מתקבל הפלט:

```
bb  7   6 42
cc  2   4 8
```

בפקודה הנ"ל מתבצעת פעולת כפל בין המספר שנמצא בשדה
השני בשורה לבין המספר שנמצא בשדה השלישי בשורה.

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec8>awk '$3>2 {print $0,x*1}' F4
```

מתקבל הפלט:

```
bb  7   6 0
cc  2   4 0
```

בדוגמה הנ"ל המשתנה x לא אותחל, ונעשה שימוש בערכו
בביטוי חשבוני ולכן הערך שמחושב עבור x במקרה זה
הוא 0.

גם בדוגמה הבאה הערך המחושב עבור x הוא 0.

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec8>awk '$3>2 {print $0,x+1}' F4
```

מתקבל הפלט:

```
bb  7   6 1
cc  2   4 1
```

בדוגמה הבאה, המשתנה x מאותחל לערך 5 וזה הערך שמחושב עבור x.

לאחר הפעלת הפקודה:

```
csys@mars~/lec8>awk '$3>2 {x=5;print $0,x+1}' F4
```

מתקבל הפלט:

```
bb  7   6 6
cc  2   4 6
```

שימוש ב- awk לביצוע חישובים במספרים לא שלמים

להבדיל מ- bash ניתן להשתמש ב- awk לביצוע חישובים אריתמטיים במספרים שאינם שלמים. לדוגמה, לאחר הפעלת הפקודה:

```
basicsys@mars~/lec8>awk '$3>2 {print $3/3}' F4
```

מתקבל הפלט:

```
2
1.33333
```

מאחר ו- bash מבצע פעולות חשבון במספרים שלמים בלבד, לאחר הפעלת הפקודה:

```
basicsys@mars~/lec8>echo $[4/3]
```

מתקבל הפלט:

```
1
```

לאחר הפעלת הפקודות:

```
basicsys@mars~/lec8>x=5.5

basicsys@mars~/lec8>y=3.2

basicsys@mars~/lec8>echo $[$x+$y]
```

מתקבל הפלט:

```
-bash: 5.5+3.2: syntax error: invalid arithmet
```

מצד שני כדי לבצע פעולות חשבון במספרים לא שלמים בעזרת awk אפשר להשתמש במבנה שמתואר בדוגמה הבאה.

לאחר הפעלת הפקודה:

```
mars~/lec8>echo $x $y | awk '{print $1+$2}'
```

מתקבל הפלט:

```
8.7
```

בפקודה הנ"ל, מאחר ולא מועבר לפקודת ה- awk שם קובץ כפרמטר, הקלט לפקודת ה- awk מגיע מה- pipeline במקום מהקובץ.

שימוש ביותר מסדרת פקודות אחת ב- awk

ניתן להשתמש ב- awk ביותר מסדרת פקודות אחת, כאשר לפני כל סדרה מופיע תנאי. כל סדרת פקודות תתבצע רק אם התנאי שמופיע מיד לפני הפקודה מתקיים.

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec8>cat F1

ab  cd   ef  gh
ab cd
d e f g h
```

לאחר הפעלת הפקודה:

```
awk 'NF>=3 {print "3",$0} NF==5 {print "5",$0}' F1
```

מתקבל הפלט:

```
3 ab  cd   ef  gh
3 d e f g h
5 d e f g h
```

בדוגמה הנ"ל עבור כל שורה מתבצע:

אם השדה השלישי בשורה גדול מ- 3 מודפס 3 ולאחריו השורה.
אם בשורה ישנם בדיוק 5 שדות אז מודפס 5 ולאחריו מודפסת השורה.

השורה האחרונה בקובץ מקיימת את שני התנאים ולכן היא מודפסת פעמיים, פעם אחת עם 3 בהתחלה ופעם שניה עם 5 בהתחלה.

לאחר הפעלת הפקודה:

```
mars~/lec8>echo "ab   cd" ef | awk '{print $2}'
```

מתקבל הפלט:

```
cd
```

הסיבה לפלט הנ"ל היא ששורת הקלט ל- awk נראית כך:

```
ab    cd ef
```

ולכן השדה השני בשורה זו הוא cd.

לאחר הפעלת הפקודה:

```
mars~/lec8>echo ab   cd ef | awk '{print $2}'
```

מתקבל הפלט:

```
cd
```

בפקודה הנ"ל שורת הקלט ל- awk נראית כך:

```
ab cd ef
```

הפקודה printf ב- awk

מבנה הפקודה printf של awk הוא:

printf מחרוזת שמכילה ביטויי המרה,פרמטר1,פרמטר2.. .

הפקודה מדפיסה את המחרוזת לאחר החלפת ביטויי ההמרה בפרמטרים המתאימים להם לפי הסדר משמאל לימין.

לדוגמה בפקודה:

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

ביטוי ההמרה %5d יוחלף על יד הדפסת 123 בתוספת שני רווחים מצד שמאל,

וביטוי ההמרה %-6d יוחלף על ידי הדפסת 456 בתוספת 3 רווחים מצד ימין.

נניח שתוכן הקובץ F3 הוא:

```
basicsys@mars~/lec8>cat F3

1999 12  ab  Fiat
200000  130   cc   Subaru
```

לאחר הפעלת הפקודה:

```
awk '{printf \
"year %8d no %4d car %-8s good\n",$1,$2,$4}' F3
```

מתקבל הפלט:

```
year     1999 no   12 car Fiat     good
year   200000 no  130 car Subaru   good
```

הדוגמה הבאה מראה כיצד ניתן לישר לימין או לשמאל כאשר המספר שלפיו מישרים לימין או לשמאל אינו קבוע אלא שווה לערך של משתנה x של bash.

לאחר הפעלת שתי הפקודות:

```
x=-10

awk '{printf "year %'$x'd no\n",$1}' F3
```

מתקבל הפלט:

```
year 1999       no
year 200000     no
```

בדוגמה הנ"ל הערך של $x אינו בתוך גרשיים כלל ולכן הסורק מחליף אותו בערך של המשתנה x של bash. ולכן הפקודה שבעצם מועברת ל- awk לביצוע היא:

```
awk '{printf "year %-10d no\n",$1}' F3
```

הרצאה מספר 9

הפעלת awk

ישנן שלוש אפשרויות להפעלת awk

1. הפעלה משורת הפקודה באופן הבא:

```
awk  תנאי 1 {סדרת פקודות 1} תנאי 2 {סדרת פקודות 2}.. רשימת קבצים
```

2. הפעלה על ידי קריאה לתוכנית awk שנמצאת בקובץ נפרד (לדוגמה קובץ בשם B) באופן הבא:

```
awk  -f  B   רשימת קבצים
```

התוכנית awk שנמצאת בקובץ B הינה במבנה הבא:

```
תנאי 1  {
        סדרת פקודות לביצוע 1
          }
תנאי 2 {
        סדרת פקודות לביצוע 2
          }
...
```

3. הפעלת תוכנית סקריפט ב- awk שנמצאת בקובץ נפרד (לדוגמה קובץ בשם C) באופן הבא:

```
C   רשימת קבצים
```

תוכנית ה- awk שנמצאת בקובץ C הינה במבנה כפי שתוארה התוכנית שנמצאת בקובץ B כאשר השורה הראשונה חיבת להיות:

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

השורה הנ"ל מסמנת למערכת שהקובץ מכיל תוכנית בשפת awk.

הלוגיקה של תוכנית awk

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

```
תנאי 1  {

        סדרת פקודות לביצוע 1

          }

תנאי 2  {

        סדרת פקודות לביצוע 2

          }
...
```

הקלט לתוכנית מגיע מרשימת הקבצים שהועברו בזמן הפעלת awk (בכל אחת משלושת הצורות להפעלת awk שנה 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 הראשון (לפני שקוראים שורות קלט):

מבצעים את כל סדרות הפקודות שהתנאי המקדים להן הוא BEGIN. במילים אחרות:

אם תנאי 1 הוא BEGIN מתבצעת סדרת פקודות לביצוע 1,

אם תנאי 2 הוא BEGIN מתבצעת סדרת פקודות לביצוע 2,

וכן הלאה...

בשלב השני מתבצע מעבר על כל שורות הקלט ועבור כל שורת קלט מבצעים את כל סדרות הפקודות לביצוע שהתנאי המקדים להן מתקיים. במילים אחרות:

אם תנאי 1 מתקיים מתבצעת סדרת פקודות לביצוע 1,

אם תנאי 2 מתקיים מתבצעת סדרת פקודות לביצוע 2,

וכן הלאה...

בשלב השלישי (אחרי שכל שורות הקלט נקראו) מבצעים את כל סדרות הפקודות שהתנאי המקדים להן הוא END. במילים אחרות:

אם תנאי 1 הוא END מתבצעת סדרת פקודות לביצוע 1,

אם תנאי 2 הוא END מתבצעת סדרת פקודות לביצוע 2, וכן הלאה...

להלן דוגמה של הפעלת awk על ידי קריאה לתוכנית awk בשם B1. התוכנית בנויה משלוש סדרות של פקודות, כך שכל סדרת פקודות מכילה פקודה אחת בלבד. מאחר ואין תנאי מקדים לפני הסדרות של הפקודות, כל הסדרות מתבצעות עבור כל שורת קלט.

נניח שתוכן הקובץ B1 הוא:

```
basicsys@mars~/lec9>cat B1

{print "hi"}
{print $1+$2}
{print $1*$2}
```

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec9>cat F1

10 20 30
40 45
3.2 5.5 6.3
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec9>awk -f B1 F1
```

מתקבל הפלט:

```
hi
30
200
hi
85
```

```
1800
hi
8.7
17.6
```

בדוגמה הבאה נוסיף לתחילת התוכנית הקודמת את השורה:
```
#!/bin/awk -f
```
ונפעיל אותה כתכנית סקריפט.

נניח שתוכן הקובץ C1 הוא:

```
basicsys@mars~/lec9>cat C1

#!/bin/awk -f
{print "hi"}
{print $1+$2}
{print $1*$2}
```

כדי שנוכל להפעיל את הקובץ C1 כתכנית סקריפט אנחנו צריכים להוסיף לו הרשאת הרצה:

```
basicsys@mars~/lec9>chmod u+x C1
```

לאחר הפעלת הפקודה:
```
basicsys@mars~/lec9>C1 F1
```
מתקבל הפלט:
```
hi
30
200
hi
85
1800
hi
8.7
17.6
```

נניח שתוכן הקובץ B1 הוא:

```
basicsys@mars~/lec9>cat B1
{print hi}
{print $1+$2}
{print $1*$2}
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec9>awk -f B1 F1
```

מתקבל הפלט:

```
30
200

85
1800

8.7
17.6
```

בדוגמה הנ"ל מאחר ולא רשמנו את hi בתוך גרשיים awk חושב ש- hi הוא שם של משתנה. מאחר ואין למשתנה hi ערך התחלתי והשימוש שנעשה בו הוא כמחרוזת (ולא בביטוי חשבוני), הערך ההתחלתי שלו הוא מחרוזת ריקה והפקודה: print hi מדפיסה שורה ריקה.

הדוגמה הבאה מראה שימוש בתנאים BEGIN ו- END. מאחר ואין כל קשר בין תנאי ה- BEGIN לתנאי ה- END, אין לצפות שלכל BEGIN יהיה END שמתאים לו.

נניח שתוכן הקובץ B1 הוא:

```
basicsys@mars~/lec9>cat B1

BEGIN {print "hi"}
{print $1+$2}
{print $1*$2}
BEGIN {print "hello"}
END {print "NR="NR}
BEGIN {print "ok"}
END {print "END"}
```

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec9>cat F1

10 20 30
40 45
3.2 5.5 6.3
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec9>awk -f B1 F1
```

מתקבל הפלט:

```
hi
hello
ok
30
200
85
1800
8.7
17.6
NR=3
END
```

נניח שתוכן הקובץ P1 הוא:

```
basicsys@mars~/lec9>cat P1

BEGIN {print ARGC}
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec9>awk -f P1 F1 F2 F3
```

מתקבל הפלט:

```
4
```

בדוגמה הבאה מציבים ערך xy למשתנה המערכת OFS וכתוצאה מכך הפקודה print מדפיסה xy בין הפרמטרים שמועברים אליה.

נניח שתוכן הקובץ P1 הוא:

```
basicsys@mars~/lec9>cat P1
BEGIN {OFS="xy"; print "ab","cd"}
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec9>awk -f P1
```

מתקבל הפלט:

```
abxycd
```

נניח שתוכן הקובץ P1 הוא:

```
basicsys@mars~/lec9>cat P1
BEGIN {OFS="xy"; print "ab","cd"
       print "de","fg"}
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec9>awk -f P1
```

מתקבל הפלט:

```
abxycd
dexyfg
```

בדוגמה הבאה מציבים ערך xy למשתנה המערכת OFS
ומציבים ערך zz למשתנה המערכת ORS וכתוצאה מכך
הפקודה print מדפיסה xy בין הפרמטרים שמועברים אליה
ובסוף הפקודה print מדפיסה zz (במקום להדפיס n\).

נניח שתוכן הקובץ P1 הוא:

```
basicsys@mars~/lec9>cat P1

BEGIN {OFS="xy"; ORS="zz"
       print "ab","cd"
       print "de","fg"}
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec9>awk -f P1
```

מתקבל הפלט:

```
abxycdzzdexyfgzzbasicsys@mars~/lec9>
```

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec9>cat F1

abc def
ghe
a b c d
```

נניח שתוכן הקובץ F2 הוא:

```
basicsys@mars~/lec9>cat F2

100
20000
25
```

לאחר הפעלת הפקודה:

```
sys@mars~/lec9>awk '{print FILENAME,$0}' F1 F2
```

מתקבל הפלט:

```
F1 abc def
F1 ghe
F1 a b c d
F2 100
F2 20000
F2 25
```

נניח שתוכן הקובץ F3 הוא:

```
basicsys@mars~/lec9>cat F3

abc
```

הדוגמה הבאה, מציגה שימוש במשתנה המערכת ARGV שהינו מערך שמכיל במקום 0 את המחרוזת awk, במקום 1 את הפרמטר הראשון, במקום 2 את הפרמטר השני, וכן הלאה.

נניח שתוכן התכנית A1 הוא:

```
basicsys@mars~/lec9>cat A1

BEGIN { for (i=0; i<length(ARGV); i++) {
    print "ARGV["i"]="ARGV[i]
    }
}
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec9>awk -f A1 F1 F2 F3
```

מתקבל הפלט:

```
ARGV[0]=awk
ARGV[1]=F1
ARGV[2]=F2
ARGV[3]=F3
```

מאחר וסדרת הפקודות בתכנית A1 מופעלת רק בשלב ה-
BEGIN כשקוראים לתכנית A1 עם שם קובץ שלא קיים R3,
לא מקבלים הודעת שגיאה, כפי שמוצג בדוגמה הבאה.

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec9>awk -f A1 F1 F2 R3
```

מתקבל הפלט:

```
ARGV[0]=awk
ARGV[1]=F1
ARGV[2]=F2
ARGV[3]=R3
```

נניח שתכן התכנית A1 הוא:

```
basicsys@mars~/lec9>cat A1

BEGIN { for (i=0; i<length(ARGV); i++) {
    print "ARGV["i"]="ARGV[i]
    }
}
{print FILENAME,$1}
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec9>awk -f A1 F1 F2 R3
```

מתקבל הפלט:

```
ARGV[0]=awk
ARGV[1]=F1
ARGV[2]=F2
ARGV[3]=R3
F1 abc
```

```
F1 ghe
F1 a
F2 100
F2 20000
F2 25
awk: A1:6: (FILENAME=F2 FNR=3) fatal: cannot open
file `R3' for reading (no such file or directory)
```

בדוגמה הנ"ל כשהתכנית עוברת את שלב ה- BEGIN היא קוראת שורות מהקבצים F1 ו- F2 ואז כשהיא מגיעה לקובץ R3 היא מנסה לפתוח אותו לקריאה, ואז מדפיסה הודעת שגיאה שהקובץ R3 לא קיים.

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec9>awk -f A1 F1 F2 F3
```

מתקבל הפלט:

```
ARGV[0]=awk
ARGV[1]=F1
ARGV[2]=F2
ARGV[3]=F3
F1 abc
F1 ghe
F1 a
F2 100
F2 20000
F2 25
F3 abc
```

התכנית הבאה מחשבת את מספר השורות, המילים והתווים בקובץ בדומה לפקודה wc של bash.

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec9>cat P1

{ nc = nc + length($0) + 1
nw = nw + NF
}
END {print NR, "lines", nw, "words", nc, "chars"}
```

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec9>cat F1

abc
de
a x
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec9>awk -f P1 F1
```

מתקבל הפלט:

```
3 lines 4 words 11 chars
```

כאשר מפעילים את התכנית הנ"ל על יותר מקובץ אחד התוצאה המתקבלת היא סכום השורות המילים והתווים בכל הקבצים.

נניח שתוכן הקובץ F2 הוא:

```
basicsys@mars~/lec9>cat F2

abdd
e f
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec9>awk -f P1 F1 F2
```

מתקבל הפלט:

```
5 lines 7 words 20 chars
```

התכנית הבאה מדפיסה לכל קובץ שורה נפרדת שמכילה את שם הקובץ ולאחר מכן את מספר השורות המילים והתווים בקובץ.

נניח שתוכן התכנית P2 הוא:

```
basicsys@mars~/lec9>cat P2

BEGIN {fname=ARGV[1]}
(FILENAME == fname) {
 nc=nc+length($0)+1
}
(FILENAME != fname) {
 print fname, nc, "chars"
 fname=FILENAME
 nc=length($0)+1
}
END {
 print fname, nc, "chars"
}
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec9>awk -f P2 F1 F2
```

מתקבל הפלט:

```
F1 11 chars
F2 9 chars
```

הרצאה מספר 10

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.

אתחול איבר במערך מתבצע על ידי פקודה:

ערך = [אינדקס] שם המערך

לדוגמה הפקודה x["ab"]=12 מאתחלת את המערך x כך שבאינדקס ab הערך הוא 12.

הוצאת איבר ממערך מתבצעת על ידי הפקודה:

delete שם המערך [אינדקס]

לדוגמה הפקודה:

```
delete x["ab"]
```

מוציאה את האיבר בעל אינדקס ab מהמערך x.

מחיקת כל אברי מערך מתבצעת על ידי הפקודה:

delete שם המערך

לדוגמה הפקודה delete x מוחקת את כל אברי המערך x.

לבדיקה האם איבר נמצא במערך אפשר להשתמש בפקודה:

if (אינדקס in שם המערך) {...}

לדוגמה:

```
if ( "ab" in x ) {...}
```

בודק אם האיבר שהאינדקס שלו הוא ab נמצא במערך x.

לקבלת מספר אברי המערך אפשר להשתמש בפקודה:

length (שם המערך)

לסריקת כל אברי המערך אפשר להשתמש במבנה הבא:

```
for ( שם משתנה in שם המערך ) { … }
```

בכל מעבר בלולאה המשתנה יקבל ערך של אינדקס של איבר במערך.

הסדר של אברי המערך נקבע על ידי המערכת ולכן אין לדעת באיזה סדר יסרקו אברי המערך.

התכנית הבאה יוצרת מערך A, מאתחלת אותו ולאחר מכן סורקת אותו ומדפיסה את איבריו. שימו לב שאין קשר בין הסדר שבו מאותחלים אברי המערך לסדר שבו הם נסרקים. הסדר שלפיו נסרקים אברי המערך שמור במערכת (ובכל סריקה ישמר סדר זה), אבל אנחנו לא יודעים מראש מה יהיה סדר זה ואין לנו שליטה עליו.

נניח שנתונה התכנית:

```
basicsys@mars~/lec10>cat A1

BEGIN {A[12]="abc";A["abc"]=100;A["de"]="w100z";
      for (y in A) {
        print y,"A["y"]="A[y]
      }
}
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f A1
```

מתקבל הפלט:

```
de A[de]=w100z
abc A[abc]=100
12 A[12]=abc
```

התכנית הבאה יוצרת מערך A ומדפיסה את מספר האיברים במערך על ידי שימוש ב- length(A). בנוסף לכך התכנית מחשבת את מספר האיברים במערך על ידי סריקת אברי המערך וקידום מונה i בכל מעבר על איבר במערך.

נניח שתוכן התכנית A2 הוא:

```
basicsys@mars~/lec10>cat A2

BEGIN {A[12]="abc";A["abc"]=100;A["de"]="w100z";
      print "length of A is:", length(A)
      for (y in A) {i++}
      print "i=", i
      }
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f A2
```

מתקבל הפלט:

```
length of A is: 3
i= 3
```

הפונקציה substr ב- awk

מבנה הפונקציה:

```
substr ( מחרוזת1, מספר1 ,מספר2  )
```

הפונקציה מחזירה את תת המחרוזת של מחרוזת1 החל מתו מספר1 (מספור התווים מתחיל מ- 1) כאשר לוקחים מספר2 תווים. אם מספר2 לא קיים אז הפונקציה מחזירה את המחרוזת החל מתו מספר1 ועד סוף המחרוזת

לדוגמה, אם המשתנה s מכיל את המחרוזת abcde אזי

```
substr(s,2,3)
```

מחזירה bcd.

נניח שתוכן התכנית A3 הוא:

```
basicsys@mars~/lec10>cat A3

BEGIN { s="abcdef";
  print "substr(s,2,3)="substr(s,2,3)
  print "substr(s,2)="substr(s,2)
  print "substr(\"abcdef\",2,3)="substr("abcdef",2,3)
```

```
      }
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f A3
```

מתקבל הפלט:

```
substr(s,2,3)=bcd
substr(s,2)=bcdef
substr("abcdef",2,3)=bcd
```

הפונקציה length ב- awk

מבנה הפונקציה: length(שם משתנה)

אם המשתנה מכיל מחרוזת הפונקציה מחזירה את מספר התווים במחרוזת,

אם המשתנה מכיל מערך הפונקציה מחזירה את מספר האיברים במערך.

התכנית הבאה מדפיסה שורה אחת שמכילה מספרים כאשר המספר במקום ה- i מכיל את אורך המילה הארוכה ביותר מבין כל המילים שנמצאות בשדה מספר i בקובץ.

נניח שתוכן התכנית A4 הוא:

```
basicsys@mars~/lec10>cat A4

{
  if (NF > max_nf) {
  max_nf = NF
  }
  for (i=1 ; i <= NF ; i++ ) {
    if (length($i) > A[i] ) {
       A[i] = length($i)
    }
  }
}
END { i=1
      while (i <= max_nf) {
        s = s " " A[i]
        i++
      }
      s = substr(s,2)
```

```
    print s
  }
```

נניח שתוכן הקובץ F3 הוא:

```
basicsys@mars~/lec10>cat F3

abcddd  eee ffffff abc
g hhhhh abc z
a b c d e f g h
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f A4 F3
```

מתקבל הפלט:

```
6 5 6 3 1 1 1 1
```

הפונקציה sub ב- awk

מבנה הפונקציה:

sub (שם משתנה, מחרוזת1 ,ביטוי רגולארי)

הפונקציה משנה את הערך של המשתנה על ידי החלפת תת המחרוזת (הראשונה משמאל) של ערך המשתנה שמתאימה לביטוי הרגולארי במחרוזת1. לדוגמה אם המשתנה s מכיל את המחרוזת cdababdab אזי לאחר ביצוע

```
sub("(ab)+","xyz",s)
```

המשתנה s יכיל cdxyzdab

במחרוזת1 ניתן להשתמש בסימון המיוחד & שמשמעותו: החלף אותו בחלק של של הערך של המשתנה שהתאים לביטוי הרגולארי. לדוגמה אם המשתנה s מכיל את המחרוזת cdababdab אזי לאחר ביצוע:

```
sub("(ab)+","&xyz&",s)
```

המשתנה s יכיל cdababxyzababdab

אם שם המשתנה לא קיים, אז ברירת המחדל היא המשתנה $0 (דהינו השורה הנוכחית ש- awk עובד עליה), ולכן במקרה זה השורה הנוכחית משתנה בהתאם לכללים הנ"ל.

הפונקציה sub מחזירה את מספר ההחלפות שבוצעו. מאחר והפונקציה sub מבצעת לכל היותר החלפה אחת, הפונקציה מחזירה 0 או 1.

הפונקציה gsub ב- awk

מבנה הפונקציה:

gsub (שם משתנה, מחרוזת1 ,ביטוי רגולארי)

הפונקציה פועלת באופן דומה לפונקציה sub אלא שמבצעת ההחלפות של כל תתי המחרוזת של ערך המשתנה שמתאימות לביטוי הרגולארי.

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

```
gsub(“(ab)+”,"xyz",s)
```

המשתנה s יכיל cdxyzwxyzz

לדוגמה, אם המשתנה s מכיל את המחרוזת: 12abc354

אזי לאחר ביצוע

```
gsub("[1-4]","&&",s)
```

המשתנה s יכיל 1122abc33544

נניח שתוכן הקובץ F3 הוא:

```
basicsys@mars~/lec10>cat F3

abcdddabc eee ffffff abc
g hhhhh abc z
```

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec10>cat P1

{
  print $0
  sub("abc",111)
  print $0
  }
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P1 F3
```

מתקבל הפלט:

```
abcdddabc eee ffffff abc
111dddabc eee ffffff abc
g hhhhh abc z
g hhhhh 111 z
```

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec10>cat P1

{
  print $0
  gsub("abc",111)
  print $0
  }
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P1 F3
```

מתקבל הפלט:

```
abcdddabc eee ffffff abc
111ddd111 eee ffffff 111
g hhhhh abc z
g hhhhh 111 z
```

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec10>cat P1

{
  print $0
  gsub("abc",111,$1)
  print $0
  }
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P1 F3
```

מתקבל הפלט:

```
abcdddabc eee ffffff abc
111ddd111 eee ffffff abc
g hhhhh abc z
g hhhhh abc z
```

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec10>cat P1

BEGIN {
       s="abcdef"
       gsub("[adf]",111,s)
       print s
      }
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P1
```

מתקבל הפלט:

```
111bc111e111
```

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec10>cat P1

BEGIN {
       s="ab12cd34"
       x=gsub("[1-3]","&&",s)
       print "x=" x
       print "s=" s
      }
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P1
```

מתקבל הפלט:

```
x=3
s=ab1122cd334
```

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec10>cat P1

BEGIN { s="ab12cd34"
        x=sub("[1-3]","&&",s)
        print "x=" x
        print "s=" s
}
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P1
```

מתקבל הפלט:

```
x=1
s=ab112cd34
```

המשתנה FS ב- awk

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

$1 יקבל ערך z

$2 יקבל ערך cdcd

$3 יקבל ערך ef

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 לרווח אחד ומכניס 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 יקבל ערך a

$2 יקבל ערך b

$3 יקבל ערך c

NF יקבל ערך 3

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec10>cat P1

BEGIN{FS="(ab)+";OFS="@"}
{print $1,$2,$3,$4,NF}
```

לאחר הפעלת הפקודה:

```
~/lec10>echo "cdababxyabab  zabwr" | awk -f P1
```

מתקבל הפלט:

```
cd@xy@  z@wr@4
```

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec10>cat P1

BEGIN{FS="";OFS="@"}
{print $1,$2,$3,$4,NF}
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>echo abcd | awk -f P1
```

מתקבל הפלט:

```
a@b@c@d@4
```

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec10>cat P1

BEGIN{FS=""}
{
 for (i=1 ; i <= NF ; i++ ) {
  print $i
 }
}
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>echo abcd | awk -f P1
```

מתקבל הפלט:

```
a
b
c
d
```

הפונקציה split ב- awk

מבנה הפונקציה:

```
split ( ביטוי רגולארי, שם מערך ,מחרוזת )
```

הפונקציה 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 יכיל את שלושת האיברים הבאים בלבד:

```
A[1]=x

A[2]=cd

A[3]=e
```

אם המערך 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….

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec10>cat P1

BEGIN {
  n=split("1aaabc2abc3",A,"a")
  for (i in A) {
   print "A[" i "]=" A[i]
  }
  print "n=" n
  for (i=1; i<=length(A); i++ ){
   print "A[" i "]=" A[i]
  }
}
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P1
```

מתקבל הפלט:

```
A[4]=bc2
A[5]=bc3
A[1]=1
A[2]=
A[3]=
n=5
A[1]=1
A[2]=
A[3]=
A[4]=bc2
A[5]=bc3
```

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec10>cat P1

BEGIN {
  x["ab"]=8;split("abcd",x,"")
  for (i in x) {
   print "x["i"]=" x[i]
  }
}
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P1
```

מתקבל הפלט:

```
x[4]=d
x[1]=a
x[2]=b
x[3]=c
```

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec10>cat P1

BEGIN {
  for (i=1; i<=100; i++ ){
   A[i]=i*2
  }
  print "before length(A)=" length(A)
  split("",A)
  print "after length(A)=" length(A)
}
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P1
```

מתקבל הפלט:

```
before length(A)=100
after length(A)=0
```

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec10>cat P1

BEGIN {
  split("123   abc   ge",A)
  for (i=1; i<=length(A); i++ ){
   print "A[" i "]=" A[i]
  }
}
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P1
```

מתקבל הפלט:

```
A[1]=123
A[2]=abc
A[3]=ge
```

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec10>cat P1

BEGIN {
  FS="a"
  split("123   abc   ge",A)
  for (i=1; i<=length(A); i++ ){
   print "A[" i "]=" A[i]
  }
}
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P1
```

מתקבל הפלט:

```
A[1]=123
A[2]=bc   ge
```

נניח שתוכן התכנית P1 הוא:

התכנית הבאה מדפיסה את תוכן שורות הקלט לאחר מחיקת כל הרווחים בין המילים שבשורות הקלט.

```
basicsys@mars~/lec10>cat P1

BEGIN {ORS=""}
{ for (j=1;j <=NF; j++) {
  A[NR,j]=$j
 }
}
END { for (i=1; i<=NR; i++) {
        for (j=1; j<= NF; j++) {
          print A[i,j]
        }
        print "\n";
      }
    }
```

נניח שתוכן הקובץ F3 הוא:

```
basicsys@mars~/lec10>cat F3

abcddd  eee ffffff abc
g hhhhh abc z
a b c d e f g h
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P1 F3
```

מתקבל הפלט:

```
abcdddeeeffffffabc
ghhhhhabcz
abcdefgh
```

הפונקציה system ב- awk

מבנה הפונקציה:

```
system ( מחרוזת )
```

הפונקציה מבצעת את המחרוזת כפקודת bash.

לדוגמה, הפקודה:

```
system("sort F1")
```

תציג על המסך את שורות הקובץ F1 ממויינות בסדר לקסיקוגרפי בהתאם לפקודה sort של bash.

הפונקציה getline ב- awk

יש מספר אפשרויות שימוש בפונקציה getline :

```
getline

getline שם משתנה

getline <  "שם קובץ"

getline   שם משתנה  <  "שם קובץ"
```

להלן פרוט המשמעות של כל אחת מהאפשרויות הנ"ל.

getline (ללא פרמטרים) גורמת לכך שמתבצע מעבר לשורת הקלט הבאה באופן מידי. המשתנים:

NF,NR,FNR,$0,$1,$2,...

מתעדכנים לפי שורת הקלט הבאה.

שם משתנה getline גורמת לכך שהמשתנה יקבל ערך ששווה לשורת הקלט הבאה. ערכי המשתנים:

NF,$0,$1,$2,...

נשארים ללא שינוי.

לדוגמה, לאחר הפקודה getline x ערך המשתנה x ישווה לשורת הקלט הבאה. וערך המשתנה $0 יישאר ללא שינוי.

הפקודה:

getline < "שם קובץ"

גורמת לכך שערכי המשתנים:

NF,$0,$1,$2,...

מתעדכנים לפי שורת הקלט הבאה מתוך הקובץ. ערכי המשתנים NR ו- FNR נשארים ללא שינוי.

לדוגמה, לאחר הפעלת הפקודה:

getline <"F1"

ערכי המשתנים:

NF,$0,$1,$2,...

מתעדכנים לפי השורה הראשונה בקובץ F1.

לאחר הפעלה נוספת של הפקודה:

getline <"F1"

ערכי המשתנים הנ"ל מתעדכנים לפי השורה השנייה בקובץ F1, וכן הלאה.

הפונקציה:

close("F1")

סוגרת את הקובץ F1 וגורמת לכך שלאחר הפעלה נוספת של הפקודה:

```
getline <"F1"
```

המשתנים הנ"ל יתעדכנו לפי השורה הראשונה בקובץ.

הפקודה:

```
getline  שם משתנה  < "שם קובץ"
```

גורמת לכך שהמשתנה יקבל ערך ששווה לשורת הקלט הבאה מהקובץ ערכי המשתנים:

```
NF,NR,FNR,$0,$1,$2,...
```

נשארים ללא שינוי.

לדוגמה, לאחר הפקודה:

```
getline x <"F1"
```

ערך המשתנה x ישווה לשורה הראשונה בקובץ F1. לאחר הפעלה נוספת של הפקודה הנ"ל ערך המשתנה x ישווה לשורה השנייה בקובץ F1 וכן הלאה...

בכל הצורות הנ"ל של שימוש בפונקציה getline הפונקציה מחזירה:

1 אם הפונקציה הצליחה לקרוא שורה (מהקלט או מהקובץ בהתאם לצורה בה משתמשים בפונקציה).
0 אם הפונקציה לא הצליחה לקרוא שורה, כי הסמן נמצא בסוף הקובץ (בקריאה מהקובץ) או בסוף הקלט (בקריאה מהקלט.
1- אם הפונקציה לא הצליחה לקרוא שורה (מהקלט או מהקובץ) בגלל שגיאה.

להלן דוגמאות לשימוש בפונקציה system ובפונקציה getline.

ננית שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec10>cat F1

30
200
10
```

ננית שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec10>cat P1

BEGIN {
       system("sort -n F1")
}
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P1
```

מתקבל הפלט:

```
10
30
200
```

ננית שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec10>cat P1

BEGIN {
       system("echo abc>|F2")
       system("echo def>>F2")
       system("cat F2")
}
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P1
```

מתקבל הפלט:

```
abc
def
```

ננית שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec10>cat P1

BEGIN {
       getline <"F1"
       print $0
       getline <"F1"
       print $0
}
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P1
```

מתקבל הפלט:

```
30
200
```

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec10>cat P1

BEGIN {
      getline <"F1"
      print $0
      getline <"F1"
      print $0
      close("F1")
      getline <"F1"
      print $0
}
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P1
```

מתקבל הפלט:

```
30
200
30
```

ננית שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec10>cat P1

BEGIN {
      getline x <"F1"
      print "$0=" $0
      print "x=" x
      getline x <"F1"
      print "$0=" $0
      print "x=" x
}
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P1
```

מתקבל הפלט:

```
$0=
x=30
$0=
x=200
```

ננית שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec10>cat P1

BEGIN {
      y=getline x<"F1"
      print "x=" x,"y=" y
      y=getline x<"F1"
      print "x=" x,"y=" y
      y=getline x<"F1"
      print "x=" x,"y=" y
      y=getline x<"F1"
      print "x=" x,"y=" y
      y=getline z<"F1"
      print "z=" z,"y=" y
}
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P1
```

מתקבל הפלט:

```
x=30 y=1
x=200 y=1
x=10 y=1
x=10 y=0
z= y=0
```

התכנית הבאה מדפיסה את אברי המערך A לפי סדר האינדקסים שלהם (בהנחה שהאינדקסים הם מספרים). עבור כל איבר במערך A מודפסת שורה אחת שמכילה את האינדקס של האיבר, והערך של האיבר (כשבינם תו רווח אחד).

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec10>cat P1
```

```
BEGIN {
      A[8]="ab"
      A[30]=15
      A[4]=70
      system("echo -n "" >|tmp")
      for (x in A) {
      system("echo " x " >>tmp")}
      system("sort -n tmp>|tmp1")
      while (getline x < "tmp1") {
          print x,A[x]
      }
}
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P1
```

מתקבל הפלט:

```
4 70
8 ab
30 15
```

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec10>cat P1

{ print $0
  getline
  print $0
}
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P1 F1
```

מתקבל הפלט:

```
30
200
10
10
```

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec10>cat P1

{ print $0
  getline x
  print $0
}
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P1 F1
```

מתקבל הפלט:

```
30
30
10
10
```

נניח שתוכן הקובץ F2 הוא:

```
basicsys@mars~/lec10>cat F2

ab
cd
```

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec10>cat P1

{ print $0
  getline x<"F2"
  print "x=" x
  print $0
}
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P1 F1
```

מתקבל הפלט:

```
30
x=ab
30
200
x=cd
200
10
x=cd
10
```

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

הפקודה: print מחרוזת1 > "שם קובץ"

ב- awk פותחת את הקובץ לכתיבה (החל מתחילת הקובץ) וכותבת את מחרוזת1 בשורה הראשונה של הקובץ. אם הקובץ לא קיים הפקודה יוצרת אותו. אם הקובץ קיים אז תוכנו נדרס ולאחר ביצוע הפקודה מופיעה בו רק מחרוזת1 בשורה הראשונה.

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

ננית שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec10>cat F1

aaa
```

ננית שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec10>cat P1

BEGIN { print "abc" >"F1"
        print "def" >"F1"
        print "gh" >"F1"
      }
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P1
```

תוכן הקובץ F1 הוא:

```
basicsys@mars~/lec10>cat F1

abc
def
gh
```

ננית שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec10>cat P1

BEGIN { print "abc" >"F1"
        print "def" >"F1"
        print "gh" >"F1"
        close("F1")
        print "xyz">"F1"
      }
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P1
```

תוכן הקובץ F1 הוא:

```
basicsys@mars~/lec10>cat F1

xyz
```

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

הפקודה: print 1 מחרוזת << "שם קובץ"

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 1 print כותבת את המחרוזת לסוף הקובץ.

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec10>cat P1

BEGIN { print "abc" >>"F1"
        print "def" >"F1"
        print "gh" >>"F1"
      }
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P1
```

תוכן הקובץ F1 הוא:

```
basicsys@mars~/lec10>cat F1

xyz
abc
def
gh
```

פונקציות שמוגדרות על ידי המשתמש ב- awk

המבנה של הגדרת פונקציה:

```
function שם הפונקציה ( רשימת ארגומנטים )
{
    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 f1 יקבל ערך 5. השמת ערך 6 למשתנה a בתוך הפונקציה f1 לא תשנה את ערך המשתנה x. לכן העברת פרמטרים כזו נקראת העברה by value.

אם x הוא משתנה מסוג מערך וננית של- x[1] יש ערך 5

ונקרא לפונקציה:

```
f1(x)
```

ובתוך הפונקציה f1 יהיה רשום a[1]=8 אזי ביציאה מהפונקציה f1 הערך של x[1] יהיה 8. לכן העברת פרמטרים כזו נקראת העברה **by reference**.

הפקודה:

return ערך

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.

הדוגמה הבאה מראה שערך של משתנה שמועבר כארגומנט ומשתנה בתוך הפונקציה, לא משתנה מחוץ לפונקציה.

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec10>cat P1

BEGIN {x=1;
       print "before f1: x=" x
       f1(x)
       print "after f1: x=" x
      }
function f1(x) {
  print "in f1: x=" x
  x=8
}
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P1
```

מתקבל הפלט:

```
before f1: x=1
in f1: x=1
after f1: x=1
```

הדוגמה הבאה מראה שמשתנה שמוגדר כארגומנט של פונקציה, ולא הוגדר לפני הקריאה לפונקציה, אינו קיים מחוץ לפונקציה.

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec10>cat P1
BEGIN {x=1;
       print "before f1: a=" a " x=" x
       f1(x)
       print "after f1: a=" a " x=" x
      }
function f1(a) {
  print "in f1: a=" a " x=" x
  a=8
}
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P1
```

מתקבל הפלט:

```
before f1: a= x=1
in f1: a=1 x=1
after f1: a= x=1
```

הדוגמה הבאה מראה שמשתנה שאינו מוגדר כארגומנט לפונקציה ומאותחל בתוך הפונקציה, קיים גם מחוץ לפונקציה. במילים אחרות המשתנים הלוקליים הם רק אלה שמוגדרים ברשימת הארגומנטים של הפונקציה.

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec10>cat P1

BEGIN {x=1;
       print "before f1: x=" x " z=" z
       f1(x);
       print "after f1: x=" x " z=" z
      }
function f1(a) {
  print "in f1: a=" a " x=" x
  a=8
  x=9
  z=10
}
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P1
```

מתקבל הפלט:

```
before f1: x=1 z=
in f1: a=1 x=1
after f1: x=9 z=10
```

הדוגמה הבאה מראה שכאשר מעבירים לפונקציה משתנה מסוג מערך, מועבר reference למשתנה ולכן השינויים שיעשו במערך בתוך הפונקציה יהיו קיימים במערך גם אחרי היציאה מהפונקציה.

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec10>cat P1

BEGIN {x[1]=8; x["ab"]=10
  print "before f1: x[1]=" x[1] " x[ab]=" x["ab"]
  f1(x)
  print "after f1: x[1]=" x[1] " x[ab]=" x["ab"]
      }
function f1(a) {
  print "inside f1 a[1]=" a[1] " a[ab]=" a["ab"]
  print "inside f1 x[1]=" x[1] " x[ab]=" x["ab"]
  a[1]=12;
  a["ab"]=13;
  x[1]=14
}
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P1
```

מתקבל הפלט:

```
before f1: x[1]=8 x[ab]=10
inside f1 a[1]=8 a[ab]=10
inside f1 x[1]=8 x[ab]=10
after f1: x[1]=14 x[ab]=13
```

עבור מטריצה ריבועית n x n של מספרים נגדיר את היקף המטריצה כריבוע החיצוני של המטריצה, דהינו חלק המטריצה שמכיל את עמודות 1 ו- n ואת שורות 1 ו- n

לדוגמה, עבור המטריצה הבאה:

```
2 2 2 2 2 2
2 1 1 1 1 2
2 1 3 3 1 2
2 1 3 3 1 2
2 1 1 1 1 2
2 2 2 2 2 2
```

ההיקף של המטריצה הוא:

```
2 2 2 2 2 2
2         2
2         2
2         2
2         2
2 2 2 2 2 2
```

עבור n זוגי, נגדיר שמטריצה ריבועית n x n היא טובה, אם כל המספרים שנמצאים בהיקף החיצוני שלה שווים, ואם כאשר נוריד ממנה את ההיקף החיצוני כל המספרים בהיקף החיצוני החדש יהיו שווים וכן הלאה עד שנגיע למטריצה 2 x 2 שכל האיברים שלה שווים. לדוגמה

המטריצה הקודמת היא טובה.

התכנית הבאה בודקת אם קובץ הקלט מכיל מטריצה טובה. במידה וכן התכנית מדפיסה YES, במידה ולא התכנית מדפיסה NO.

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec10>cat F1

2 2 2 2 2 2
2 1 1 1 1 2
2 1 3 3 1 2
2 1 3 3 1 2
2 1 1 1 1 2
2 2 2 2 2 2
```

נניח שתוכן התכנית P2 הוא:

```
basicsys@mars~/lec10>cat P2

{
  for (i=1; i<= NF; i++) {
   A[NR,i]=$i
  }
}
END {
```

```
  for (i=1; i <=NR/2; i++) {
    s=A[i,i];
    for (j=i; j <= NR-(i-1); j++) {
      if (A[i,j] != s) { print "NO" ; exit}
      if (A[j,i] != s) { print "NO" ; exit}
      if (A[NR-(i-1),j] != s)
         { print "NO" ; exit}
      if (A[j,NR-(i-1)] != s)
         { print "NO" ; exit}
    }
  }
print "YES"
}
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P2 F1
```

מתקבל הפלט:

```
YES
```

התכנית הבאה מבצעת אותה משימה כמו של התכנית הקודמת תוך שימוש בפונקציות הבאות:

```
check_row(A,i,j1,j2)
```

בודקת שכל האיברים שנמצאים בשורה i במערך A בין עמודות j1 ו- j2 (כולל עמודות j1 ו- j2) זהים. במידה וכן הפונקציה מחזירה YES, במידה ולא הפונקציה מחזירה NO.

```
check_column(A,i,j1,j2)
```

בודקת שכל האיברים שנמצאים בעמודה i במערך A בין שורות j1 ו- j2 (כולל שורות j1 ו- j2) זהים. במידה וכן הפונקציה מחזירה YES, במידה ולא הפונקציה מחזירה NO.

נניח שתוכן התכנית P3 הוא:

```
basicsys@mars~/lec10>cat P3
{
  for (i=1; i<= NF; i++) {
   A[NR,i]=$i
  }
}
```

```
END {
 for (i=1; i <=NR/2; i++) {
   if (check_row(A,i,i,NR-(i-1))=="NO")
       { print "NO" ; exit}
   if (check_row(A,NR-(i-1),i,NR-(i-1))=="NO")
       { print "NO" ; exit}
   if (check_column(A,i,i,NR-(i-1))=="NO")
       { print "NO" ; exit}
   if (check_column(A,NR-(i-1),i,NR-(i-1))=="NO")
       { print "NO" ; exit}
 }
 print "YES"
}

function check_row(A,i,j1,j2) {
 s=A[i,j1]
 for (p=j1; p<= j2; p++) {
  if (A[i,p] != s) { return "NO" }
 }
 return "YES"
}

function check_column(A,i,j1,j2) {
 s=A[j1,i]
 for (p=j1; p<= j2; p++) {
  if (A[p,i] != s) { return "NO" }
 }
 return "YES" }
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P3 F1
```

מתקבל הפלט:

```
YES
```

התכנית הבאה מבצעת אותה משימה כמו של התכנית הקודמת בהבדל שהיא יכולה לקבל מספר קבצים כפרמטרים, ועבור כל קובץ היא מדפיסה שורה שמכילה את שם הקובץ, לאחר מכן תו :, לאחר מכן תו רווח ולאחר מכן YES או NO אם הקובץ מכיל מטריצה טובה או לא.

השיטה שלפיה פועלת התכנית היא להוסיף את שם הקובץ שאליו שייכת השורה בתחילת האינדקס של האיבר של

המערך A שמכיל את השורה. כתוצאה מכך האינדקסים של כל השורות של קובץ מסוים שנמצאות במערך A מתחילים בשם הקובץ.

נזכיר שהמערכים ב- awk הם חד ממדיים והאינדקסים של איברי מערך ב- awk הם מחרוזות. לפיכך, אם למשל לאיבר במערך יש אינדקס F1,2,3 לא מדובר כאן במערך תלת ממדי, אלא במערך חד ממדי שהאינדקס שלו הוא המחרוזת: F1,2,3 שמכילה בתוכה שני תווי פסיק.

נניח שתוכן התכנית P4 הוא:

```
basicsys@mars~/lec10>cat P4
{
for (i=1; i<= NF; i++) {
  A[FILENAME,FNR,i]=$i
 }
}
END {
     for (x=1; x < length(ARGV); x++) {
      if (check_file(A,ARGV[x],FNR)=="NO")
         { print ARGV[x] ": NO" }
       else
         {print ARGV[x] ": YES"}
     }
}

function check_file(A,file,n) {
 for (i=1; i <=n/2; i++) {
   if (check_row(A,file,i,i,n-(i-1))=="NO")
       { return "NO"}
   if (check_row(A,file,n-(i-1),i,n-(i-1))=="NO")
       { return "NO"}
   if (check_column(A,file,i,i,n-(i-1))=="NO")
       { return "NO" ; exit}
if (check_column(A,file,n-(i-1),i,n-(i-1))=="NO")
       { return "NO" }
  }
  return "YES"
}
```

```
function check_row(A,file,i,j1,j2) {
 s=A[file,i,j1]
 for (p=j1; p<= j2; p++) {
  if (A[file,i,p] != s) { return "NO" }
 }
 return "YES"
}

function check_column(A,file,i,j1,j2) {
 s=A[file,j1,i]
 for (p=j1; p<= j2; p++) {
  if (A[file,p,i] != s) { return "NO" }
 }
 return "YES"
}
```

ננית שתוכן הקובץ F2 הוא:

```
basicsys@mars~/lec10>cat F2

2 2 2 2 2 2
2 1 1 1 1 2
2 1 3 3 1 2
2 1 3 3 1 2
2 1 1 1 1 2
2 2 2 2 2 5
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P4 F1 F2
```

מתקבל הפלט:

```
F1: YES
F2: NO
```

התכנית הקודמת מניחה שכל הקבצים שמועברים לה כפרמטרים מכילים מטריצות ריבועיות באותו גודל.

התכנית הבאה משפרת את התכנית הקודמת בכך שהיא יכולה לקבל כפרמטרים קבצים שמכילים מטריצות ריבועיות בגודל שונה.

השיטה שבה התכנית פועלת היא שימוש במערך נוסף בשם B, כך שעבור כל קובץ, האיבר B[שם הקובץ] מכיל את גודל הקובץ.

נניח שתוכן התכנית P5 הוא:

```
basicsys@mars~/lec10>cat P5

{
for (i=1; i<= NF; i++) {
  A[FILENAME,FNR,i]=$i
  B[FILENAME]=FNR
 }
}
END {
     for (x=1; x < length(ARGV); x++) {
      if (check_file(A,ARGV[x],B[ARGV[x]])=="NO")
           { print ARGV[x] ": NO" }
      else {print ARGV[x] ": YES"}
     }
}


function check_file(A,file,n) {

  for (i=1; i <=n/2; i++) {
   if (check_row(A,file,i,i,n-(i-1))=="NO")
       { return "NO"}
   if (check_row(A,file,n-(i-1),i,n-(i-1))=="NO")
       { return "NO"}
   if (check_column(A,file,i,i,n-(i-1))=="NO")
       { return "NO" ; exit}
if (check_column(A,file,n-(i-1),i,n-(i-1))=="NO")
       { return "NO" }
  }
  return "YES"
}

function check_row(A,file,i,j1,j2) {
 s=A[file,i,j1]
 for (p=j1; p<= j2; p++) {
  if (A[file,i,p] != s) { return "NO" }
 }
 return "YES"
}
```

```
function check_column(A,file,i,j1,j2) {
 s=A[file,j1,i]
 for (p=j1; p<= j2; p++) {
  if (A[file,p,i] != s) { return "NO" }
 }
 return "YES"
}
```

נניח שתוכן הקובץ F3 הוא:

```
basicsys@mars~/lec10>cat F3

2 2 2 2
2 1 1 2
2 1 1 2
2 2 2 2
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P5 F1 F2 F3
```

מתקבל הפלט:

```
F1: YES
F2: NO
F3: YES
```

התכנית הבאה מקבלת כפרמטר קובץ שכל שורה בו היא במבנה הבא:

רשימת השפות שהמתכנת שולט בהן : שם מתכנת
(השפות מופרדות על ידי פסיקים)

התכנית מדפיסה שורה אחת עבור כל שפה שמכילה את שם השפה, לאחריו תו :, לאחריו תו רווח יחיד, ולאחריו רשימת כל המתכנתים ששולטים בשפה. אין חשיבות לסדר בין השורות ולסדר השמות בתוך כל שורה.

ננית שתוכן התכנית P6 הוא:

```
basicsys@mars~/lec10>cat P6
{
  split($0,A,":")
  split(A[2],B,",")
  for (x in B) {
   C[B[x]]= C[B[x]] " " A[1]
   }
}
END {
   for (y in C) {
    print y ":" C[y]
   }
}
```

ננית שתוכן הקובץ F4 הוא:

```
basicsys@mars~/lec10>cat F4

dan:C,C++,JAVA
uri:JAVA,Perl
yossi:Perl,Bash,C
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec10>awk -f P6 F4
```

מתקבל הפלט:

```
Bash: yossi
JAVA: dan uri
C: dan yossi
C++: dan
Perl: uri yossi
```

הרצאה מספר 11

פונקציות ב- Bash

הגדרת פונקציה:

```
function שם הפונקציה {
                    גוף הפונקציה
                }
```

צורה נוספת להגדרת פונקציה:

```
שם הפונקציה () {
                    גוף הפונקציה
                }
```

אפשר לקרוא לפונקציה רק לאחר שהפונקציה הוגדרה.

המשתנים של הפונקציה

המשתנים של הפונקציה שלא הוגדרו על ידי הפקודה local (שתתואר בהמשך) מוכרים גם מחוץ לפונקציה והפונקציה יכולה לשנות אותם.

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec11>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"
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec11>P1
```

מתקבל הפלט:

```
outside before calling f1 x=8 y=
inside f1 x=8
outside after calling f1 x=9 y=10
```

הפקודה local

הפקודה: שם משתנה local
מגדירה משתנה מקומי שמוכר רק בתוך הפונקציה, ונעלם ביציאה מהפונקציה.

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec11>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"
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec11>P1
```

מתקבל הפלט:

```
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=
```

העברת פרמטרים לפונקציה

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 של הקריאה לפונקציה (ולא בפרמטר הראשון של הקריאה לסקריפט שבתוכו מוגדרת הפונקציה).

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec11>cat P1

echo $1 $2
f1() {
  echo $1 $2
  for x in "$@"
   do
     echo "$x"
   done
}
f1
f1 "20    30"  40
echo $1 $2
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec11>P1 400 5
```

מתקבל הפלט:

```
400 5

20 30 40
20    30
40
400 5
```

הפעלת פונקציה על ידי קריאה מהסוג: (שם הפונקציה)$

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

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec11>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
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec11>P1
```

מתקבל הפלט:

```
13
y=13
x=25
y=13
85
y=13
145
y=145
```

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec11>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
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec11>P1
```

מתקבל הפלט:

```
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
```

סקריפט ופונקציה בעלי שם זהה

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

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec11>cat f1

echo "I am file f1"
```

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec11>cat P1

f1() {
 echo I am f1 in P1
}
f1
./f1
f1
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec11>P1
```

מתקבל הפלט:

```
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.

נניח שמבנה התיקיה ~basicsys/win15/ex7/d2/d2

הוא כפי שמתואר על ידי הפקודה הבאה:

```
basicsys@mars~/lec11>tree
~basicsys/win15/ex7/d2/d2
/home/cs/segel/basicsys/win15/ex7/d2/d2
|-- A
`-- d3
    |-- E
    `-- d4
        `-- S
2 directories, 3 files
```

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec11>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 {}" \;
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec11>P1 ~basicsys/win15/ex7/d2/d2
```

מתקבל הפלט:

```
A
E
S
```

שימוש ב- ${שם משתנה}

הצורה שם משתנה$ היא קיצור ל- ${שם משתנה}. כאשר יש שתי אפשרויות שונות לפרוש שם משתנה$ מומלץ להשתמש ב- ${שם משתנה}.

לדוגמה,

```
basicsys@mars~/lec11>x=abcde

basicsys@mars~/lec11>echo $x

abcde
```

```
basicsys@mars~/lec11>echo ${x}
abcde
basicsys@mars~/lec11>xx=5
basicsys@mars~/lec11>echo ${x}x
abcdex
basicsys@mars~/lec11>echo ${xx}
5
basicsys@mars~/lec11>echo $xx
5
```

פעולות על מחרוזות ב- bash

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

```
basicsys@mars~/lec11>x=abcdef
basicsys@mars~/lec11>echo ${#x}
6
```

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

מחרוזת expr length

הפקודה מדפיסה את מספר התווים שנמצאים במחרוזת.

```
basicsys@mars~/lec11>expr length $x
6
```

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~/lec11>expr match abcdef ab.[cd]

4

basicsys@mars~/lec11>expr match abcdef a*.[cd]

3

basicsy~/lec11>expr match abcdef '\(a*.[cd]\)'

abc

expr match ababcdcde '\(\(ab\)\+\(cd\)\+\)'

ababcdcd

expr match ababcdcde '\(\(ab\)\+\)\(cd\)\+'

abab
```

קבוצת תווים מחרוזת expr index

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

```
basicsys@mars~/lec11>expr index abcde db

2

basicsys@mars~/lec11>expr index abcde rc

3

basicsys@mars~/lec11>expr index abcde rt

0
```

expr substr מחרוזת מספר1 מספר2

מחזירה את תת המחרוזת החל מתו מספר1 כאשר לוקחים מספר2 תווים. מספור התווים במחרוזת מתחיל מ- 1.

```
basicsys@mars~/lec11>expr substr abcdefg 3 4

cdef

basicsys@mars~/lec11>expr substr abcdefg 3 7

cdefg
```

${שם משתנה:מספר1:מספר2}

מוחלף בתת המחרוזת שמתקבלת מהמחרוזת שהמשתנה מכיל החל מתו מספר1 כאשר לוקחים מספר2 תווים. במידה ומספר2 לא קיים אז לוקחים תווים עד סוף המחרוזת. מספור התווים מתחיל מ- 0.

```
basicsys@mars~/lec11>x=abcdefg

basicsys@mars~/lec11>echo ${x:3:4}

defg

basicsys@mars~/lec11>echo ${x:3}

defg

basicsys@mars~/lec11>y=${x:2}

basicsys@mars~/lec11>echo "$y"

cdefg
```

${שם משתנה#ביטוי בסגנון גלוב}

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

```
basicsys@mars~/lec11>x=abcdefg

basicsys@mars~/lec11>echo ${x#a??}

defg

basicsys@mars~/lec11>echo ${x#a*}

bcdefg
```

${שם משתנה ## ביטוי בסגנון גלוב}

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

לדוגמה, הפקודה הבאה מדפיסה שורה ריקה.

```
basicsys@mars~/lec11>x=abcdefg

basicsys@mars~/lec11>echo ${x##a*}
```

${שם משתנה % ביטוי בסגנון גלוב}

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

```
basicsys@mars~/lec11>x=abcdefg

basicsys@mars~/lec11>echo ${x%e*}

abcd

basicsys@mars~/lec11>x=fabcdab

basicsys@mars~/lec11>echo ${x%ab*}

fabcd
```

${שם משתנה %% ביטוי בסגנון גלוב}

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

```
basicsys@mars~/lec11>x=fabcdab

basicsys@mars~/lec11>echo ${x%%ab*}

f
```

${שם משתנה / ביטוי בסגנון גלוב / מחרוזת1}

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 במחרוזת1. ערך המשתנה נשאר ללא שינוי.

```
basicsys@mars~/lec11>x=ababcdab

basicsys@mars~/lec11>echo ${x/ab/zzz}

zzzabcdab

basicsys@mars~/lec11>echo ${x/ab*/zzz}

zzz
```

${שם משתנה // ביטוי בסגנון גלוב / מחרוזת1}

מוחלף בתת המחרוזת שמתקבלת מהמחרוזת שהמשתנה מכיל לאחר החלפות של כל תתי המחרוזות שמתאימות לביטוי לפי חוקי גלוב במחרוזת1. ערך המשתנה נשאר ללא שינוי.

```
basicsys@mars~/lec11>x=ababcdab

basicsys@mars~/lec11>echo ${x//ab/zzz}

zzzzzzcdzzz
```

הפקודה rev

מבנה הפקודה: רשימת שמות קבצים rev

הפקודה מדפיסה את תוכן הקבצים לאחר ביצוע reverse על השורות שלהם.

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec11>cat F1

abc   def
gh  123 45
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec11>rev F1
```

מתקבל הפלט:

```
fed   cba
54 321  hg
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec11>echo abc | rev
```

מתקבל הפלט:

```
cba
```

הרחבת סוגריים מסולסלים (brace expansion)

הביטוי:

{ מחרוזת 3 , מחרוזת 4 } { מחרוזת 3 , מחרוזת 2 , מחרוזת 1 }

מוחלף על ידי הסורק בכל האפשרויות של הצרופים שלהם.

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec11>echo a{b,c}e
```

מתקבל הפלט:

```
abe ace
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec11>echo a{b,c}{1,2,3}r
```

מתקבל הפלט:

```
ab1r ab2r ab3r ac1r ac2r ac3r
```

הביטוי {מספר2..מספר1}

מוחלף על ידי הסורק בסדרת המספרים החל ממספר1 ועד מספר2 (כולל).

לאחר הפעלת הפקודה:

```
basics~/lec11>for x in {1..10}; do echo $x ; done
```

מתקבל הפלט:

```
1
2
3
4
5
6
7
8
9
10
```

לאחר הפעלת הפקודה:

```
basi~/lec11>for x in $(seq 10); do echo $x ; done
```

מתקבל הפלט:

```
1
2
3
4
5
6
7
8
9
10
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec11>echo {1..10}
```

מתקבל הפלט:

```
1 2 3 4 5 6 7 8 9 10
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec11>echo {3..10}
```

מתקבל הפלט:

```
3 4 5 6 7 8 9 10
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec11>echo {c h}
```

מתקבל הפלט:

```
{c h}
```

הביטוי {תו 2 .. תו 1}

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

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec11>echo {c..h}
```

מתקבל הפלט:

```
c d e f g h
```

הפקודה eval

מבנה הפקודה: רשימת פרמטרים eval

הסורק עובר על הפרמטרים ויוצר מהם מחרוזת. לאחר מכן הפקודה eval גורמת לכך שהמחרוזת שנוצרה מופעלת כפקודת bash.

בדוגמאות הבאות נראה שרק על ידי שימוש ב- eval ניתן להדפיס סדרת מספרים החל ממספר שערכו נמצא במשתנה x ועד מספר שערכו נמצא במשתנה y, בעזרת הרחבת סוגריים מסולסלים.

```
basicsys@mars~/lec11>x=3
basicsys@mars~/lec11>y=8
                                    לאחר הפעלת הפקודה:
basicsys@mars~/lec11>echo {$x..$y}
                                          מתקבל הפלט:
{3..8}
                                    לאחר הפעלת הפקודה:
basicsys@mars~/lec11>echo "{$x..$y}"
                                          מתקבל הפלט:
{3..8}
                                    לאחר הפעלת הפקודה:
basicsys@mars~/lec11>echo \{$x..$y\}
                                          מתקבל הפלט:
{3..8}
                                    לאחר הפעלת הפקודה:
basicsys@mars~/lec11>eval echo {$x..$y}
                                          מתקבל הפלט:
3 4 5 6 7 8
                                    לאחר הפעלת הפקודה:
basicsys@mars~/lec11>echo $(echo {$x..$y})
                                          מתקבל הפלט:
{3..8}
```

מערכים ב- bash

הגדרת מערך:

שם המערך = (רשימת אברי המערך מופרדים על ידי רווח)

אברי המערך יאותחלו באופן הבא: האיבר הראשון ברשימה יהיה באינדקס 0, האיבר השני ברשימה יהיה באינדקס 1 וכן הלאה..

```
basicsys@mars~/lec11>a=20
basicsys@mars~/lec11>echo $a
20
basicsys@mars~/lec11>a=(20 30 40)
basicsys@mars~/lec11>echo $a
20
```

${שם מערך[אינדקס]}

מוחלף בערך של האיבר במערך שנמצא באינדקס הנתון.

```
basicsys@mars~/lec11>a=(20 30 40)
basicsys@mars~/lec11>echo ${a[0]}
20
basicsys@mars~/lec11>echo ${a[1]}
30
basicsys@mars~/lec11>echo ${a[2]}
40
```

${שם מערך[@]}

מוחלף ברשימת כל אברי המערך (לאחר צמצום רווחים).

```
basicsys@mars~/lec11>a=(20 30 40)
basicsys@mars~/lec11>echo ${a[@]}
20 30 40
basicsys@mars~/lec11>echo "${a[@]}"
20 30 40
```

"${שם מערך[@]}"

מוחלף ברשימת כל אברי המערך (ללא צמצום רווחים).

```
basicsys@mars~/lec11>a=("ab  10" "cd   20" 40)
basicsys@mars~/lec11>echo "${a[@]}"
ab  10 cd   20 40
basicsys@mars~/lec11>echo ${a[@]}
ab 10 cd 20 40
basicsys@mars~/lec11>a[1]="40 1111"
basicsys@mars~/lec11>echo "${a[@]}"
ab  10 40 1111 40
```

${#שם מערך[@]}

מוחלף במספר האיברים במערך.

```
basicsys@mars~/lec11>b=({2..5}{a..c})
basicsys@mars~/lec11>echo ${b[@]}
2a 2b 2c 3a 3b 3c 4a 4b 4c 5a 5b 5c
basicsys@mars~/lec11>echo "${#b[@]}"
12
```

```
basicsys@mars~/lec11>a=("ab  10" "cd   20" 40)
basicsys@mars~/lec11>echo "${a[@]}"
ab  10 cd   20 40
basicsys@mars~/lec11>echo "${#a[@]}"
3
basicsys@mars~/lec11>echo ${#a[@]}
3
```

${ שם מערך [@] : מספר 1 : מספר 2 }

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

```
basicsys@mars~/lec11>b=({2..5}{a..c})
basicsys@mars~/lec11>echo ${b[@]}
2a 2b 2c 3a 3b 3c 4a 4b 4c 5a 5b 5c
basicsys@mars~/lec11>echo "${b[@]:2:3}"
2c 3a 3b
basicsys@mars~/lec11>b=({a..c}{1..2}{d..f})
basicsys@mars~/lec11>echo "${b[@]}"
a1d a1e a1f a2d a2e a2f b1d b1e b1f b2d b2e b2f
c1d c1e c1f c2d c2e c2f
basicsys@mars~/lec11>echo ${#b[@]}
18
```

${ שם מערך }

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

${#שם מערך}

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

```
basicsys@mars~/lec11>b=({a..c}{1..2}{d..f})

basicsys@mars~/lec11>echo "${b[@]}"

a1d a1e a1f a2d a2e a2f b1d b1e b1f b2d b2e b2f
c1d c1e c1f c2d c2e c2f

basicsys@mars~/lec11>echo ${b}

a1d

basicsys@mars~/lec11>echo ${#b}

3
```

${#שם מערך[@]/ביטוי בסגנון גלוב}

מוחלף ברשימת אברי המערך לאחר ביצוע קיצוץ משמאל בכל אחד מאברי המערך של החלק (הארוך ביותר) שמתאים לביטוי בסגנון גלוב.

${שם מערך[@]/%ביטוי בסגנון גלוב}

מוחלף ברשימת אברי המערך לאחר ביצוע קיצוץ מימין בכל אחד מאברי המערך של החלק (הארוך ביותר) שמתאים לביטוי בסגנון גלוב.

```
basicsys@mars~/lec11>b=({a..c}{1..2}{d..f})

basicsys@mars~/lec11>echo "${b[@]}"

a1d a1e a1f a2d a2e a2f b1d b1e b1f b2d b2e b2f
c1d c1e c1f c2d c2e c2f

basicsys@mars~/lec11>echo ${b[@]/#?}

1d 1e 1f 2d 2e 2f 1d 1e 1f 2d 2e 2f 1d 1e 1f 2d
2e 2f
```

```
basicsys@mars~/lec11>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 פעולות נוספות על מחרוזות על כל אחד מאברי המערך כפי שמוצג בדוגמאות הבאות.

```
basicsys@mars~/lec11>b=({a..c}1{a..c})

basicsys@mars~/lec11>echo ${b[@]}

a1a a1b a1c b1a b1b b1c c1a c1b c1c

basicsys@mars~/lec11>echo ${b[@]/a/zzz}

zzz1a zzz1b zzz1c b1zzz b1b b1c c1zzz c1b c1c

basicsys@mars~/lec11>echo ${b[@]//a/zzz}

zzz1zzz zzz1b zzz1c b1zzz b1b b1c c1zzz c1b c1c

basicsys@mars~/lec11>b=({a..c}%{a..c})

basicsys@mars~/lec11>echo ${b[@]}

a%a a%b a%c b%a b%b b%c c%a c%b c%c
```

הפעולות %% ו- ## של מחרוזות, לא קימות על אברי מערכים ולכן הפקודה הבאה מפרשת את ה- % השני כתו %.

```
basicsys@mars~/lec11>echo ${b[@]/%%?}

a a a b b b c c c
```

מיון לפי שדות ב- bash

האופציה -k מספר1 ,מספר2 של sort גורמת לכך שהמיון יתבצע לפי התחום שמוגדר על ידי זוג המספרים: משדה מספר1 ועד שדה מספר2.

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec6>cat F1

Jim Alchin 21 Seattle
Bill Gates 8  Seattle
Steve Jobs 246 Nevada
Scott Neally 2122 Los Angeles
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec6>sort F1
```

מתקבל הפלט:

```
Bill Gates 8  Seattle
Jim Alchin 21 Seattle
Scott Neally 2122 Los Angeles
Steve Jobs 246 Nevada
```

לאחר הפעלת הפקודה:

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

מתקבל הפלט:

```
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 בתחום יהיה בסדר מספרי.

לאחר הפעלת הפקודה:

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

מתקבל הפלט:

```
Bill Gates 8  Seattle
Jim Alchin 21 Seattle
Steve Jobs 246 Nevada
Scott Neally 2122 Los Angeles
```

הפלט של שלושת הפקודות הבאות זהה לפלט של הפקודה הנ"ל. לאחר הפעלת הפקודה:

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

מתקבל הפלט:

```
Bill Gates 8  Seattle
Jim Alchin 21 Seattle
Steve Jobs 246 Nevada
Scott Neally 2122 Los Angeles
```

לאחר הפעלת הפקודה:

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

מתקבל הפלט:

```
Bill Gates 8  Seattle
Jim Alchin 21 Seattle
Steve Jobs 246 Nevada
Scott Neally 2122 Los Angeles
```

לאחר הפעלת הפקודה:

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

מתקבל הפלט:

```
Bill Gates 8  Seattle
Jim Alchin 21 Seattle
Steve Jobs 246 Nevada
Scott Neally 2122 Los Angeles
```

ניתן להוסיף את האות r למספר 1 או למספר 2 (או לשניהם) והמשמעות היא שהמיון בתחום יהיה בסדר הפוך. לאחר הפעלת הפקודה:

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

מתקבל הפלט:

```
Scott Neally 2122 Los Angeles
Steve Jobs 246 Nevada
Jim Alchin 21 Seattle
Bill Gates 8  Seattle
```

ניתן להגדיר את התחום על ידי מספר זוגות כמו בדוגמה הבאה. במקרה זה המיון מתחיל לפי התחום שמגדיר זוג המספרים הראשון, עבור השורות ששוות בתחום זה, המיון ממשיך לפי התחום שמגדיר הזוג השני וכן הלאה...

נניח שתוכן הקובץ F2 הוא:

```
basicsys@mars~/lec6>cat F2

Jim Alchin 21 Seattle aef
Bill Gates 21 Seattle aef
Steve Jobs 21 Nevada aaa
Scott Neally 85 Los Angeles
```

לאחר הפעלת הפקודה:

```
basicsys@m~/lec6>sort -k 3n,3n -k 4,4 -k 1,1 F2
```

מתקבל הפלט:

```
Steve Jobs 21 Nevada aaa
Bill Gates 21 Seattle aef
Jim Alchin 21 Seattle aef
Scott Neally 85 Los Angeles
```

הפעלת סקריפט ב- bash בצורה: שם הסקריפט

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.

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec11>cat P1

x=9
```

לאחר הפעלת שלושת הפקודות הבאות:

```
basicsys@mars~/lec11>x=100

basicsys@mars~/lec11>P1

basicsys@mars~/lec11>echo $x
```

מתקבל הפלט:

```
100
```

לאחר הפעלת שתי הפקודות הבאות:

```
basicsys@mars~/lec11>y=$(P1)

basicsys@mars~/lec11>echo $x
```

מתקבל הפלט:

```
100
```

הפעלת סקריפט ב- bash בצורה: שם הסקריפט . (נקודה)

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.

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec11>cat P1

x=9
```

לאחר הפעלת שתי הפקודות הבאות:

```
basicsys@mars~/lec11>. P1

basicsys@mars~/lec11>echo $x
```

מתקבל הפלט:

```
9
```

שימוש ב- 'מחרוזת2' = מחרוזת1 alias ב- bash

הפקודה: 'מחרוזת2' = מחרוזת1 alias

ב- bash גורמת לכך שבכל פעם שנרשום מחרוזת1 היא תוחלף במחרוזת2 לפני ש- bash יתחיל לבצע אותה. ניתן להשתמש בפקודה זו כדי ליצור קיצורים לפקודות נפוצות.

```
basicsys@mars~/lec11>alias m='more'

basicsys@mars~/lec11>m F1

xyz
abc
def
gh
```

inodes ב- bash

לכל קובץ ב- unix יש מבנה שנקרא inode שמכיל אינפורמציה על הקובץ שכוללת בין השאר את גודל הקובץ, ההרשאות של הקובץ, זמן יצירת הקובץ וכו'. למבנה הזה יש מצביע, שנקרא inode number שמשמש את המערכת כדי לזהות את הקובץ ולהגיע ישירות ל- inode שלו.

האופציה i- של הפקודה ls מאפשרת לראות בנוסף לשם הקובץ את ה- inode number שלו.

```
basicsys@mars~/lec11>ls -l F1

-rw------- 1 basicsys basicsys 24 Jan 14 17:51 F1

basicsys@mars~/lec11>ls -i F1

664961 F1
```

ln 1 שם קובץ 2 שם קובץ

הפקודה: ln 1 שם קובץ 2 שם קובץ

מוסיפה את שם קובץ 2 כשם נוסף לקובץ 1. לשני השמות יש אותו inode number ולכן שניהם מתייחסים לאותו קובץ. כל שינוי שנעשה בקובץ כאשר פונים אליו באחד משני השמות, מופיע באופן מידי כאשר פונים לקובץ בשמו השני. שני השמות נקראים hard links לקובץ שאליו הם מתייחסים.

לאחר הפעלת שתי הפקודות:

```
basicsys@mars~/lec11>ln F1 F2
basicsys@mars~/lec11>ls -li F*
```

מתקבל הפלט:

```
664961 -rw------- 2 basicsys 24 Jan 14 17:51 F1
664961 -rw------- 2 basicsys 24 Jan 14 17:51 F2
```

לאחר הפעלת שתי הפקודות:

```
basicsys@mars~/lec11>chmod o+r F1
basicsys@mars~/lec11>ls -li F*
```

מתקבל הפלט:

```
664961 -rw----r-- 2 basicsys 24 Jan 14 17:51 F1
664961 -rw----r-- 2 basicsys 24 Jan 14 17:51 F2
```

לאחר הפעלת שתי הפקודות:

```
basicsys@mars~/lec11>chmod o+x F2
basicsys@mars~/lec11>ls -li F*
```

מתקבל הפלט:

```
664961 -rw----r-x 2 basicsys 24 Jan 14 17:51 F1
664961 -rw----r-x 2 basicsys 24 Jan 14 17:51 F2
```

ננית שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec11>cat F1

hello hi
aaaaaaa bbbbbb
```

ננית שתוכן הקובץ F2 הוא:

```
basicsys@mars~/lec11>cat F2

hello hi
aaaaaaa bbbbbb
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec11>echo yyyy >> F2
```

תוכן הקובץ F1 הוא:

```
basicsys@mars~/lec11>cat F1

hello hi
aaaaaaa bbbbbb
yyyy
```

תוכן הקובץ F2 הוא:

```
basicsys@mars~/lec11>cat F2

hello hi
aaaaaaa bbbbbb
yyyy
```

שני השמות F1 ו- F2 מתייחסים לאותו קובץ פיזי שה-inode number שלו הוא 664961. לאחר הפעלת הפקודה הבאה שמוחקת את הקובץ F1:

```
basicsys@mars~/lec11>rm F1
```

הקובץ שה- inode number שלו הוא 664961 עדין קיים ויש לו רק שם אחד עכשיו שהוא F2 כפי שמראה הפקודה הבאה:

```
basicsys@mars~/lec11>ls -li F*

664961 -rw----r-x 1 basicsys 29 Jan 14 17:58 F2
```

לאחר הפעלת הפקודה הבאה שמוחקת את הקובץ F2:

```
basicsys@mars~/lec11>rm F2
```

לא קיים יותר קובץ שה inode number שלו הוא 664961
כי לא נשארו שמות שמשויכים ל inode number הזה.
כאשר נבצע את הפקודה הבאה נקבל שאין יותר קבצים
ששמם מתחיל ב- F.

```
basicsys@mars~/lec11>ls -li F*
```

```
ls: F*: No such file or directory
```

מחרוזת1 שם קובץ1 ln -s

הפקודה: מחרוזת1 שם קובץ1 ln -s

יוצרת קישור לקובץ1 שמתואר על ידי מחרוזת1. הקישור
הזה נקרא symbolic link ויש לו inode number שונה
מזה של קובץ1. האינפורמציה שהקישור הזה מכיל היא
מסלול שממנו אפשר להגיע לקובץ1. אם בעתיד נמחק את
קובץ1 אז שימוש בקישור הזה שלו יגרום לשגיאה.
משתמשים ב- symbolic links כדי לתת קישור לתיקיות
או לקבצים שנמצאים במערכת אחרת. (אין אפשרות לתת
hard links לתיקיות או לקבצים במערכת אחרת).

נניח שתוכן הקובץ F1 הוא:

```
basicsys@mars~/lec11>cat F1
```

```
hello hi
abcdef
```

לאחר הפעלת 3 הפקודות הבאות:

```
basicsys@mars~/lec11>ln F1 F2
```

```
basicsys@mars~/lec11>ln -s F1 F3
```

```
basicsys@mars~/lec11>ls -li F*
```

מתקבל הפלט:

```
664961 -rw------- 2 basicsys 16 Jan 14 18:08 F1
664961 -rw------- 2 basicsys 16 Jan 14 18:08 F2
664965 lrwxrwxrwx 1 basicsys  2 Jan 14 18:F3->F1
```

לאחר הפעלת 2 הפקודות הבאות:

```
basicsys@mars~/lec11>chmod o+r F3

basicsys@mars~/lec11>ls -li F*
```

מתקבל הפלט:

```
664961 -rw----r-- 2 basicsys 16 Jan 14 18:08 F1
664961 -rw----r-- 2 basicsys 16 Jan 14 18:08 F2
664965 lrwxrwxrwx 1 basicsys  2 Jan 14 18:F3->F1
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec11>echo yyyy >> F3
```

תוכן הקובץ F1 הוא:

```
basicsys@mars~/lec11>cat F1

hello hi
abcdef
yyyy
```

תוכן הקובץ F2 הוא:

```
basicsys@mars~/lec11>cat F2

hello hi
abcdef
yyyy
```

תוכן הקובץ F3 הוא:

```
basicsys@mars~/lec11>cat F3

hello hi
abcdef
yyyy
```

לאחר הפעלת 2 הפקודות הבאות:

```
basicsys@mars~/lec11>rm F1

basicsys@mars~/lec11>ls -li F*
```

מתקבל הפלט:

```
664961 -rw----r-- 1 basicsys 21 Jan 14 18:10 F2
664965 lrwxrwxrwx 1 basicsys  2 Jan 14 18:F3->F1
```

מאחר F3 הוא symbolic link שמפנה לקובץ F1, והקובץ F1 נמחק, כאשר נרצה לראות את תוכן F3 נקבל שגיאה. לכן, לאחר הפעלת הפקודה הבאה:

```
basicsys@mars~/lec11>cat F3
```

מתקבל הפלט:

```
cat: F3: No such file or directory
```

נמחק את F3 על ידי הפקודה הבאה:

```
basicsys@mars~/lec11>rm F3
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec11>ls -li F*
```

מתקבל הפלט:

```
664961 -rw----r-- 1 basicsys 21 Jan 14 18:10 F2
```

בקרת תהליכים ב- bash

הפעלת תכנית ברקע

ניתן להפעיל תכנית ברקע על ידי הצורה:

```
& שם התכנית
```

לדוגמה, התכנית הבאה היא לולאה אינסופית. לאחר הפעלתה יהיה עלינו ללחוץ על ctrl+c כדי להפסיקה.

```
basicsys@mars~/lec11>cat P1
```

```
i=1
while [ $i -le 2 ]
 do
  y=2
 done
```

נוכל להפעיל את התכנית ברקע על ידי הפקודה הבאה:

```
basicsys@mars~/lec11>P1 &
```

```
[1] 21100
```

במצב זה נוכל להמשיך להפעיל פקודות נוספות תוך כדי כך שהתכנית P1 ממשיכה להתבצע ברקע.

```
basicsys@mars~/lec11>ls
```

```
examples  F2  lec11e.txt  lec11e.txt.save  P1
```

הפקודה jobs

הפקודה jobs מציגה את התכניות שמתבצעות כרגע.

```
basicsys@mars~/lec11>jobs
```

```
[1]+  Running                 P1 &
```

הפקודה ps

הפקודה ps מציגה את התהליכים שמתבצעים כרגע. ההבדל בין תהליך לבין תכנית הוא שתכנית אחת יכולה להפעיל כמה תהליכים. בדוגמה שלהלן אנו רואים שלושה תהליכים. אחד הוא ה- bash , השני הוא התכנית P1 שרצה ברקע ומפעילה כרגע רק תהליך אחד והשלישי הוא התהליך שמבצע את הפקודה ps.

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec11>ps
```

מתקבל הפלט:

```
  PID TTY          TIME CMD
20965 pts/22   00:00:00 bash
21348 pts/22   00:00:13 P1
21376 pts/22   00:00:00 ps
```

ניצור עותק נוסף של התכנית P1 בשם P2 על ידי הפקודה הבאה:

```
basicsys@mars~/lec11>cp P1 P2
```

נפעיל את התכניות P1 ו- P2 ברקע על יד שתי הפקודות הבאות:

```
basicsys@mars~/lec11>P1 &
[1] 21389
basicsys@mars~/lec11>P2 &
[2] 21417
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec11>jobs
```

מתקבל הפלט:

```
[1]-  Running                 P1 &
[2]+  Running                 P2 &
```

כפי שניתן לראות בפלט של הפקודה jobs הנ"ל, כשיש כמה תכניות שמתבצעות במקביל ה- + מציין את התכנית האחרונה.

kill -9 %

הפקודה % **kill -9 מוחקת** את התכנית האחרונה מבין התכניות שמתבצעות כרגע.

מספר תהליך kill -9

הפקודה מספר תהליך 9- kill מוחקת את התהליך שמספרו מועבר כפרמטר לפקודה.

נניח שבמערכת רצות כרגע שתי תכניות כפי שמראה הפקודה הבאה:

```
basicsys@mars~/lec11>jobs

[1]-  Running                 P1 &
[2]+  Running                 P2 &
```

נמחק את התכנית האחרונה על ידי הפקודה הבאה:

```
basicsys@mars~/lec11>kill -9 %
```

עכשיו לאחר ביצוע הפקודה הבאה:

```
basicsys@mars~/lec11>jobs
```

מתקבל הפלט:

```
[1]-  Running                 P1 &
[2]+  Killed                  P2
```

השורה האחרונה בפלט הנ"ל זמנית ונמחקת לאחר זמן קצר, ולכן כשנבצע שוב את הפקודה הבאה:

```
basicsys@mars~/lec11>jobs
```

יתקבל הפלט:

```
[1]+  Running                 P1 &
```

נמחק שוב את התכנית האחרונה על ידי הפקודה הבאה:

```
basicsys@mars~/lec11>kill -9 %
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec11>jobs
```

יתקבל הפלט:

```
[1]+  Killed                  P1
```

נניח שתוכן התכנית P1 הוא:

```
basicsys@mars~/lec11>cat P1

i=1
while [ $i -le 2 ]
 do
  echo abc | P2
 done
```

התכנית הנ"ל מפעילה את התכנית P2 שתוכנה הוא:

```
basicsys@mars~/lec11>cat P2

i=1
while [ $i -le 2 ]
 do
  y=2
 done
```

נפעיל את התכנית P1 ברקע על ידי הפקודה הבאה:

```
basicsys@mars~/lec11>P1 &

[1] 25997
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec11>jobs
```

מתקבל הפלט:

```
[1]+  Running                 P1 &
```

לפיכך, יש לנו רק תכנית אחת שמתבצעת.

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec11>ps
```

מתקבל הפלט:

```
  PID TTY          TIME CMD
25660 pts/2    00:00:00 bash
25997 pts/2    00:00:00 P1
26026 pts/2    00:00:00 P2
26054 pts/2    00:00:00 ps
```

לפיכך, יש לנו 4 תהליכים שמתבצעים.

נמחק את התהליך שמבצע את התכנית P2 על ידי הפקודה הבאה:

```
basicsys@mars~/lec11>kill -9 26026
```

לאחר המחיקה נקבל את ההודעה הבאה:

```
basicsys@mars~/lec11>./P1: line 6: 26025 Done
     26026 Killed                  | P2
```

לאחר המחיקה התכנית P1 חוזרת לתחילת הלולאה שלה ושוב מפעילה את התכנית P2 אבל על ידי תהליך אחר.

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec11>ps
```

מתקבל הפלט:

```
  PID TTY          TIME CMD
25660 pts/2    00:00:00 bash
25997 pts/2    00:00:00 P1
26056 pts/2    00:00:02 P2
26084 pts/2    00:00:00 ps
```

ניתן לראות שמספר התהליך 26056 שמפעיל את התכנית P2 שונה ממספר התהליך 26026 שהפעיל את התכנית P2

הקודמת. נמחק את התהליך שמפעיל את התכנית P1 על ידי הפקודה:

```
basicsys@mars~/lec11>kill -9 25997
```

לאחר הפעלת הפקודה:

```
basicsys@mars~/lec11>ps
```

מתקבל הפלט:

```
  PID TTY          TIME CMD
25660 pts/2    00:00:00 bash
26056 pts/2    00:01:19 P2
26090 pts/2    00:00:00 ps
[1]+  Killed                  P1
```

הרצאת תגבור 1

דוגמה 1

כתוב/כתבי תוכנית סקריפט ב- bash בשם P1 שמקבלת כפרמטרים מספר (בהמשך נקרא לו i) ולאחריו רשימת מילים (בהמשך נקרא לה רשימה1) ומדפיסה לפלט את כל המילים ברשימה1 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
```

דוגמה 2

כתוב/כתבי תוכנית סקריפט ב- bash בשם P2 שמקבלת
כפרמטרים רשימת מילים (בהמשך נקרא לה רשימה 1)
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
```

דוגמה 3

כתוב/כתבי תוכנית סקריפט ב- bash בשם P3 שמקבלת כפרמטרים מחרוזת (בהמשך נקרא לה מחרוזת1) ותיקיה (בהמשך נקרא לה תיקיה1) ומדפיסה לפלט את כל שמות הקבצים (הסופיים, דהינו בלי המסלול אליהם) שנמצאים בתיקיה1 (בעומק כלשהו) ואינם מכילים את מחרוזת1. 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. (יתכן שקיים קובץ נוסף כזה, אבל השם A יופיע רק פעם אחת בפלט). וכן הלאה לגבי השמות B,D,E,G,S 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

for f in $(cat tmp1)
 do
  echo $f | tr "/" "\n" | tail -1 >> tmp2
 done
sort -u tmp2
```

דוגמה 4

כתוב/כתבי תוכנית סקריפט ב- bash בשם P4 שמקבלת כפרמטרים מחרוזת (בהמשך נקרא לה מחרוזת1) ושם קובץ (בהמשך נקרא לו קובץ1) ומדפיסה לפלט את מספר הפעמים שהמחרוזת מופיעה בקובץ כמילה.

לדוגמה, נניח שתוכן הקובץ F1 הוא:

```
ab1 ab@ ab
ab  cdab ab
a   ab
```

לאחר הפעלת התוכנית על ידי הפקודה:

```
P4 ab F1
```

יתקבל הפלט:

```
4
```

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
```

דוגמה 5

כתוב/כתבי תוכנית סקריפט ב- bash בשם P5 שמקבלת כפרמטרים מחרוזת (בהמשך נקרא לה מחרוזת1), מספר (בהמשך נקרא לו i) ושם תיקיה (בהמשך נקרא לה תיקיה 1) ומדפיסה לפלט את מספר הקבצים בתיקיה1 (בעומק כלשהו) שהמחרוזת מופיעה בהם כמילה בדיוק i פעמים.

לדוגמה, לאחר הפעלת התוכנית על ידי הפקודה:

```
P5 ab 1 ~basicsys/win14/ex7
```

יתקבל הפלט:

```
2
```

הסבר לפלט: המספר 2 מציין שבתיקיה ~basicsys/win14/ex7 קיימים 2 קבצים שהמילה ab מופיעה בהם בדיוק פעם אחת.

לאחר הפעלת התוכנית על ידי הפקודה:

```
P5 ab 2 ~basicsys/win14/ex7
```

יתקבל הפלט:

```
0
```

הסבר לפלט: המספר 0 מציין שבתיקיה ~basicsys/win14/ex7 לא קיימים קבצים שהמילה ab מופיעה בהם בדיוק פעמיים.

פתרון:

```
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
```

דוגמה 6

כתוב/כתבי תוכנית סקריפט ב- bash בשם P6 שמקבלת כפרמטרים רשימת מחרוזות (בהמשך נקרא לה רשימה1), לאחריה המחרוזת -nums לאחריה רשימת מספרים (בהמשך נקרא לה רשימה2) לאחריה המחרוזת -dirs ולאחריה רשימת תיקיות (בהמשך נקרא לה רשימה3) ומדפיסה לפלט שורה אחת עבור כל צרוף של מחרוזת מרשימה1, מספר מרשימה2 ותיקיה מרשימה3 שמכילה את המחרוזת, המספר והתיקייה (כשבינם מפריד תו רווח אחד בדיוק), לאחר מכן תו רווח, ולאחריו מספר שמציין את מספר הקבצים בתיקיה שמספר ההופעות בהם של המחרוזת כמילה שווה למספר.

לדוגמה, לאחר הפעלת התוכנית על ידי הפקודה:

```
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

/home/cs/segel/basicsys/win14/ex7

שהמחרוזת ab מופיעה בהם כמילה בדיוק פעם אחת.

פתרון:

```
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
```

הרצאת תגבור 2

דוגמה 1

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) ומדפיסה לפלט את סכום המספרים בשורה ה- i של המטריצה.

לדוגמה, נניח שתוכן הקובץ F1 הוא:

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

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

```
15
```

פתרון:

```
#!/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
}
```

דוגמה 2

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:

```
18 40
22 26
7 8
```

לאחר הפעלת התוכנית על ידי הפקודה: P2 2 F1 F3 F2
יתקבל הפלט:

```
F1: 15
F3: 48
F2: 105
```

פתרון:

```
#!/bin/awk -f
BEGIN { row=ARGV[1]; 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
 } }
```

דוגמה 3

כתוב/כתבי תוכנית ב- awk בשם P3 שמקבלת פרמטרים כמו בדוגמה 2 ומדפיסה לפלט שורה אחת עבור כל קובץ כמו דוגמה 2, אבל על סדר השורות בפלט להיות בסדר לכסיקוגרפי עולה (בהתאם לפקודה sort ללא אופציות). מותר להשתמש לכל היותר ב- 4 פקודות system לפתרון השאלה.

לדוגמה, לאחר הפעלת התוכנית על ידי הפקודה: P3 2 F1 F3 F2 יתקבל הפלט:

```
F1: 15
F2: 105
F3: 48
```

פתרון:

```
#!/bin/awk -f
BEGIN { row=ARGV[1]; ARGV[1]="" }
{
 for (i=1; i<= NF; i++) {
 A[FILENAME,FNR,i]=$i
 }
 B[FILENAME,FNR]=NF
}
END {
system("echo -n "" >| tmp")
for (x=2; x < length(ARGV); x++) {
  system("echo " ARGV[x] ">>tmp")
  }
system("sort tmp >|tmp1")
while (getline file<"tmp1"){
 s=0
 for (j=1; j<= B[file,row]; j++) {
   s=s+A[file,row,j]
   }
 print file ": " s
 }
}
```

דוגמה 4

כתוב/כתבי תוכנית ב- **awk** בשם **P4** שמקבלת כפרמטרים רשימת מספרים (בהמשך נקרא לה רשימה1) ולאחריה רשימת שמות קבצים שמכילים מטריצות ריבועיות (בהמשך נקרא לה רשימה2) ומדפיסה לפלט שורה אחת עבור כל קובץ שמכילה את שם הקובץ לאחר מכן תו רווח אחד ולאחר מכן מספר שמציין את סכום העמודות של הקובץ עבור העמודות שמופיעות ברשימת המספרים (אם מופיעה ברשימה1 עמודה שלא נמצאת בקובץ אז העמודה הזאת לא נלקחת בחשבון 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 ברשימה1.

לדוגמה, נניח שתוכן הקובץ **F4** הוא:

```
10 20 30 40
1 2 3 4
-1 -2 -3 -4
1 2 3 4
```

נניח שתוכן הקובץ **G** הוא:

```
100
```

נניח שתוכן הקובץ **H** הוא:

```
6 2 3
8 4 1
1 2 5
```

נניח שתוכן הקובץ **F5** הוא:

```
6 2
8 4
```

לאחר הפעלת התוכנית על ידי הפקודה:

```
P4 1 3 10 F4 G H F5
```

יתקבל הפלט:

```
F4 44
G 100
H 24
F5 14
```

<u>פתרון</u>:

```
#!/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
}
```

דוגמה 5

כתוב תוכנית Script - ב sed בשם P5 שמקבלת כפרמטר שם קובץ (בהמשך נקרא לו קובץ 1) התוכנית מדפיסה לפלט את השורות בקובץ 1 שמכילות לפחות שתי ספרות, כאשר בשורות האלה מתבצעת החלפת בין המילה הראשונה למילה האחרונה.
לדוגמה, נניח שתוכן הקובץ F6 הוא:

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

לדוגמה, לאחר הפעלת התוכנית על ידי הפקודה: P5 F6
יתקבל הפלט:

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

פתרון:

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