#!/bin/bash

1 שימוש ב-Bash

1.1 רשימת פקודות

פקודה	משמעות ודגלים	תחביר - Syntax
ls	**הדפסת** שמות (ופרטי) הקבצים בתיקייה הנוכחית או בתיקייה שהתקבלה בארגומנט - l- מדפיס פרטים נוספים על הקובץ - a- מדפיס את הקבצים בתיקייה המצוינת כולל קבצים המתחילים ב "."	ls [-l] [-a] [dir]
cat	**הדפסת** תוכן קבצים למסך.	cat <files>
more	**הדפסת** תוכן קבצים למסך בצורה מאורגנת (מציגה מסך שלם ומחכה לקלט מהמשתמש לפני הצגת מסך נוסף).	more <files>
less	**הדפסה ועיבוד** תוכן קובץ בממשק טקסטואלי.	less <file>
mv	הזזת או שינוי שם קובץ. ניתן להזיז כמה קבצים לתוך תיקייה.	mv filename1 filename2 mv <files> dirname
cp	העתקת קובץ.	cp filename1 filename2
rm	**מחיקת** קבצים או תיקיות. - r– מוחק את כל הקבצים והתיקיות רקורסיבית.	rm [-r] <files>
diff	הדפסת הבדלים בין קבצים	diff filename1 filename2
chmod	שינוי הרשאות קבצים/תיקיות g=group , u=user - o+r נותן הרשאות קריאה לקבוצה others - o-r מבטל הרשאות קריאה לקבוצה others X= רק לתיקיה	chmod <options> <files>
mkdir	**יצירת** תיקייה חדשה בתיקייה הנוכחית	mkdir <dirname>
cd	**שינוי** התיקייה הנוכחית	cd <dirname>
pwd	**הדפסת** התיקייה הנוכחית	pwd
make	**הרצת** התוכנה make המחפשת קובץ makefile ומנסה לבנות בו את ה target הראשון. - ניתן לציין target	make [target]
alias	**הגדרת קיצורים** ב-shell	alias <name>="<command>"
unalias	**ביטול קיצור**	unalias <alias>
head	הדפסת **שורות ראשונות** מהקלט. - אם לא מוגדר מספר יודפסו 10 שורות - אם המספר שהוגדר שלילי יודפסו כל השורות מלבד # השורות האחרונות - עבור מספר חיובי ניתן לרשום גם #-	head [-n#] [files]
tail	הדפסת **השורות האחרונות** מהקלט - עבור #- מדפיסה את # השורות האחרונות בקלט - עבור #+ מדפיסה החל מהשורה ה-# - ברירת המחדל היא הדפסת 10 השורות האחרונות	tail [-/+#] [files]
	דוגמה לשימוש ב-pipeline עבור הדפסת שורות 2-3 מקובץ מסוים	head -3 a_file | tail -2

```
sortedColNums=($(for i in "${colNums[@]}"; do echo $i; done | sort -n))
```

sort	הדפסת הקלט ממוין (לפי שורות) - n- ממיינת מספרים לפי ערכם - r- מדפיס בסדר יורד (ברירת המחדל היא סדר עולה) - #k- מתייחס לכל שורה החל מהמילה ה-# (המילים ממסופרות החל מ-1) - f- מתעלם מהבדלי uppercase/lowercase - b- מתעלם מרווחים בתחילת השורה - s- מיון יציב - שומר על הסדר המקורי בין שורות שערכן שווה	sort [options] [files] >sort data data: World who for Father Father World for who
echo	הדפסת מחרוזת למסך n- מדפיס ללא ירידת שורה	echo [-n] [string]
uniq	הדפסת עותק יחיד של שורות זהות סמוכות - c- מדפיס כל שורה פעם אחת ואת מספר העותקים שלה - d- מדפיס רק שורות המופיעות יותר מפעם אחת - u- מדפיס רק שורות המופיעות פעם אחת בלבד - ניתן להשתמש רק באחת מהאפשרויות c-/d-/u- בו זמנית - #- התעלם מ-# המילים הראשונות בקבלת ההחלטה האם שורות זהות)	uniq [options] [input [output]] >uniq file file: one two three one one two two three three three one one
grep	**חיפוש מילים מסוימות או ביטויים מורכבים** **מדפיס את כל השורות בקובץ המכילות את הביטוי שהוגדר** - v- מדפיס את השורות בהן לא מופיע הביטוי - i- מתעלם מהבדלי uppercase/lowercase - w- מדפיסה את כל השורות בהן <expression> מופיע בדיוק (לא כתת מחרוזת) - n- הדפס את השורות ואת מספרן בקבצים - l- הדפס רק את שמות הקבצים בהן נמצאו שורות מתאימות - c- הדפס רק את כמות השורות שנמצאו בכל קובץ ללא הדפסת השורות עצמן הביטוי לחיפוש יכול להיות מורכב יותר: - כדי לחפש מחרוזת עם רווחים יש להוסיף גרשיים - התו ^ מייצג תחילת שורה והתו $ סוף שורה	grep [options] <expression> [files] >grep -i cow farm1 cow Betsy slim cow Dazy Fat Cow Burger two cows Dartsy & Teo
cut	**הפרדת עמודות מתוך הקלט** - c<list>- הדפסת התווים בשורה המתאימים לאינדקסים - f<list>- הדפסת השדות בשורה המתאימים לאינדקסים - השדות בשורה מופרדים כברירת מחדל ע"י Tab - אם התו המפריד אינו קיים בשורה תודפס כל השורה - d-"?" השתמש בתו שהוגדר כתו המפריד (עבור שימוש בדגל f-) - רשימת האינדקסים מורכבת ממספר אינדקסים מופרדים בפסיק, כאשר בנוסף ניתן לבקש טווח אינדקסים ע"י שימוש בתו - (מקף) לדוגמה, -c1,2,5-6,10 ידפיס את התווים באינדקסים 1, 2, 5, 6, ו-10 ומעלה - אם האינדקסים המבוקשים מחוץ לתחום תודפס שורה ריקה	cut [options] [files] > cut -d":" -f1 file2 a11 b21 c31 c32 c33 file2: a11:a12:a13 b21:a15 c31 c32 c33 > cut -c1-3,5,8-10 file1 a11a a1 b21b b2 c31c c3 file1: a11 a12 a13 a14 a15 b21 b22 b23 b24 b25 c31 c32 c33
wc	**ספירת תווים, מילים או שורת בקלט** - c- מדפיס את מספר התווים בלבד - l- מדפיס את מספר השורות בלבד - w- מדפיס את מספר המילים בלבד - אם לא צוין דגל מסוים wc מדפיסה את כל שלושת המספרים - אם wc מופעלת על מספר קבצים מודפסת גם שורת סיכום	wc [options] [files] >wc mtm_ex1.h example.c 120 641 4161 mtm_ex2.h 136 692 4543 example.c 256 1333 8704 total

tee	**שכפול הפלט** משכפלת את הקלט הסטנדרטי ומדפיסה אותו לפלט הסטנדרטי וכן לכל אחד מהקבצים ברשימה [files] - a- משרשרת לקבצים במקום לכתוב אותם מחדש מבנה הפקודה אינו סטנדרטי, היא מקבלת קלט רק מהקלט הסטנדרטי	tee [options] [files]
printf	**הדפסה מעוצבת של טקסט** - %s במחרוזת הפורמט מציין שיש להכניס כאן את ערך הארגומנט הבא - %[-]m[.n]s מדפיס את המחרוזת המתאימה מרשימת המחרוזות כך שהתוצאה תהיה באורך m תווים ותכיל לכל היותר n תווים מהמחרוזת. המחרוזת תהיה מיושרת לימין אלא אם מופיע - (מקף). - אם יש יותר ארגומנטים מהדרוש ברשימה, הפונקציה תדפיס את השורה מחדש ותשתמש בארגומנטים הבאים בכל פעם עד שייגמרו.	printf <format> [arguments] >printf "%s %s!\n" Hello world Hello world!

1.2 סינטקס בסיסי

1.2.1 תבניות

- * מתאים למחרוזת כלשהי (כולל ריקה)
- ? מתאים לתו כלשהו יחיד
- [ ] מתייחס למספר תווים אפשריים.
  - ✓ **התווים יכולים להינתן:** כטווח של תווים, למשל a-z, או אחד אחרי השני במפורש.
- { } מתייחס למספר מחרוזות שונות. אפשרות זו אינה מתחשבת בקבצים קיימים:

```
> echo a3b.{c,txt}
a3b.c a3b.txt
```

1.2.2 משתנים

- $ לקריאת ערך ממשתנה.
- {} לסימון שם המשתנה לאופרטור $
- **unset <varname>** לביטול משתנה.
- מספר התווים במשתנה: שימוש ב-#

```
> a=Hell
> echo ${a}o
Hello

> echo ${#str}
```

1.2.3 מערכים

- הגדרה על ידי שימוש בסוגריים: arr=(1 2 3)
- קריאת האיבר הראשון - ${<varname>} = קריאת שם המשתנה של המערך. (שקולה לקריאת (אינדקס 0) מהרשימה).

```
> echo $arr
1
```

- קריאת כל המערך - ${<varname>[*]}

```
echo ${arr[*]}
1 2 3
```

- גישה לאיבר והוספת איבר חדש - אופרטור [ ]

```
arr[3] = 4
```

- מספר האיברים במערך - ${#<varname>[*]}

```
>${#arr[*]}
4
```

- גישה לתחומים של איברים במערך - ${${<varname>[*]:<num1>:<num2>}
  - num1- אינדקס ההתחלה של התחום, num2- מספר האיברים בתחום (אם הוא אינו מופיע יודפסו כל האיברים החל מהאינדקס המבוקש)

```
> echo A:${arr[*]:0:2}
A:1 2
```

1.2.4 סוגי גרשיים

- גרשיים כפולים " "

תפקיד: משמשים לשמירה על רווחים.

  - משתמשים ב-" " כאשר יש צורך לשמור מחרוזות שלמות במדויק.
  - בתוך גרשיים אלו לא מתבצעות החלפות של תבניות בשמות הקבצים המתאימים.

דוגמה:

```
> echo "*.c" : *.c
*.c : main.c app.c hello.c
```

- גרשיים בודדים ' '

תפקיד: מונעים החלפות בתחומם.

דוגמה:

```
> echo lets make some '$$'
lets make some $$
```

- גרשיים הפוכים

תפקיד: מבצעים command substitution

  - ניתן "לשרשר" פקודות .

דוגמה:

```
> echo The length of $str is `echo -n $str | wc -c`
The length of Hello is 5
```

1.3 עבודה עם תסריטים

1. **הגישה לפרמטרים המועברים** בשורת הפקודה לתסריט

סימן	משמעות
$n	קריאת הפרמטר ה-n לתסריט
$*	רשימת כל הארגומנטים לתסריט
$@	שמירה על מספר הארגומנטים הנכון, במידה וייתכנו רווחים בתוך הארגומנטים.
$0	שם התסריט
$#	מספר הארגומנטים

2. **הרצת תסריט**: בתחילת התסריט יש להוסיף את השורה : `#!/bin/bash`

3. **סוגי העברת פרמטרים** לתסריטי עזר

סוג העברה	דוגמה
שורת הפקודה	`helper_script $arg1 $arg2`
Pipeline	`echo $arg1 $arg2 | helper_script`
קובץ זמני	`echo $arg1 $arg2 > temp` `helper_script < temp`

4. **סוגי החזרת ערכים**

סוג החזרה	דוגמה
backticks	`result = `helper_script` `

helper_script | another_script	Pipeline
helper_script > temp	קובץ זמני

1.4 חישובים אריתמטיים

- ניתן להחליף ביטוי אריתמטי בערכו על ידי $((<expression>))
  - הערך המספרי של המחרוזות ישמש בחישוב הביטוי
  - משתנים יוחלפו בערכם (גם ללא $)
  - אם אחד הארגומנטים אינו מספר, ערכו יהיה 0 בחישוב
  - משתנה שאינו מוגדר, או ערכו אינו מספר יחושב כ-0
  - פעולות חשבוניות פשוטות: +, -, * ו-/ , השמות: =, הגדלות והקטנות: =+, =-, -- ו-++
  - ניתן גם להשתמש בפקודה let, לדוגמא: let n++

1.5 מבני בקרה

- לולאת while

```
while <expression>; do
   <commands>
done
```

- לולאת for

מעבר על איברי מערך

  - <varname> הוא שם המשתנה שיכיל בכל פעם איבר מהרשימה
  - <array> היא רשימה של מחרוזות

```
for <varname> in <array> ; do
   <commands>
done
```

לחלופין:

```
for ((i = 1; i <= 3; i++)); do
   <commands>
done
```

- תנאי if

```
if <expression>; then
   <commands>
fi
```

- תנאי if עם else

```
if <expression>; then
   <commands>
else
   <commands>
fi
```

- בכל התנאים ניתן להשתמש באופרטורים &&, || שמשמעותם דומה ל-C.

1.6 סוגי תנאים

1.6.1 תנאי מהצורה [[ ]]

- בתוך [[ ]] האופרטורים ==, > ו-< משווים מחרוזות (לפי סדר לקסיקוגרפי)
- ניתן לבצע השוואות על ערכי מספרים בעזרת דגלים כגון -eq, -gt, -le
- התנאי <filename> -f בודק האם קיים קובץ בשם <filename>
- התנאי <dirname> -d בודק האם קיימת תיקייה בשם <dirname>
- האופרטור = מאפשר התאמת מחרוזת לתבנית
  - הארגומנט השמאלי הוא מחרוזת רגילה
  - הארגומנט הימני הוא תבנית אשר יכולה לכלול את הסימנים *, ? ו-[ ]
- האופרטור =! הוא השלילה של אופרטור ההתאמה =
  - המשמעות של = ו-== זהה

דוגמה:

```
> end_with_z="some string with z"
> if [[ "$end_with_z" = *[zZ] ]]; then echo match; fi
match
> if [[ "this string start with t" = t* ]]; then echo true; fi
true
> if [[ "this string doesn't start with t" = [^t]* ]]; then echo true; fi
> file=test4.in
> if [[ $file = test*.in ]]; then echo test file; fi
test file
> if [[ "string doesn't start with t" != t* ]]; then echo true; fi
true
```

1.6.2 תנאי מהצורה (( ))

- בתוך (( )) האופרטורים ==, =!, > ו- < מתייחסים לערכים מספריים
- אין צורך לרשום $ לפני שם משתנה
- ניתן לבצע פעולות חשבוניות
- תנאים המוגדרים בעזרת (( )) מתנהגים כמו ביצוע פעולות חשבוניות בעזרת $(( ))

דוגמה:

```
> if (( 11 < 7 )); then echo true; fi
> i=5
> if (( i >= 0 && i <= 10 )); then echo true; fi
true
> if [[ 11 -eq 11 ]]; then echo true; fi
true
> if (( 0 == Hello )); then echo true; fi
true
> if (( ++i == 6 )); then echo true; fi
true
> if (( ++i == 6 )); then echo true; fi
```

1.7 פונקציות

- השימוש בפונקציה יכול להתבצע רק אחרי הגדרתה.

```
function <name> {
   <commands>
}
```

- אפשר להעביר לפונקציה ארגומנטים
  - משתמשת בהם בדומה לשימוש בארגומנטים המועברים לתסריט
  - לא יכולה לגשת לארגומנטים של התסריט שקרא לה

- משתנה מקומי בפונקציה בעזרת המילה local:
- "החזרת" ערכים מפונקציה בעזרת command substitution (backticks)

1.8 קריאת קלט

- קריאת שורה מהקלט הסטנדרטי: `read <flags> <variable name>`
- השורה תיקלט לתוך שם המשתנה שהוגדר
  - הדגל a- יחלק את השורה לפי מילים לתוך מערך
- הביטוי read יוחלף על ידי bash בשורת קלט שתיקלט מהקלט הסטנדרטי
- דוגמה:

```
> read line
Hello world
> echo $line
Hello world
> read -a line
Hello world
> echo $line
Hello
> echo ${line[*]}
Hello world
```

- קריאת קובץ ע"י הפניית קלט ושימוש ב-read: `read line < filename`
- קריאה מקובץ שורה אחר שורה:

```
while read line; do
   echo $line
done < "$1"
```

2 קבצי Makefile

דוגמה ל-Makefile:

```
CC = gcc
OBJS = a.o b.o
EXEC = prog
DEBUG_FLAG = # now empty, assign -g for debug
COMP_FLAG = -std=c99 -Wall -Werror

$(EXEC) : $(OBJS)
 $(CC) $(DEBUG_FLAG) $(OBJS) -o $@
a.o : a.c a.h b.h
 $(CC) -c $(DEBUG_FLAG) $(COMP_FLAG) $*.c
b.o : b.c b.h
 $(CC) -c $(DEBUG_FLAG) $(COMP_FLAG) $*.c
clean:
 rm -f $(OBJS) $(EXEC)
```

ניתן להשיג את שורות התלויות עבור קבצי c בעזרת הפקודה gcc -MM <files>. למשל, אם הקובץ a.c מבצע #include ל-a.h ו-b.h, אז הפקודה gcc -MM a.c תגרום להדפסת:

```
a.o: a.c a.h b.h
```