סיכום כלי פיתוח – שחר הכרי 2018

עזרים וסימונים:

$-משתמש רגיל #-משתמש מערכת

~ - (טילדה) סימון לתיקיית הבית של המשתמש.

דוג':~/music – תיקיית music שבתיקיית הבית.

/ - root directory (שבתוכו נמצאת תיקיית הבית (/home/shahar/ שהיא ~)).

. – הpath הנוכחי

.. – הpath הקודם בהיררכיה

- -הדיירקטורי הקודם שעבדנו עליו (לאו דווקא הקודם בהיררכיה).

man ___ – פירוט כל הפקודות הקשורות לאותה פקודה

לחצן "**TAB**" – השלמה אוטומטית ,גם של **שמות** וגם של **פקודות**!

דוג': אם נרשום mk ואז TAB יודפסו כל הפקודות המתחילות בmk.

Cntl+shift+c/v – העתקה\הדבקה.

Shift+pageUp/Down – עלייה וירידה ללא צורך בעכבר.

חץ למעלה\למטה – פקודות אחרונות.

= Cntl shift - הגדלת כתב.

- Cntl - הקטנת כתב.

Cntl c – עצירת התהליך.פקודה.

> - משפך (שינוי standart output) - דריסה

>> - append הוספה (סוג של משפך)

<u>הערות:</u>

cat text1 > text2 – מעתיק את תוכן text1 לtext2 אך דורס את תוכנו.

Echo "something.." > text2 משנה את תוכן text2 ל-"something.." אך דורס את תוכנו.

<u>Append(הוספה ללא דריסה).</u>

Cat text1 >> text2 – מוסיף את תוכן text1 לtext2 בלי לדרוס את תוכנו.

Echo "something.." >> text2 מוסיף את "something.." לtext2 בלי לדרוס את תוכנו.

Regex(regular expressions)

"*" – asterisk/wildcark דוגמא:

*123 ls יציג את כל הקבצים והתיקיות המתחילות ב123:

123 1233 12334 1233.txt

Ls *33* יציג את כל הקבצים והתיקיות ששמם מכיל 33

1233 12334 1233.txt

הצגת מה שמתחיל ב"2" או ב"t" – ls 2* t*

הערה: בפקודות רבות * ייצג תו אחד מכל סוג ו **.** ייצג **כמות אין סופית** של תווים מכל סוג (לדוג בgrep).

[ ] – text[1-4].txt יתן את כל הקבצים בשם text_.txt

כאשר במקום "_" תבוא ספרה בין 1 ל-4

text[1357].txt את כל הקבצים בשם text_.txt

כאשר במקום "_" תבוא ספרה אחת (לא יותר) מבין הספרות 1 3 5 7

[^abc]- כל char חוץ מabc

[^a-c]- כל char חוץ מabc

/s – whitespace characters – דוג: רווחים tabs, newlines.

/S – non whitespace characters.

" \" – רווח (לדוג' ביצירת שם).

**** = סלש \.

\{\} – \{5\}{0-7} ייתן 5 ספרות מ0 עד 7.

[[:space:]] = רווח\שורה חדשה.

[[:blank:]] – רווח רגיל(\n) או רווח tab (\t).

[:lower:] – אותיות קטנות abc.

[:upper:] – אותיות גדולות ABC.

[:xdigit:] – מספרים בהקסה.

^ - תחילת שורה.

$ - סוף שורה.

Outputs

> file redirects stdout to file

1> file redirects stdout to file

2> file redirects stderr to file

&> file redirects stdout and stderr to file

The **>>** appends to a file or creates the file if it doesn't exist.

The **>** overwrites the file if it exists or creates it if it doesn't exist.

Throw unwanter outpot to **/dev/nu**

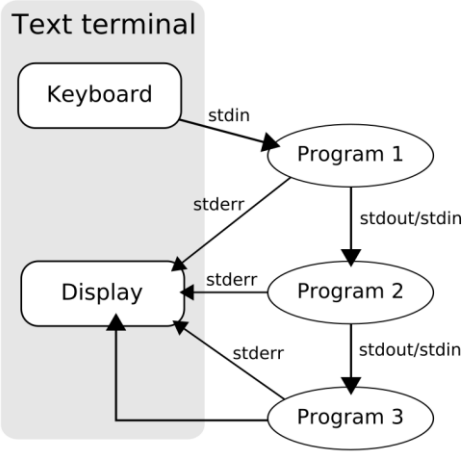
2>/dev/null – will discard stderr

Pipeline

Program 1 stdout == Program 2 stdin

Program 2 stdout == Program 3 stdin

Program 3 stdout == console stdin



For example, to list files in the current directory (ls), retain only the lines of ls output containing the string "key" (grep), and view the result in a scrolling page (less), a user types the following into the command line of a terminal:

ls -l | grep key | less

By default, **the standard error streams ("stderr")** of the processes in a pipeline **are not passed on through the pipe; instead, they are merged and directed to the console.**

הערה: **{פקודה} exec- פקודה** יעשה פעולה דומה לpipeline

<u>פקודות:</u>

<u>הערה:</u> יש פעמים שנראה אופציה שלא מופיעה ב"man" של הפקודה אבל משלבת 2 אופציות , לדוגמא grep -iv היא בעצם grep -i -v...

<u>date</u> – תאריך ושעה

<u>cal</u> – תיתן הדפסה של לוח שנה של החודש. cal 12 ייתן לוח שנה שנתי.

<u>reset/clear</u> – ניקוי מסך הטרמינל.

<u>passwd</u> – שינוי סיסמת user.

<u>ls</u> – הצגת תוכן התיקייה

ls -a – הצגת תוכן התיקייה כלל תיקיות נסתרות

Ls -l – הצגה ברשימה עם מידע כלל .

ls -t – סידור לפי זמן

Ls -S – הצגה בסידור לפי גודל

Ls -h – הצגה בסידור לפי גודל בהצגה רגילה (כמו ls רגיל ,לא רשימה)

Ls -r – הצגה בסדר הפוך.

Ls -ltr – קומבינציה של -l -t -r

Ls -lst – ממיין לפי זמן עבודה על הקובץ בסדר יורד.

<u>dir</u> – כמו ls

<u>pwd</u> – הpath הנוכחי

<u>touch</u> – יצירת קובץ.

אם קיים קובץ בעל שם זהה לא תיצור קובץ(לא תבצע דריסה) ולא תתריע על שגיאה.

<u>mkdir</u> – יצירת תיקייה

<u>rmdir</u> – מחיקת תיקייה (לא תימחק אם לא ריקה).

<u>rm</u> – מחיקת קובץ

rm -r – מחיקת תיקייה והתוכן שלה כולל תיקיות פנימיות (-r = רקורסיה).

rm -rf - / לא להשתמש !!! מחיקה הכי חזקה!!!

<u>cp</u> – העתקה (copy)

cp 1.txt 2.txt מעתיק את 1.txt ל-2.txt (אם לא קיים הוא יוצר את 2.txt)

.. cp 1.txt מעתיק את 1.txt ל תיקייה קודמת בהיררכיה.

cp 1.txt ../2.txt מעתיק את 1.txt ל2.txt בתיקיה הקודמת בהיררכיה.

cp -r – לצורך העתקת תיקיות לדוגמא: cp -r 2/ 3/ מעתיק את תוכן תיקייה 2 לתיקייה 3

Cp -r 2 3 מעתיק את תיקייה 2 כלל התוכן אל תוך תיקייה 3

cp -i text1 text2 – ידרוס את text2 ויעתיק אליו את text1 אך ישאל אם לדרוס אם יש משהו אם text2 קיים.

<u>mv</u> – העברה (move כמו cut+rename)

mv -i – שואל אם לדרוס לפני (במידה וקיים קובץ בשם שאליו אנו רוצים להעביר)

<u>chmod</u> – שינוי הרשאות קובץ.

לשם יצירת script יש לתת תו הרשאות 755:

chmod +x script.sh **או** chmod 755

<u>File</u> – תיתן את פורמט הקובץ.

לדוג': עבור test2.txt תיתן הפקודה file text

Test2: ASCII text

<u>echo</u> – מדפיס מה שרשום אחרי הפקודה(אל הstdout)

echo "111" ידפיס 111

echo -n "something" ידפיס "something" בהמשך לשורה הקודמת ורק אז ירד שורה

<u>cat</u> – catenate מדפיס את מה שכתוב בקובץ (אל הstdout).

cat 1.txt ידפיס את תוכן 1.txt .

Cat 1.txt > 2.txt דורס את 2.txt ושם בו את תוכן 1.txt

head - הפקודה מדפיסה מספר שורות מתחילת הקובץ.

אם לא מציינים את מספר השורות יודפסו 10 השורות הראשונות `head text`

הדפסת 20 השורות הראשונות **head -n 20 text.txt**

tail - הפקודה מדפיסה מספר שורות מסוף הקובץ.

אם לא מציינים את מספר השורות יודפסו 10 השורות האחרונות: `tail text`

הדפסת 20 השורות האחרונות: **tail -n 20 text.txt**

Grep – מוצא את השורות המכילות את המחרוזת המבוקשת.

grep "aa" – יחזיק את השורות המכילות "aa"

grep ^# - יחזיר את השורות המתחילות ב"#"

grep ^#1 – יחזיר את השורות המתחילות ב"#1"

grep KELLY$ - יחזיר את השורות המסתיימות ב"KELLY"

grep -I – אותה פעולה עם ignore cases.

grep -v – יחזיר את השורות שלא מכילות את המחרוזת – invert.

***הפקודה עובדת גם עם קובץ(grep hey 1.txt) וגם עם String (cat 1.txt | grep hey)**

Sed – החלפת\מחיקת טקסט

sed '/^#/ d' – מוחק בקובץ את השורות המתחילות ב"#".

sed '/00/ d' – מוחק בקובץ את השורות המכילות "00".

sed '/11/c\223344' – יחליף את השורות בקובץ המכילות "11" בשורה "223344".

sed 's/old/new/g' – יחליף את המילה old במילה new בגל מקום בתוך הקובץ.

sed 's/ \. \/ / /g' - מחליף "./" בכלום.(לפני כל תו ./ נאלץ לשים "\" .

***הפקודה sed עובדת גם עם קובץ(sort 1.txt) וגם עם String (cat 1.txt | sort).**

Sed -I – תעשה את אותה פעולה אך **תשנה את תוכן הקובץ!**

דוג': sed -i '/00/ d' 1.txt – מוחק בקובץ את השורות המכילות "00".

***הפקודה sed -i עובדת רק עם קובץ(sed 's/old/new/g' 1.txt) ולא עם String!**

Tr – החלפת תוים trandlate

tr 'abcd' '1234' או **tr [a-d] [1-4]**, יחליף בהתאמה "a" ל"1", "b" ל"2" וכו..

cut – פעולת "גזירת טקסט" מתוך קובץ

cut -c2 test.txt – תדפיס את העמודה השניה בקובץ (c=column).

cut -c2-5 test.txt – תדפיס את העמודות 2-5 בקובץ.

cut -c2- test.txt – תדפיס החל מעמודה 2 עד הסוף בקובץ.

cut -c-5 test.txt – תדפיס את כל העמודות על עמודה 5 בקובץ.

cut -d';' -f2 text.txt – תדפיס את המילה השנייה המופרדת ע"י התו ';' בכל שורה.

cut -d';' -f2,4 text.txt – תדפיס את המילה השנייה והרביעית המופרדת ע"י התו ';' בכל שורה.

Sort – מיון

Sort– מיון השורות לפי ערך ascii (השוואה כמו ע"י מטודת CompareTo של String בjava).

Sort -n– מיון לפי ערך מספרי (על השורות שמכילות רק מספר ,את שאר השורות ימקם בתחתית).

Sort -k

***הפקודה עובדת גם עם קובץ(sort 1.txt) וגם עם stdout (cat 1.txt | sort)**

wc – פעולות count (ספירה)

לדוגמא: wc -l file.txt או cat file.txt | wc -l ידפיס את מספר השורות(l=lines) בקובץ.

Uniq – פקודה העובדת עם שורות שחוזרות על עצמן.

uniq -c – תדפיס כל שורה פעם אחת(גם אם מופיה יותר) ולידה את מספר הפעמין שהופיעה.

uniq -d – תדפיס רק את השורות שחוזרות על עצמן, אחת לכל קבוצה..

uniq -s N – עושה את אותה פעולה תוך כדי התעלמות מN התווים הראשונים.

ignore cases – **uniq -i**

Find – פקודה אשר מחפשת קובץ לפי דרישה

לדוג': **find / -name "*.txt"** ייתן את כל בקבצים עם סיומת txt בתיקיית / .

Find . -name "*.txt" ייתן את כל בקבצים עם סיומת txt בתיקייה הנוכחית.

Find / home -iname tmp.txt – מחפש את הקובץ לפי הכתוב ומתעלם מסוג האותיות(ignore cases קטנות\גדולות).

find . -type f -name "*" - ימצא את כל קבצים - files (-type f).

find . -type d -name "*" - ימצא את כל תיקיות – dir's (-type d).

find / -size +50M -size -100M - מוצאת את הקבצים בין 50 ל100 מגה.

find /tmp -empty - מציגה את הקבצים הריקים בספריה /tmp
find . -type d -empty "*" – מחפש תיקיות ריקות.

find . -maxdepth 1 -type f -name "."* – מציגה רק קבצים מוסתרים בספריה הנוכחית בלבד

maxdepth – עומק ברקורסיה (שלא יכנס לתיקיות מעבר לתיקייה הנוכחית).

find / -cmin -60 – קבצים שנערכו\נוצרו ב60 דק' האחרונות.

find /home -atime 50 – קבצים שנערכו\נוצרו ב50 יום האחרונים.

find . -type f -perm 0777 - מחפש קבצים עם הרשאה 0777.

find / -type f ! -perm 0755 – מחפש קבצים ללא הרשאה 0755.

find . -name "*.txt" -print0 | xargs -0 rm יעשה: rm ./a.txt ./b.txt ./c.txt

find . -name "*.txt" -print0 - ימצא את כל הקבצים עם סיומת txt

בתיקייה הנוכחית וידפיס אותם בפורמט של כתיבת פקודת bash: ./a.txt ./b.txt ./c.txt

הפקודה rm **xargs -0 |** לוקחת את הפלט שיוצא מהפקודה **find . -name "*.txt" -print0** ושמה אותו אחרי הפקודה rm: **פלט rm**

find *.txt -print -exec cat {} \;

Locate – פקודת חיפוש יעילה יותר אך מוצאת רק קבצים שעברו "אינדוקס".

כלים:

Nano – תוכנה מובנית לעריכת טקסט.

פתיחה ע"י הפקודה nano.

אם נרצה לערוך קובץ קיים נרשום nano text (שם הקובץ ללא סיומת לאחר nano).

קיצורים למטה ע"י מקש+cntl 22

לדוג: cntl+x = יציאה , cntl+o = שמירה.

אם נצא בלי לשמור nano ישאל אותנו אם לשמור את השינויים.

More – פורמט הצגה בצורת דפדוף דפים.

לחיצה על מקש space תרד עמוד אחד כל פעם.

לחיצה על מקש Enter תרד שורה.

לחיצה על Q תצא מהממשק.

Less – הרחבה של More

גלילה ממשיכה מההתחלה ועד הסוף ללא הגבלת עמודים.

הקשת מספר תוריד אותך מספר שורות למטה (לדוגמה הקשת המספר 10 תוריד 10 שורות למטה).

אותם מקשים עובדים:

לחיצה על מקש space תרד עמוד אחד כל פעם.

לחיצה על מקש Enter תרד שורה.

לחיצה על Q תצא מהממשק.

הערה:

כל פקודה שנשים אחריה "more |" או "less|"

תציג את הפלט בממשקים האלו (בעזרת pipeline).

לדוגמא: ls ~ |less

רשימת פקודות של הקורס A-Z

1.	alias	Create an alias
2.	break	Exit from a loop
3.	cal	Display a calendar
4.	case	Conditionally perform a command
5.	cat	Display the contents of a file
6.	cd	Change Directory
7.	chgrp	Change group ownership
8.	chmod	Change access permissions
9.	chown	Change file owner and group
10.	clear	Clear terminal screen
11.	cp	Copy one or more files to another location
12.	cut	Divide a file into several parts
13.	date	Display or change the date & time
14.	dir	Briefly list directory contents
15.	echo	Display message on screen
16.	ed	A line-oriented text editor (edlin)
17.	enable	Enable and disable builtin shell commands
18.	expr	Evaluate expressions
19.	find	Search for files that meet a desired criteria
20.	for	Expand *words*, and execute *commands*
21.	grep	Search file(s) for lines that match a given pattern
22.	head	Output the first part of file(s)
23.	history	Command History
24.	if	Conditionally perform a command
25.	join	Join lines on a common field
26.	less	Display output one screen at a time
27.	let	Perform arithmetic on shell variables
28.	ln	Make links between files
29.	locate	Find files
30.	logout	Exit a login shell
31.	ls	List information about file(s)
32.	man	Help manual
33.	mkdir	Create new folder(s)
34.	more	Display output one screen at a time
35.	mv	Move or rename files or directories
36.	paste	Merge lines of files
37.	pwd	Print Working Directory
38.	read	read a line from standard input
39.	rm	Remove files
40.	rmdir	Remove folder(s)
41.	select	Accept keyboard input
42.	set	Manipulate shell variables and functions
43.	sleep	Delay for a specified time
44.	sort	Sort text files
45.	tail	Output the last part of files
46.	touch	Change file timestamps
47.	umask	Users file creation mask
48.	uniq	Uniquify files
49.	until	Execute commands (until error)
50.	wc	Print byte, word, and line counts
51.	whereis	Report all known instances of a command
52.	while	Execute commands
53.	whoami	Print the current user id and name (`id -un')
54.	xargs	Execute utility, passing constructed argument list(s)

סקריפט Scripting

- הערה (comment) , **#/bin/bash** – הערה חובה בכל סקריפט (קונבנציה)

קובץ הסקריפט יהיה בסיומת **sh.** וינתנו לו הרשאות **755** –

chmod +x script.sh או chmod 755 script.sh

הרצת הסקריפט תתבצע עי **./script.sh**

שמירה מדויקת על **רווחים** חשובה להרצת הסקריפט! (גם בהצהרה ולרוב גם בתנאים\לולאות).

הערה: "something" **echo -n** ידפיס "something" בהמשך לשורה הקודמת ורק אז ירד שורה

"read b" – קליטה מהמשתמש אל תוך המשתנה b.

exit 1 – הפקודה **תפסיק את הסקריפט** באמצע בשורה שנשים אותה.

משתנים arguments:

הצהרה כל משתנית תעשה ע"י כתיבת שמותיהם ולמה הם שווים(לא חובה):

X=5

y2

Number1=true

String="hey"

לבקשת **הערך (value) של המשתנים** נשים לפני שם המשתנה את הסימן $ (דולר)

דוגמא: x=5

Echo "x" – ידפיס "x"

Echo "$x" – ידפיס "5"

לבקשת **ערך של פקודה** נשתמש ב **$( )** או ב ' ' :

b=$(pwd) – התוצאה של הפקודה pwd נשמרת בערך b.

b=`pwd` - התוצאה של הפקודה pwd נשמרת בערך b. **(שימוש בגרש ` במקש הטילדה (~) !)**

B=$(date + %d / %m + %Y) – יכניס לb את "21/06/2019"

B=$(date + %d-%m-%y) – יכניס לb את "21-06-19"

ניתן להצהיר על כמה משתנים **באותה שורה** ע"י מפריד ";"

דוגמא: x=5;y=7;number=true

***תקף לכל מקום בקוד** – תמיד ניתן לכתוב כמה שורות בשורה אחת והפריד בינהן בעזרת ";"

דוגמא:

while [condition]; do

יעשה אותה פעולה כמו

while [condition]

do

משתני Boolean אפשר להצהיר בשני דרכים:

1. B=true

Boolean לא באמת קיים אלא מחרוזת המכילה "true" או "false" ובודקים אותה כמחרוזת רגילה , רק כאשר משתמשים באופרנדים a- o- של תנאי משווים לtrue$ false$ הקבועים.

משתני String:

1. Str=" - יהפוך את המשתנה Str למחרוזת ריקה (ערכה null).
2. Str="hey" – הצהרה רגילה על משתנה מחרוזרת
3. Str=$number – בהנחה שיש במשתנה "number" מספר ולא מחרוזת

   יכנס לStr המחרוזת המייצגת את המספר ,דוג "53" ולא

   המספר 53!
4. **b=${a:12:5}** –יכניס לb תת מחרוזת של זו שנמצאת בa **החל ממקום 12** (אינדקס מתחיל מ0) **באורך 5 תווים**(אפשרי לשים במקום מספרים ערך של משתנה , לדוג': $c).

משתנים חיצוניים מהמשתמש command line argument:

./script.sh word1 word2 word3

- $0 would contain "./script.sh "
- $1 would contain "word1"
- $2 would contain "word2"
- $3 would contain "word3"

משתנים מובנים:

$USER , $true , $false

$? – יכיל מספר שגיא (1,2,..) אם הקוד שמעליו נתן error (גם אם עושים 2>/dev/null), יחזיר 0 אחרת.

$# - יחזיר את משפר המשתנים שנכנסו מהמשתמש(לא כלל הראשון $0)

דוגמא: עבור **"./script.sh word1 word2 word3"** המשתנה יכיל את הספרה **3** (אפילו שיש 4 כלל $0).

$*** או **$@ - יגיל מחרוזת עם כל הערכים מ$1 והלאה עד האחרון עם רווחים בינהם.

(בעזרת for ומשתנה זה נוכל לעבור על כל המשתנים).

אופרנדים של אריתמטיקה:

+ - * / (פעולות רגילות) ++ -- (הגדלה\הקטנה ב1 לדוגמא ++a)

% מודולו בעזרת הפקודה expr - num=`expr $n % 10`

דרכי שימוש:

```
n1=5; n2=7
n3=$[$n1+$n2]
n3=$ [$n1 + $n2 ]
n3=$[$n1+7]
n3=$[5+7]
let n3=$n1+$n2
n3 = `expr $n1 + $n2`    # using ` sign in tilde key (~)
let " n3 = $n1 +    $n2"
let n3=5+$n2
```

כל 8 הפקודות האחרונות עושות את אותה פעולה – מכניסות לערך n3 את המספר 12

(אם אחרי כל פקודה מ8 האחרונות נשים "echo $n3" יודפס המספר 12 8 פעמים..).

הערך שn3 מקבל הוא מספרי!

טעות נפוצה!: n1=5; n2=7**; n3=$n1+$n2**; echo $n3

במקרה זה יקבל n3 ערך מחרוזת יודפס "5+7" ולא "12"!.

משפטי תנאי if:

Syntax:

Regular if	If + else	Else if	If(+else) inside if
If [condition] then ##Do something fi	If [condition] then ##Do something else ##Do something fi	if [condition] then ##Do something elIf [condition] then ## Do something else ## Do something fi	if [condition] then if [condition] then ##Do something else ## Do something fi fi

הערה: הפקודה **break** תעצור את הלולאה.

אופרנדים:

(בינארי = השוואה מספרית , אסקי = השוואת מחרוזות לפי ערך ascii)

	Equal to (==)	Not equal to (!=)
Ascii	= == דוגמאות: if ["$a" = "$b"] if ["$a" == "$b"]	!= דוגמאות: if ["$a" != "$b"] הערה: אם רוצים לבדוק אם שווה **לסט** (כמו [a-z]) צריך לשים **בסוגריים כפולים והסט ללא גרשיים!**. -> אותו דבר בequal
Binary	-eq דוגמא: if ["$a" -eq "$b"]	-ne דוגמא: if ["$a" -ne "$b"]

	Gretater then (>)	Gretaer or equal (>=)
Ascii	[[>]] [\>] דוגמאות: if [["$a" > "$b"]] if ["$a" \> "$b"]	
Binary	-gt דוגמא: if ["$a" -gt "$b"]	-ge דוגמא: if ["$a" -ge "$b"]

	less then (<)	less or equal (<=)
Ascii	[[<]] [\<] דוגמאות: if [["$a" < "$b"]] if ["$a" \< "$b"]	
Binary	-lt דוגמא: if ["$a" -lt "$b"]	-le דוגמא: if ["$a" -ge "$b"]

:Boolean

b1=true; b2=true

	Logical AND	Logical OR
בתוך סוגריים: **[-o]**	**-a** דוגמאות: If ["$b1" -a "$b2"] If ["$b1" -a "b2"] if ["$b1" -a "$b2"]	**-o** דוגמאות: If ["$b1" -o "$b2"] If ["$b1" -o "b2"] if ["$b1" -o "$b2"] הערה: השני הפקודות האלו צריכים להשוות את המשתנים ל$true או $false הקבועים (לא ייתן תוצאות נכונות עם מחרוזות).
מחוץ לסוגריים: **[ ] || [ ]** **או בתוך סוגריים כפולים:** **[[&&]]**	**&&** דוגמאות: If [condition] && [condition] If [[$b1 && $b2]] If [[$b1 && b2]] If [[b1 && b2]]	|| דוגמאות: If [condition] || [condition] If [[$b1 || $b2]] If [[$b1 || b2]] If [[b1 || b2]]

אופרנדים של מחרוזות:

String="hey"

is null	**is NOT** null
-z דוגמאות: if [-z "$String"] (תנאי לא יתבצע)	-n דוגמאות: if [-n "$String"] (תנאי יתבצע)

is file	**is directory(Folder)**
-f דוגמאות: if [-f "$String"]	-d דוגמאות: if [-d "$String"]

Switch case:

מימוש:

```
read ans
case $ans in
   [Yy] | [yY][Ee][Ss] ) echo "you chose yes"
      ##Do Something
    ;;
    [Nn] | [nN][oO] ) echo "you chose no"
       ##Do Something
    ;;
   *) echo "your answere is not recognized"
esac
```

לולאות:

For:

מימוש:

```
for ((    i=0; i<=$# ; i++ ))
do
  ##Do Something
done
```

For each:

פועלת כמו **"for each" על String**. היא עוברת על כל "מילה" .

(מילה=תת מחרוזת המופרדת ברווחים משאר המחרוזת – "מילה1 מילה2 מילה3 מילה4").

דוגמא:

```
#!/bin/bash
            for i in $( ls )
                 do
                  echo $i
            done
```

סקריפט אשר ידפיס את שמות כל הקבצים בתיקיה הנוכחית אחד אחד.

דוגמא:

סכימת כל המשתנים שהתקבלו מהמשתמש שהם מספרים:

sum=0
for i in $@;do
  if [$i -eq $i] 2>/dev/null
  then
  echo $i
  let sum=$sum+$i
  fi
done
echo "this is the sum of the numbers $sum"
	קלט\פלט:
	shahar@DESKTOP-6N1O9P7:~$ ./kript4.sh 22
	55 dfg
	22
	55
	this is the sum of the numbers 77

While:

מימוש:

```
while [ condition ]
   do
      ##Do Something
done
```

מימוש לולאת for רגילה בעזרת לולאת while:

```
COUNTER=0
while [ $COUNTER -lt 10 ]; do
                    echo The counter is $COUNTER
                    let COUNTER=COUNTER+1
done
```

הערה: **done < ./file.txt** ייתן את הקלט ללולאה מתוך הקובץ במקום מהמשתמש במקרה שיש בלולאה **read X** **ויכניס בכל איטרציה שורה מקובץ הטקסט את תוך המשתנה X**.

דוגמאות:

while [$NUM -lt $SIZE]
do
  read FILE
  if [-f $FILE]
  then
  echo $FILE >> files
  elif [-d $FILE]
  then
  echo $FILE >> dirs
  fi
  NUM=`expr $NUM + 1 `
done < /tmp/listing
	while read LINE
	do
	  Echo "The Name is $LINE"
	done < names.txt

לכתוב סקיפט (15)

כתוב Script אשר מקבל כפרמטרים את שם הקובץ וסוג (F ל קובץ ,D ל ספריה).
הפרמטרים הנ"ל יסופקו ל Script בשורת הפקודה (command line argument- כפי שהודגם בכיתה).
הScript יבצע בדיקה האם מספר הפרמטרים שנשלחו אליו בשורת הפקודה (command line) אכן מספיקים לשם הרצתו, ובמידה ואינם מספיקים יש להציג למשתמש הודעה מתאימה .
ה Script יבדוק לפי הסוג שנשלח אליו הוא שם של קובץ רגיל (F) או סיפרייה (D).
אם האות שונה מ F ו D - יש להציג למשתמש הודעה מתאימה .
במידה והקובץ הוא קובץ רגיל יש להציג מתוכו את כל השורות המכילות את המילים המתחילות ב "W".
במידה והקובץ הוא סיפרייה יש להציג את רשימת הקבצים ותתי הספריות שבה בפורמט מפורט (כלומר שם, גודל הקובץ, בעלים, הרשאות וכו').
אם הקובץ "לא עונה" על שני התנאים הקודמים יש להודיע למשתמש כי זהו איננו קובץ ולא סיפרייה, ולשאול אותו האם הוא מעוניין לנסות שוב ? שים לב : רק לחיצה על המקש Y תאפשר קליטת אותם פרמטרים (שם הקובץ וסוג) בעזרת קריאה מהמסך (ולא מ command line) וביצועה מחודשת של דרישות ה Script .

```
f=$1;s=$2
while [ true ];do
  if [ -z "$f" ] || [ -z "$s" ];then
    "echo "not enough virlables
  else
      if [[ "$f" != [fF] ]] && [[ "$f" != [dD] ]];then
        if [[ "$s" != [Ff] ]] && [[ "$s" != [dD] ]];then
              "echo "kind is not identified
          else
                kind=$s
                file=$f
          fi
        else
              kind=$f
              file=$s
        fi
     if [[ $kind == [Ff] ]]
      then
        cat $file 2>/dev/null | grep ^[wW] 2>    /dev/null
        if [ $? -ne 0 ];then
            "echo "this isnt a file
        fi
     elif [[ $kind == [Dd] ]];then
        ls $file 2>/dev/null
        if [ $? -ne 0 ];then
        "echo "this is not a dir "
        fi
      fi
  fi

fl=true
echo "do you want to continue?"
while [ $fl == "true" ];do
   read ans
   case $ans in
   "[Yy] | [yY][Ee][Ss] ) echo "we will continue
   echo enter filename and type
   read f
   read s
    fl=false
     ;;
     [nN] | [Nn][Oo] ) echo "we will stop"
    ;exit 1
    ;;
     *) echo "answer is not identified"
    ;continue
    ;;
     esac
  done
done
```

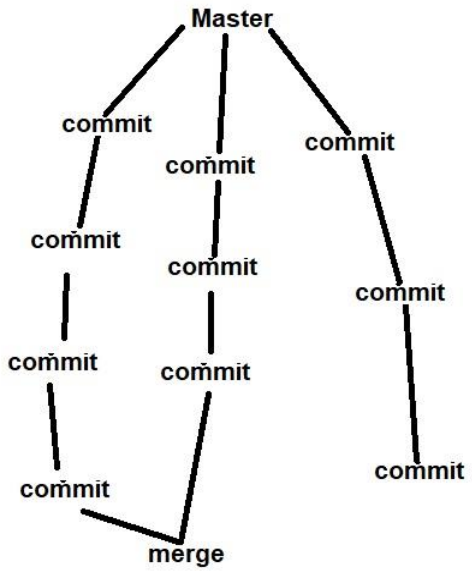
נותן את מספר המחלקים המשותפים לשני המספרים:

```
read a b
m=$a
if [ $b -lt $m ]
then
m=$b
fi
while [ $m -ne 0 ]
do
x=`expr $a % $m`
y=`expr $b % $m`
if [ $x -eq 0 -a $y -eq 0 ]
then
echo
break
fi
m=`expr $m - 1`
done
```

קוד אשר מדפיס קודם את כל שמות הקבצים ואז את כל שמות התיקיות בתיקייה הנוכחית.

```
ls > /tmp/listing
SIZE=`cat /tmp/listing | wc -l`
NUM=0
while [ $NUM -lt $SIZE ]
do
    read FILE
    if [ -f $FILE ]
    then
        echo $FILE >> files
       elif [ -d $FILE ]
       then
          echo $FILE >> dirs
     fi
     NUM=`expr $NUM + 1`
done < /tmp/listing
echo "Files:"
echo "-----"
cat files
echo " "
echo "Directories:"
echo "-----------"
cat dirs
rm files dirs /tmp/listing
```

GIT



Git init – יצירת repository (directory אשר מנוהל ע"י git).

"הוספת קובץ" : git add (הופך את הקובץ לtracked או שומר את השינויים שעברו מהcommit הקודם ,שינויים אלו נשמרים בcommit הבא).

תהליך זה נקרא מעבר למצב stage.

"שמירת השינויים": git commit

"מקפיא" את מצב הקבצים מהstaging ושומר את מצבם בcommit (מעין שמירה מפוקחת ,כל commit שמור בgit מה שמאפשר "לחזור אחורה" במידה ונעשה נזק בלתי הפיך לקוד המנוהל ע"י הgit).

"git commit -m "comment for the commit – פעולת קומיט + הערה.

כמו כן כלbranch בנוי מcommitים ובסוף כל הbranchים מאוחדים לגרסה אחת (הקרויה לרוב Master).

git status – ע"י פקודה זו נוכל לראות איזה קבצים נערכו או איזה קבצים חדשים נוספו מהcommit הקודם– חדשים יצויינו כuntracked כאלו ששונו יצויינו כmodified.

אם עדין לא נעשה עליהם "add" (עברו לstaged) יסומנו באדום ,אם כן נעשה עליהם add יסומנו בירוק.

git log – ע"י פקודה זו נוכל לראות בכל branch את כל הcommitים שבוצעו בו ופרטים על כל commit.

git log --all --graph –decorate – יראה את הבראנצ'ים והקומיטים כעץ גרפי.

HEAD יצביע על הקומיט שעליו אנו עומדים.

קובץ gitignore. – קובץ שאנחנו יוצרים בתוך הrepo(התיקייה שבמעקב ע"י הgit שיצרנו) וכל שם של קובץ או תיקייה שנכתוב בתוכו יגרום להתעלמות הgit מהקובץ\תיקייה.

יצירת הקובץ touch .gitignore (הקובץ מוסתר לכן יש "." בתחילת שמו).

ע"י הוספת:

/a.txt

Test

נגרום לgit להתעלם מהקובץ a.txt ומהתיקייה Test.

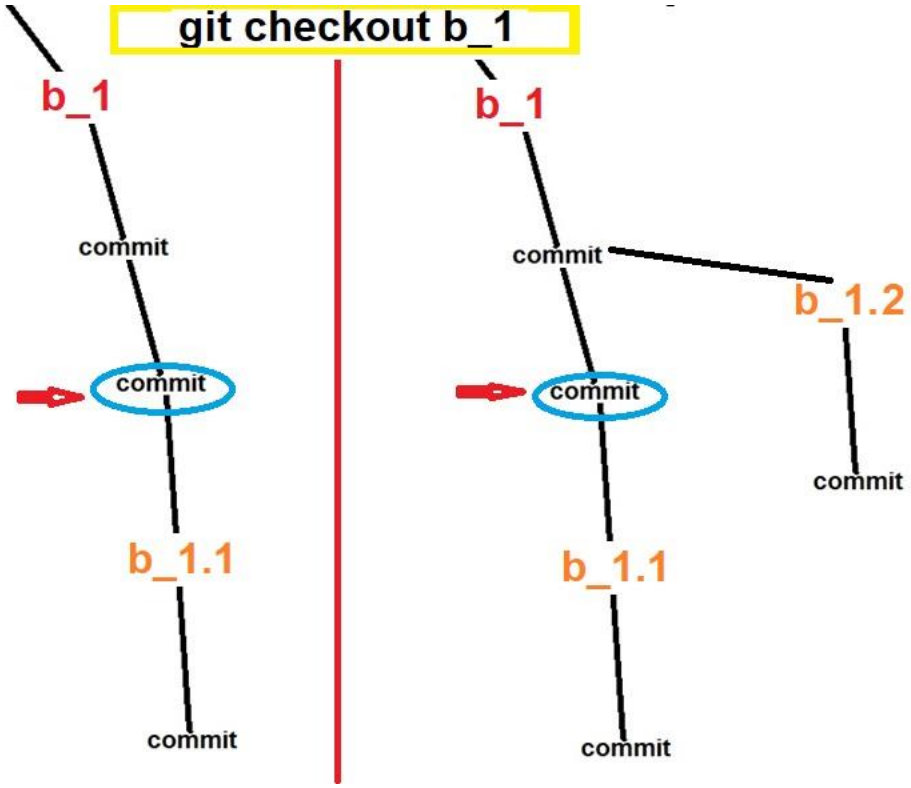
(ב/a.txt, ה-"/" מתייחס לrepo ולא לתיקיית הroot כיוון שמבחינת הgit הrepo היא התיקייה הראשונה בהיררכיה).

touch ~/.gitignore

git config --global core.excludesFile ~/.gitignore

Git checkout – לעבור לבראנצ' או לקומיט מסויים או ליצור בראנצ' חדש:

Git checkout b_1



יגרום לך לעבור לבראנצ' b_1 (כלומר לקומיט החדש ביותר על הבראנצ'

אם קיים בראנצ' שממשיך את b_1 לא יעבור אליו ויעמוד על הקומיט האחרון לפני יצירת בראנצ' זה)

כמו כן **git checkout master** יביא אותנו לקומיט האחרון של master לפני ההסתעפות האחרונה כמו כל בראנצ' אחר.

git checkout 8ae6324067f168566dcce4268c88ab24ba085eb9

יגרום לך לעמוד על **הקומיט**: 8ae6324067f168566dcce4268c88ab24ba085eb9

git checkout -b branchb – יצירת בראנצ' חדש החל מהקומיט עליו אתה עומד ומעבר (checkout) אליו.

Git merge b_1.11 – יעשה את פעולת המיזוג (merge) כאשר הבראנצ' b_1 ימוזג אל תוך הבראנצ' עליו אנו עומדים (מהקומיט עליו אנו עומדים)

אם יש קונפליקט(conflict) כלומר יהיה קובץ כלשהו ששורה אחת (או יותר) תיהיה

שונה בין שני הקומיטים שהולכים להתמזג השורה תשתנה לצורה הזו:

hello

aa

HEAD >>>>>>>

b_1.12

=======

11111

b_1.11 <<<<<<<

(השורה 111111 בקומיט שרוצים למזג בבראנצ' b_1.11 והשורה aa בקומיט מהבראנצ' שאנו עומדים עליו (HEAD = b_1.12)).

5853575e387e66c30834f8f117a527eb372b2d6e git reset –hard - תמחק את כל הקומיטים **עד** הcommit שכתוב בפקודה לא כולל.

הערה: אם עומדים על בראנצ' מסוים והקומיט המדובר בmaster ,הפקודה תמחק את כל הבראנצ' אך לא תמחק את הקומיטים מהמאסטר או מכל בראנצ' אחר (עד הקומיט הכי קרוב לקומיט המבוקש אך באותו בראנצ').

הערה: git reset –hard HEAD ימחק את כל השינויים שבוצעו מאז הקומיט האחרון לא כולל (יחזיר לקומיט שאנחנו עומדים עליו ללא השיויים שבוצעו).

Remote

Git clone – לשכפל את ה-git מהremote אל התיקייה שאתה עומד עליה.

בתיקייה זו תיפתח תיקיה בשם repo1 ובתוכה יהיה העתק של ה-repo מהremote והעתק של הgit.

לדוגמא:

git clone https://github.com/shaharhikri/repo1.git

Git push – "ידחוף" את הבראנצ' אל הgit המקורי בremote.

לא באמת ידחוף ישר אלא יפתח בקשה למיזוג(merge) הgit החל מבראצ' מסוים בתוך הgit בremote, בקשה זו נקראת pull request

משום שהremote עושה pull למה שאנו עשינו לו push.

```
git push --set-upstream origin name/of/my/branch
```

git config credential.helper store – זוכר את המשתמש והסיסמא של ה remote repo לצורך pull נוסף , לדוגמא:

$ git config credential.helper store

$ git push http://example.com/repo.git

<Username: <type your username

<Password: <type your password

[several days later]

$ git push http://example.com/repo.git

[your credentials are used automatically]

Git rebase – ...yyy ..asr sadf... ..git rebase fgdk – הפעולה תיקח את הבראנצ' ...yyy ואת כל הקומיטים (והבראנצים) הממשיכים אותו ותשים אותו על הקומיט ...asr

עוד חומרים על GIT:

Assume the following history exists and the current branch is "topic":

```
          A---B---C topic
         /
    D---E---F---G master
```

From this point, the result of either of the following commands:

```
git rebase master
git rebase master topic
```

would be:

```
                  A'--B'--C' topic
                 /
    D---E---F---G master
```

NOTE: The latter form is just a short-hand of `git checkout topic` followed by `git rebase master`. When rebase exits `topic` will remain the checked-out branch.
If the upstream branch already contains a change you have made (e.g., because you mailed a patch which was applied upstream), then that commit will be skipped. For example, running `git rebase master` on the following history (in which `A'` and `A` introduce the same set of changes, but have different committer information):

```
          A---B---C topic
         /
    D---E---A'---F master
```

will result in:

```
                   B'---C' topic
                  /
```

```
D---E---A'---F master
```

Here is how you would transplant a topic branch based on one branch to another, to pretend that you forked the topic branch from the latter branch, using `rebase --onto`. First let's assume your *topic* is based on branch *next*. For example, a feature developed in *topic* depends on some functionality which is found in *next*.

```
o---o---o---o---o   master
     \
      o---o---o---o---o   next
                       \
                        o---o---o   topic
```

We want to make *topic* forked from branch *master*; for example, because the functionality on which *topic*depends was merged into the more stable *master* branch. We want our tree to look like this:

```
o---o---o---o---o   master
    |               \
    |                o'--o'--o'   topic
     \
      o---o---o---o---o   next
```

We can get this using the following command:

```
git rebase --onto master next topic
```

Another example of --onto option is to rebase part of a branch. If we have the following situation:

```
                    H---I---J topicB
                   /
          E---F---G   topicA
         /
A---B---C---D   master
```

then the command

```
git rebase --onto master topicA topicB
```

would result in:

```
          H'--I'--J'   topicB
         /
         | E---F---G   topicA
         |/
A---B---C---D   master
```

This is useful when topicB does not depend on topicA.

A range of commits could also be removed with rebase. If we have the following situation:

```
E---F---G---H---I---J   topicA
```

then the command

```
git rebase --onto topicA~5 topicA~3 topicA
```

would result in the removal of commits F and G:

```
E---H'---I'---J'   topicA
```

This is useful if F and G were flawed in some way, or should not be part of topicA. Note that the argument to --onto and the <upstream> parameter can be any valid commit-ish.

In case of conflict, *git rebase* will stop at the first problematic commit and leave conflict markers in the tree. You can use *git diff* to locate the markers (<<<<<<) and make edits to resolve the conflict. For each file you edit, you need to tell Git that the conflict has been resolved, typically this would be done with

git diff [<options>] [<commit>] [--] [<path>…]

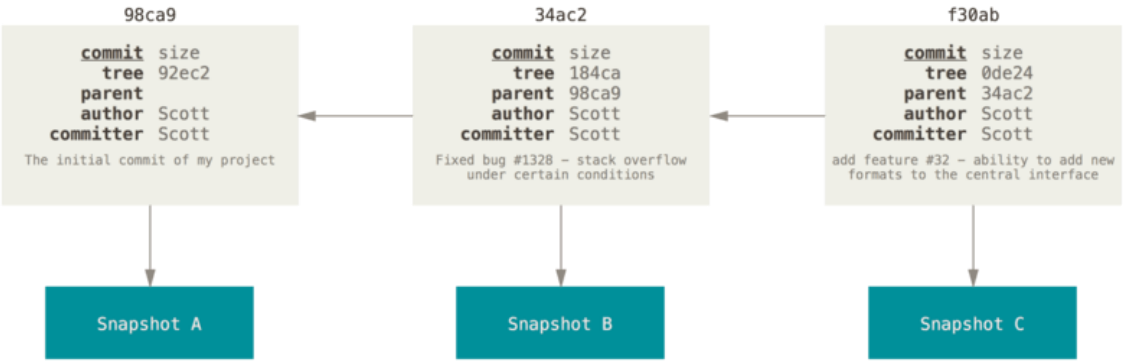
Show changes between the working tree and the index or a tree, changes between the index and a tree, changes between two trees, changes between two blob objects, or changes between two files on disk.

git rm --cached file.txt

git commit -m "file.txt removed from git"

rm file.txt

To remove a file from Git, you have to remove it from your tracked files (more accurately, remove it from your staging area) and then commit.



Oneline tree decorated:

git log --oneline –decorate

To synchronize your work with a given remote, you run a git fetch <remote> command (in our case, git fetch origin).

This command looks up which server “origin” is (in this case, it’s git.ourcompany.com), fetches any data from it that you don’t yet have, and updates your local database, moving your origin/master pointer to its new, more up-to-date position.

Pushing

When you want to share a branch with the world, you need to push it up to a remote to which you have write access.

Your local branches aren’t automatically synchronized to the remotes you write to — you have to explicitly push the branches you want to share.

If you have a branch named serverfix that you want to work on with others, you can push it up thesame way you pushed your first branch. Run git push <remote> <branch>

To set up a local branch with a different name than the remote branch, you can easily use the first version with a different local branch name.

$ git checkout -b sf origin/serverfix

Branch sf set up to track remote branch serverfix from origin.

Switched to a new branch 'sf'

Now, your local branch sf will automatically pull from origin/serverfix.

If you already have a local branch and want to set it to a remote branch you just pulled down, or want to change the upstream branch you’re tracking, you can use the -u option to git branch to explicitly set it at any time.

$ git branch -u origin/serverfix

Branch serverfix set up to track remote branch serverfix from origin.

Pulling

While the git fetch command will fetch all the changes on the server that you don’t have yet, it will not modify your working directory at all.

However, there is a command called git pull which is essentially a git fetch immediately followed by a git merge in most cases.

CREATE

Clone an existing repository

```
$ git clone ssh://user@domain.com/repo.git
```

Create a new local repository

```
$ git init
```

LOCAL CHANGES

Changed files in your working directory

```
$ git status
```

Changes to tracked files

```
$ git diff
```

Add all current changes to the next commit

```
$ git add .
```

Add some changes in <file> to the next commit

```
$ git add -p <file>
```

Commit all local changes in tracked files

```
$ git commit -a
```

Commit previously staged changes

```
$ git commit
```

Change the last commit

Don't amend published commits!

```
$ git commit --amend
```

COMMIT HISTORY

Show all commits, starting with newest

```
$ git log
```

Show changes over time for a specific file

```
$ git log -p <file>
```

Who changed what and when in <file>

```
$ git blame <file>
```

BRANCHES & TAGS

List all existing branches

```
$ git branch -av
```

Switch HEAD branch

```
$ git checkout <branch>
```

Create a new branch based on your current HEAD

```
$ git branch <new-branch>
```

Create a new tracking branch based on a remote branch

```
$ git checkout --track <remote/branch>
```

Delete a local branch

```
$ git branch -d <branch>
```

Mark the current commit with a tag

```
$ git tag <tag-name>
```

UPDATE & PUBLISH

List all currently configured remotes

```
$ git remote -v
```

Show information about a remote

```
$ git remote show <remote>
```

Add new remote repository, named <remote>

```
$ git remote add <shortname> <url>
```

Download all changes from <remote>, but don't integrate into HEAD

```
$ git fetch <remote>
```

Download changes and directly merge/integrate into HEAD

```
$ git pull <remote> <branch>
```

Publish local changes on a remote

```
$ git push <remote> <branch>
```

Delete a branch on the remote

```
$ git branch -dr <remote/branch>
```

Publish your tags

```
$ git push --tags
```

MERGE & REBASE

Merge <branch> into your current HEAD

```
$ git merge <branch>
```

Rebase your current HEAD onto <branch>

Don't rebase published commits!

```
$ git rebase <branch>
```

Abort a rebase

```
$ git rebase --abort
```

Continue a rebase after resolving conflicts

```
$ git rebase --continue
```

Use your configured merge tool to solve conflicts

```
$ git mergetool
```

Use your editor to manually solve conflicts and (after resolving) mark file as resolved

```
$ git add <resolved-file>
$ git rm <resolved-file>
```

UNDO

Discard all local changes in your working directory

```
$ git reset –hard HEAD
```

Discard local changes in a specific file

```
$ git checkout HEAD <file>
```

Revert a commit (by producing a new commit with contrary changes)

```
$ git revert <commit>
```

Reset your HEAD pointer to a previous commit …and discard all changes since then

```
$ git reset –hard <commit>
```

…and preserve all changes as unstaged changes

```
$ git reset <commit>
```

…and preserve uncommitted local changes

```
$ git reset –keep <commit>
```