LINUX
למתקדמים

טיפים, טריקים
ותכנות ב-Bash

אוהד מליחי

בס"ד

עיצוב והפקה: הוצאת הוד-עמי, הרצליה

עיצוב עטיפה: רוני סמט

נדפס בישראל 11/2016

הפצה: **הוצאת הוד-עמי**

09-9564716

info@hod-ami.co.il www.hod-ami.co.il

מסת"ב **978-965-361-423-9** ISBN

תוכן העניינים

הקדמה

מערכות ההפעלה לינוקס צוברות תאוצה רבה בשנים האחרונות מסיבות רבות: המודעות לעולם הקוד הפתוח (Open Source) החינמי, הנגיש והגמיש גוברת, לינוקס הרחיבה את תכונותיה ההיסטוריות מהיות שרת ללא יכולות גרפיות למערכת בעלת ממשקים גרפיים (GUI) מודרניים, תלת-ממדיים, מגוונים ומשוכללים שמושכים גם את המשתמש הביתי; היות שהמערכת גמישה מאוד מתבססים עליה ליצירת רכיבי תקשורת רבים ומגוונים מעבר לעולם המחשב הביתי כגון טאבלטים וסמארטפונים, נתבים, מתגים, חומות אש, סטרימרים, מפצלי עומסים ועוד (ולכן גם האנשים שמנהלים ומתחזקים מערכות אלה מתעניינים ביכולות המערכת); הפתיחות של המערכת מאפשרת לאנשי אבטחת מידע לבצע באמצעותה בדיקות חדירות ובדיקות אבטחת מידע נוספות ביתר קלות ; ארגונים רבים בעולם כולל ממשלות מנצלים את הכוח של קהילת הקוד הפתוח ומשתמשים במוצרי קוד פתוח לתועלתן ללא צורך בפיתוח ביתי או בהשקעה מיוחדת להנגשת שירותים מסוגים שונים כגון אתרי אינטרנט; מפתחים בכל הרמות ומכל הסוגים ממפתחי סקריפטים ועד מפתחי אפליקציות משתמשים במערכות ובתשתיות קוד פתוח ומנצלים אותן לחיסכון בקוד וליעילות פיתוח מירבית ועוד.

לכן נכון להיום פלח שוק מגוון למדיי של אנשי מחשבים וגם משתמשים ביתיים משתמשים במערכות לינוקס מסוגים שונים למטרות שונות רבות.

לספר זה 2 מטרות:

1. להציג את היכולות המתקדמות של המערכות האלו על ידי הצגת טיפים שימושיים למשתמשים מתקדמים או משתמשים שרוצים להכיר יותר לעומק את היכולות של מערכות אלה ובנוסף לעזור למנהלי מערכות לינוקס מסוגים שונים להשיג במהירות את מבוקשם מהמערכת על ידי עיון בטיפים המעשיים ומימושם באופן יחסית מיידי.
2. לתת סקירה מפורטת ומעשית שמלמדת פיתוח סקריפטים ב-Bash שהינו ה-SHELL הדיפולטיבי של לינוקס - שתסייע לאנשי מערכת לייצר לעצמם תהליכים יעילים ומהירים יותר ובכללם פתרונות אוטומציה במערכת.

מדובר על "מערכות" ברבים היות שבפועל קיימות הפצות רבות של מערכות דומות.

ברוב המקרים הפצה תהיה מבוססת על אחת מבין 2 המשפחות הגדולות של לינוקס שהן Red Hat ו-Debian, אך יש משפחות נוספות.

לכן כל טיפ בספר יתייחס בעיקר למשפחות אלו, אך בפועל יהיה ניתן ליישום בלא מעט מערכות נוספות.

לשם קלות ההבנה, הטיפים יתייחסו למערכות הספציפיות המוכרות CentOS ו-Ubuntu ראשית היות שכל אחת מהן מהווה דוגמא לאחת מהמשפחות לעיל (CentOS הינה ממשפחת Red Hat ו-Ubuntu הינה ממשפחת Debian), שנית היות שהן יותר מוכרות ופופולאריות מהשאר ושלישית כי שתיהן חינמיות.

לעתים הטיפ יהיה זהה אפילו ל-2 המערכות האלו למרות השוני ביניהן. במקרה כזה הכותרת של הטיפ תכיל את 2 השמות בשורה אחת. אחרת - אם יהיה שוני - תהיה שורה נפרדת לכל מערכת.

בנוסף, לאחרונה CentOS יצאה בגרסה חדשה - גרסה 7 - שהינה שונה מהותית מגרסה 6 שקדמה לה. אם טיפ מסוים ימומש באופן שונה בין 2 הגרסאות האלו השוני יצוין באותו טיפ.

כאמור הטיפים מתייחסים למשפחות שלמות של מערכות הפעלה ולא ל-2 מערכות ספציפיות בלבד.

כמו כן כמובן שאת רוב ככל הפקודות יש להריץ כ-ROOT או עם הרשאות ROOT (עם SUDO) גם אם זה לא צוין מפורשות.

ככלל, ספר זה נועד בעיקר למשתמשים שלא עובדים עם GUI ומתעסקים עם הטרמינל ועם סקריפטים ואוטומציה. מי שמעדיף את הטרמינל באופן כללי, יכול למצוא בספר זה כלים להבנה טובה יותר של המערכת וכפועל יוצא לנצל אותה (גם את ההבנה וגם את המערכת) לשימוש יעיל יותר. בנוסף, גם כיום ה-GUI בלינוקס הוא עדיין בסיסי מאוד ולא מתחיל להתקרב ליכולות שיש לשורת הפקודה ולכן יש פחות טעם להרחיב עליו. למרות זאת, קיימים בספר גם טיפים שמתייחסים ל-GUI כאשר יש להם פוטנציאל להיות שימושיים.

אנשים נוטים לחשוב שעולם הקוד הפתוח יכול להתחבר רק לעצמו. הם טועים טעות מרה שעשויה לעלות להם ביוקר. כל הרעיון בקוד פתוח הוא לאפשר את שימושו בכל פלטפורמה, כולל ווינדוס כמובן. הכנסת יכולות וכלים מעולם זה לתוך ווינדוס משדרגת לו משמעותית את שורת הפקודה ומוסיפה לו את היכולות הקלאסיות של לינוקס כגון יכולות חיפוש וסינון תוכן מתקדמות והפעלת תהליכים באופן אוטומטי. ניתן להשתמש ב-Bash ממש בתוך ווינדוס (דרך Cygwin למשל שהינה תשתית להפעלת רכיבי לינוקס בתוך וינדוס) ואף להתקין כלים קלאסיים אחרים כגון: wget, curl, awk, grep, find, file, sed, scp ועוד.

אפילו מיקרוסופט בעצמה רואה את הפוטנציאל הזה ואת העוצמה שצבר עולם הקוד הפתוח. היא החליטה לאחרונה לשלב אותו במוצריה כמה שיותר. דוגמא אחת לכך היא הוספת היכולת להפעיל מערכת לינוקסית שלמה מבוססת Ubuntu (בפועל מדובר בגרסה רגילה אמיתית של אובונטו!). הגרסה רצה ברמת הקרנל של ווינדוס בווינדוס 10 בגרסתו האחרונה.

יתכן שחלק מתוכן הספר יראה בסיסי יחסית ולא מתקדם במיוחד אך זאת משלוש סיבות עיקריות: האחת שלימות נתונים, השנייה הדרגתיות בהצגת שימושים מתקדמים לטובת הבנה טובה יותר והשלישית נובעת מכך שהספר מנסה לייצר מאגר נתונים שימושי של טיפים מעשיים ולא כל טיפ כזה הוא בהכרח מורכב. סיבות משניות הינן מתן אפשרות להיזכר בבסיס של פקודה מתקדמת במהירות מבלי להסתבך עם דוגמאות מסורבלות והצגת בסיס לבנייה קלה יותר של פקודות עתידיות באופן עצמאי.

יתר על כן, היות שאנשים מגיעים אל לינוקס מרקעים שונים וכיוונים שונים, כל אחד יכיר את המערכת מכיוון אחר ולכן ההגדרה של "מתקדמים" כאן היא לאו דווקא מקצוענים בתחום בעלי שנים של ניסיון אלא בעלי רקע מספיק להבנת התוכן כפי שהוא מוצג בספר. גם ככה באופן כללי בעולם הסיסטם, עקב היקפו הרב, איש סיסטם אחד תמיד יוכל לחדש לאיש סיסטם אחר ולהיפך ולא משנה כמה ניסיון יש לשניהם בתחום.

כולי תקווה שהספר אכן יגשים את מטרתו ויספק גישה מהירה לנתונים שימושיים יומיומיים.

באיחולי קריאה מהנה ופורייה,

המחבר

לשאלות, הערות, הארות או כל דבר אחר ניתן לשלוח מייל ל- ohad.malichi@gmail.com

מספר עקרונות כלליים לפני שמתחילים

1. **הכל זה קובץ** - בלינוקס, כל דבר הוא קובץ. מקבצים רגילים וקבצי הגדרות דרך התקני אחסון ועד רכיבי חומרה ומשאבי מערכת. חשוב להבין את זה כי המשמעות היא שהרבה פקודות שנראות לכאורה פשוטות ובסיסיות, יכולות בפועל לבצע שינויים מהותיים במערכת כמו למשל הפקודה echo שמדפיסה תווים, פעולה שנחשבת פשוטה מאוד לכאורה, יכולה לאפס הגדרות חומרה אם משתמשים בה לשלוח ערכים לקבצים המתאימים. דוגמא נוספת היא שימוש בעורך קבצים פשוט כגון VI על מנת להגדיר הגדרות של סרוויסים ושל מערכת ההפעלה עצמה. בנוסף יש לשים לב שמהכיוון ההפוך, זה די פשוט לבצע פעולות מסוכנות על המערכת ולכן יש להיזהר.

2. **כל פקודה היא עולם קטן** - לינוקס מכילה לא מעט כלים קטנים שלכאורה נראים פשוטים אבל בפועל כל אחד מהם הוא עולם ומלואו. הפקודה sed למשל, יודעת לבצע מניפולציות מורכבות מאוד על קבצי טקסט והפקודה awk שבדרך כלל משתמשים בה לחיתוך קל ונוח של טקסט היא למעשה שפת תכנות שלמה בפני עצמה.

3. **כלים קטנים יוצרים עולם גדול** - לינוקס מורכבת מהמון המון כלים קטנים (או פקודות) שמשלימים אחד את השני. ההסתכלות הזו חשובה כדי להבין שעל מנת לבצע פעולה כלשהי כל מה שצריך זה לחבר יחדיו את כל הכלים שמיישמים את הפעולה הרצויה ולזכור שבהכרח יש כלי שמתאים לכל חלק בפעולה. אם נרצה נניח למצוא אוסף קבצים לפי הגדרות מסוימות, אותם לסנן שוב לפי שם ואז לבצע עליהם פעולה מסוימת נוכל לשלב בין find, grep ו- xargs. באותו אופן ניתן לשלב הרבה יותר פקודות יחדיו ועם הרבה יותר גיוון (ראו דוגמאות בהמשך).

4. **לא לפחד ממורכבות** - לחידוד נוסף של הנקודה הקודמת, חשוב לזכור לנצל את המודולאריות הזאת של לינוקס ולדעת שיש לכל פעולה שנוכל לחשוב עליה - קטנה ובסיסית ככל שתהיה - או פקודה או כלי שיודעים לבצע אותה. בשילוב של מספר כלים כאלו במקביל (ואין לזה גבול) ניתן להשיג איזו תוצאה שרוצים. דוגמא לפקודה בסיסית שכזו היא הפקודה rev, שפשוט הופכת את התווים של כל שורת קלט שהיא מקבלת (קיצור של המילה REVERSE).

5. **הכל פתוח** - עם כל היכולות הקיימות במערכת, כאילו שזה לא מספיק, כל הקוד שממנו היא מורכבת פתוח ונגיש לשינויים. זה אומר שאם בכל זאת תהיה הרגשה שחסר משהו או שמשהו לא ממומש במאה אחוז כמו שאנחנו רוצים, גם אותו ניתן לעדכן או לשדרג או לשנות לפי הצורך הספציפי שלנו. זה לא קורה הרבה אבל העובדה שזה אפשרי וקיים עבורנו אם רק נרצה בכך, היא עוצמתית במיוחד כשלעצמה.

6. **גדלי אותיות** - לא לשכוח שלינוקס רגישה לגדלי אותיות. A זה לא a. אז אם הגעתם מווינדוס ללינוקס לאחרונה או שניסיתם להעתיק פקודה מהמייל והיא התחילה באות גדולה - תבינו למה היא לא עובדת. כל הפקודות בלינוקס הינן עם אותיות קטנות בלבד. צורה אחרת לא תעבוד. לאורך הספר פקודה עשויה להופיע גם באותיות גדולות - זאת

כאשר הפקודה תוזכר כמושג ולא כפשוטה - ולכן חשוב לזכור שכאשר רוצים להשתמש בפקודה כלשהי בפועל יש להריצה רק תוך שימוש באותיות קטנות.

7. **סוגי רישיונות** - למרות שאנו עוסקים בקוד פתוח שהגדרתו הקלאסית היא פתוח ולכן חינמי, בפועל המצב מעט מורכב יותר מכך. כל רכיב קוד פתוח משויך לרישיון מסוים שבו בחר המפתח. קיימים בשוק לא מעט סוגי רישיונות עבור מוצרי קוד פתוח ולכל אחד תנאים משלו. בתנאים אלו כלולים למשל שימוש חופשי לחלוטין כולל שינויים, איסור שימוש לצרכים מסחריים והסכמה להפוך את הקוד שלך גם לקוד פתוח כתנאי לשימוש במוצר. שמות של רישיונות לדוגמא הם: Apache License 2.0, GNU General Public License (GPL), GNU Library or "Lesser" General Public License (LGPL), Mozilla Public License 2.0 ו- Creative Commons (CC) license.

8. **ביטויים רגולריים** - פקודות לינוקס רבות מאפשרות את השימוש בביטויים רגולריים. ביטוי רגולרי הוא ביטוי כללי שמורכב משפה ואוסף של הגדרות והוא מאפשר להתייחס כביטוי בודד לאוסף שלם של ביטויים שחלים עליהם התנאים שהוגדרו בביטוי הרגולרי. במילים אחרות, ביטוי רגולרי בודד מגדיר שפה שלמה ובכל פעם שנציין אותו, זה יהיה שקול לכתיבת כל השפה כולה.

מדובר בהרבה יותר מסתם * או ? שמציינים בהתאמה "הכל" או "תו בודד כלשהו".

כל אוסף בעולם ניתן לתחום לביטוי רגולרי. אוספים כגון "כתובת IP בגרסה 4" או "כל המילים שמתחילות בתו A" ועוד ועוד.

דוגמאות לביטויים רגולריים פשוטים (יש ביטויים הרבה הרבה יותר מורכבים):

1. ^A.*Z$ - כל הביטויים שמתחילים (^) באות A גדולה ומסתיימים ($) באות Z גדולה.
2. *[0-9] - כל הביטויים שמכילים אפס או יותר פעמים את אחת הספרות 0-9.
3. \s.*end - כל ביטוי שמכיל רווח, אוסף של תווים (חוץ משורה חדשה) ואת המילה end.

יכולת מתקדמת זו, מאפשרת בקלות (יחסית...לעתים לוקח זמן לבנות את הביטוי הנכון) להשיג כל פלט שנרצה לפי איזו תבנית שנצטרך (אותה יגדיר הביטוי הרגולרי שיצרנו) ולכן חשוב להכירה ולנצלה.

חשוב גם לדעת שביטוי רגולרי נמצא בשימוש בהקשרים נוספים ולא רק בתוך לינוקס. לדוגמא, עורך הטקסט החינמי Notepad++ (שרץ על ווינדוס) מאפשר לבצע חיפושים והחלפות טקסט גם באמצעות ביטויים רגולריים דבר שמקל מאוד לפעמים. בנוסף, גם כל שפות התכנות מאפשרות את השימוש בהם.

לאורך כל הספר ניתן למצוא דוגמאות לביטויים רגולריים ולאופן השימוש בהם.

בנוסף, יש ברשת מסמך בשם "Regx Cheat Sheet" עם רשימה מסודרת ונוחה של כל הכללים וההגדרות שמאפשרים ליצור ביטויים רגולריים (המסמך לא מופיע בספר מכיוון שהגדרות הרישיון שלו לא מאפשרות זאת).

היכרות לעומק עם ביטויים רגולריים הינה מחוץ לטווח ההכרה של ספר זה אך מפאת החשיבות של הנושא הוא הוזכר כאן בקצרה.

9. **<u>דינאמיות</u>** - לינוקס הינה מערכת הפעלה דינאמית. בעת עלייתה, היא מזהה בזמן אמת את רכיבי החומרה שיש במחשב שטען אותה ומגדירה אותם לשימוש על ידי טעינת הדרייברים המתאימים. זה אומר שלא משנה על איזה מחשב נריץ אותה, היא תמיד תפעל גם אם התקנו אותה על מחשב עם חומרה מסוג אחד ולקחנו אותה **לאחר התקנה** למחשב עם חומרה מסוג אחר. זאת בניגוד למערכת ההפעלה ווינדוס למשל שלאחר ההתקנה מכילה רק את הדרייברים של החומרה הנוכחית שעליה היא הותקנה ולכן היא לא תפעל על חומרה אחרת לאחר התקנה. יתר על כן, ניתן גם לאחר עליית המערכת לטעון מודולים אל תוך הקרנל על מנת להפעיל תכונות נוספות שלא הופעלו כברירת מחדל.

10. **<u>מודולאריות</u>** - לינוקס הינה מערכת הפעלה מודולארית. ניתן לחלק את המשמעות של תכונה זו לשניים:

1. חלקים קטנים - לינוקס הינה מערכת המורכבת מאוסף גדול של חבילות קטנות. מה שלא צריך לאותו מקרה לא יוגדר. לכן אם לא הצלחנו לבצע משהו זה לא אומר שאין את היכולת אלא שהיא פשוט לא הותקנה עדיין ויש להתקינה כי כברירת מחדל בד"כ אין בה צורך. תכונה זו מאפשרת למערכת להיות רזה ומהירה כי באמצעותה נוכל לדאוג שרק יכולות שנעשה בהן שימוש יותקנו. העובדה שקוד המערכת עצמו כולו פתוח מרחיבה עוד יותר את תכונת המודולאריות של לינוקס ומאפשרת לנו לבחור אפילו ביתר דיוק את הרכיבים שנשתמש בהם בכל רמה שהיא - חומרה ותוכנה גם יחד.

2. חיבור בין הקרנל לשירותי המערכת - ניתן להגדיר את הקרנל של המערכת ואת המערכת עצמה כשני מודולים נפרדים. אם הקרנל לא תקין, ניתן להשתמש בקרנל אחר כדי להפעיל את המערכת ולא חייבים להשתמש בגרסה ספציפית אחת. כך ניתן למשל להפעיל את המערכת על פלטפורמות שונות על ידי שינוי מתאים של גרסת הקרנל שיאפשר להפעיל את המערכת על הפלטפורמה החדשה מבלי לשנות כלום בשירותים או ביישומים המותקנים על המערכת ובהגדרותיהם.

11. **<u>שורה חדשה בלינוקס</u>** - לינוקס מגדירה שורה חדשה על ידי תו המערכת LF (\n) - תו בודד. לעומתה ווינדוס מגדירה שורה חדשה על ידי זוג תווי המערכת - CR (\r) ו- LF (\n) יחדיו.

צריך להכיר את העניין הזה כי יש פקודות לינוקס שלא יודעות להציג נכון קבצים שנוצרו בווינדוס ואז נראה בקובץ תווים לא ברורים כגון "M^" כי מבחינת לינוקס יש כאן תו מערכת מיותר שלא צריך להופיע והתוצאה היא קובץ לא קריא וחמור מכך - אם נשתמש בו בתהליך כלשהו כגון אם נקרא אותו למשל באמצעות סקריפט, הסקריפט עשוי להיכשל רק בגלל התווים המיותרים בסוף השורה. מהצד השני, פתיחת קובץ בפורמט לינוקסי בווינדוס בתוכנה שלא יודעת להבחין בו (כגון NOTEPAD) תגרום

להצגת כל התוכן בשורה אחת בלבד כי התוכנה לא תזהה את הסימן לשורה חדשה בשום מקום (היא תמצא רק תו אחד במקום שניים) וגם זה לא קריא במיוחד.

את הבעיה הזו ניתן לפתור די בקלות על ידי המרת הקובץ לפורמט לינוקסי או בצד של ווינדוס או בצד של לינוקס.

בצד של ווינדוס ניתן להשתמש בעורך טקסט מתקדם כגון Notepad++ ודומיו שמאפשרים כאופציה להמיר קבצים מפורמט ווינדוס לפורמט לינוקס ולהיפך.

בצד של לינוקס, קיימות 2 פקודות עבור המרה זו. הן נקראות unix2dos ו-dos2unix בהתאמה (מדובר בפקודות ותיקות והיסטורית השוני היה במקור בין UNIX ל-DOS ולכן הפקודות נקראות בשם זה).

כך נוכל לעבוד עם קבצים שהגיעו מווינדוס ללינוקס ללא בעיות וכן להיפך.

הכינויים של התווים האלה נקבעו כפי שהם גם מסיבה היסטורית - הם נקראו על שם הפעולות שהתבצעו במכונת הכתיבה המכנית הישנה בעת מעבר לשורה חדשה - CR זה קיצור של Carriage Return (זו ההחזרה של החלק הנע במכונה חזרה למיקומו לאחר סיום כתיבה של שורה) ו-LF זה קיצור של Line Feed (זו ההזנה של שורה ריקה חדשה על ידי הזזת הדף בתוך המכונה).

12. **ESCAPE של תווים** - משמעות המושג ESCAPE של תווים הוא לגרום להדפסת תו שבמצב רגיל יש לו משמעות מיוחדת בתוך המסגרת שבה אנו נמצאים כעת כתו ליטרלי כלומר כפי שהוא ממש כטקסט.

היכולת הזו תקפה גם לרמת פקודה ולא רק לרמת תו. במקרה של פקודה הכוונה היא להפעלת הפקודה במצב המקורי הקלאסי שלה.

בהקשר של תווים, זהו מושג חשוב היות שלא תמיד נרצה שתו בעל תפקיד מיוחד יעובד במסגרת הנוכחית שלנו אלא נרצה לשלוח אותו למסגרת פנימית אחרת כי המסגרת הפנימית היא זו שצריכה לעבד אותו ובמקרה זה נרצה שהוא יכנס אל המסגרת החיצונית כתו טקסט פשוט.

המילה "מסגרת" בהקשר הזה מתייחסת ל-Bash, לפקודה אחרת, לשפה אחרת או לכל תשתית אחרת שניתן לעבוד אתה ולאו דווקא רק בלינוקס.

בהקשר של פקודה, נרצה להפעיל לעתים את הפקודה במצב קלאסי ולפעמים במצב מותאם אישית אך נרצה ששני המצבים יהיו נגישים לנו במקביל בכל זמן נתון. ההבחנה בין 2 המצבים מתבצעת בזכות ESCAPE.

כדי לגרום לתו מיוחד להיות מודפס כפי שהוא, עלינו פשוט לקרוא לו עם תו ה- ESCAPE.

כדי לגרום לפקודה לרוץ במצבה הקלאסי, נעשה בדיוק את אותו הדבר.

בנוסף ובאופן הפוך, תו ה-ESCAPE משמש גם ליצירת ביטוי מיוחד מתו שאין לו בדרך כלל משמעות מיוחדת.

תו ה-ESCAPE הינו התו סלאש הפוך - " \ " (איך עושים ESCAPE לתו ה-ESCAPE? ....)

<u>דוגמאות:</u>

1. שימוש במשתנים של awk כדי לחתוך פלט של פקודה מורכבת - ESCAPE של התו $:

גם הפקודה awk וגם Bash משתמשים בתו $ כדי לציין שמדובר במשתנה.

עם פקודה פשוטה של awk אין שום בעיה היות שנעשה בה שימוש בזוג גרשיים בודדים ובתוכה מופיע הביטוי עם ה- $. הגרשיים הבודדים גורמים להצגת כל תו כפי שהוא.

העניין מסתבך עם פקודה מורכבת שמצריכה אותנו להשתמש בגרש הבודד לטובת פישוט הפקודה. במקרה כזה ניאלץ להשתמש בזוג תווי גרשיים כפולים כדי להריץ את awk, ואז אם ננסה להגדיר ביטוי עם התו $ בתוך הפקודה awk, Bash יעבד אותו בתור משתנה כי הוא המסגרת הראשונה שלנו ואז מה שיכנס לפקודה יהיה לאחר העיבוד של הביטוי על ידי Bash או במילים אחרות תוכן של משתנה והפקודה awk לא תעבוד כפי שרצינו או שהיא תיכשל לגמרי.

על מנת להעביר ל-awk את הביטוי כדי שהוא יהיה זה שיעבד את המשתנה, נצטרך להשתמש בתו ה- ESCAPE כדי ש-Bash יזהה אותו כטקסט פשוט.

נניח שהגדרנו את:

```
ARP_BIN=/sbin/arp
awk_BIN=/bin/awk
```

ואז:

<u>לפני ESCAPE:</u>

פקודה מורכבת:

```
IPS=$ARP_BIN'  -n  | '$awk_BIN' " {if (index($3, \":\"))  print  $1}" '
echo  $IPS | sh
```

פלט:

```
awk: {if (index(, ":")) print }
awk:          ^ syntax error
awk: fatal: 0 is invalid as number of arguments for index
```

עקב העובדה שלא השתמשנו בתו ה- ESCAPE, Bash המיר את הביטויים $1 ו-$3 למשתנים ריקים שערכם כלום. לכן awk מתלוננת כאן על קריאה לא נכונה לפונקציה (חסרים לה נתונים...).

אחרי ESCAPE:

פקודה מורכבת:

```
IPS=$ARP_BIN'  -n  | '$awk_BIN' "{if (index(\$3, \":\")) print  \$1}" '
echo  $IPS | sh
```

פלט:

```
192.168.40.254
192.168.40.2
```

כפי שניתן לראות הפקודה שהרצנו שתפקידה להציג את רשימת כתובות ה-IP שקיימות בטבלת ה- ARP הנוכחית של המערכת עבדה כראוי.

2. sed של ביטוי - ESCAPE של התו נקודה:

לתו נקודה (.) ישנה משמעות מיוחדת עבור הפקודה sed. פירושו "תו בודד כלשהו".

לכן אם נרצה להתייחס אליו כפי שהוא כנקודה ממש בתוך sed, נצטרך להשתמש בתו ה-ESCAPE לשם כך.

לפני ESCAPE:

פקודה:

```
echo "google.com" | sed "s/./ dot  /g"
```

פלט:

```
dot  dot  dot  dot  dot  dot  dot  dot  dot  dot
```

ללא ESCAPE, פקודת ההחלפה שהגדרנו באמצעות sed שתפקידה להחליף את התו "." במילה dot, תחליף **כל** תו במילה הזו ולכן יצא לנו הפלט לעיל.

אחרי ESCAPE:

פקודה:

```
echo "google.com" | sed "s/\./ dot  /g"
```

פלט:

```
google dot com
```

ניתן לראות שעם ESCAPE נקבל את מה שרצינו - החלפה אחת בלבד של התו ".".

3. התו רווח:

אם קיימים קבצים שיש בשמות שלהם רווחים, על מנת לעבוד איתם יש צורך בשימוש בתו ESCAPE עבור התו רווח, במיוחד עם השלמתם האוטומטית

באמצעות התו TAB. מנגנון ההשלמה האוטומטית ייעצר במקרה שבו יש יותר מקובץ אחד במיקום הנוכחי עם שם דומה (כרגיל) ועל מנת לבחור דווקא את הקובץ עם הרווח נדרש להקליד את תו ה-ESCAPE סלאש הפוך כאשר המנגנון נעצר על רווח. רק לאחר מכן מנגנון ההשלמה ימשיך לעבוד ולהשלים את שם הקובץ עם הרווח. זה פחות קריטי במצבים אחרים היות שניתן להתחמק מהבעיה על ידי שימוש בגרשיים כדי לסמן את ההתחלה והסוף של שם הקובץ.

חשוב לציין שאם מדובר בקובץ בודד עם רווחים, מנגנון ההשלמה ישלים אותו ללא קושי ויוסיף סלאשים לפני כל רווח בעצמו היות שמדובר בקובץ יחיד ואין במקרה זה את הבעיה של להבין איזה קובץ המשתמש רוצה לבחור.

4. ESCAPE של פקודה:

כידוע, ניתן להשתמש בפקודה alias כדי ליצור שם קצר לפקודה. אם הגדרנו alias מותאם אישית לפקודה קיימת באותו השם ונרצה להריץ את הפקודה בגרסתה המקורית, נוכל להשתמש בתו ה- ESCAPE.

אם נגדיר למשל את:

```
alias ls="ls  -al"
```

נקבל:

<u>לפני ESCAPE:</u>

פקודה:

```
ls
```

פלט:

```
total 8
drwxrwxr-x. 2 ohadm ohadm 4096 Jul  7 10:29 .
drwxr-xr-x. 3 ohadm ohadm 4096 Jul  6 13:53 ..
-rw-rw-r--. 1 ohadm ohadm    0 Jul  7 10:29 file1
-rw-rw-r--. 1 ohadm ohadm    0 Jul  7 10:29 file2
-rw-rw-r--. 1 ohadm ohadm    0 Jul  7 10:29 file3
-rw-rw-r--. 1 ohadm ohadm    0 Jul  7 10:29 .hiddenFile1
```

<u>אחרי ESCAPE:</u>

פקודה:

```
\ls
```

פלט:

```
file1  file2  file3
```

כלומר לפני ה- ESCAPE של הפקודה קיבלנו את הפלט לפי ה-ALIAS שהגדרנו שכולל הדפסת פרטים אודות הקבצים והצגת קבצים נסתרים ולאחריו את הפלט הדיפולטיבי של הרצה רגילה של ls.

5. יחס הפוך - תו מיוחד לשורה חדשה - ESCAPE של התו n קטנה:

הדפסת האות n קטנה באופן רגיל תגרום להדפסת האות כפי שהיא אך אם נבצע עליה ESCAPE עם סלאש הפוך, נקבל תו מיוחד שפירושו "הדפס את התו שמגדיר שורה חדשה".

<u>לפני ESCAPE:</u>

פקודה:

```
echo  "line1 n line2"
```

פלט:

```
line1 n line2
```

<u>אחרי ESCAPE:</u>

פקודה:

```
echo  -e  "line1 \n line2"
```

פלט:

```
line1
line2
```

<u>הערות:</u>

פקודת ה- echo השנייה משתמשת במתג e– שמגדיר לה להתייחס לתווים מיוחדים. בלעדיו השורה תודפס ללא שורה חדשה אך זה לא קשור ל- ESCAPE אלא לאופן הפעולה של הפקודה echo.

טיפים וטריקים

כלי מערכת

חלק זה מנסה להציג את העוצמה שיש לשילוב בין מספר כלי מערכת שונים כדי לקבל איזו תוצאה שנרצה.

הוא מציג פקודות שימושיות יומיומיות.

כל הפקודות בו קיימות ב-2 המשפחות של מערכות ההפעלה ולכן אין צורך לציין זאת עבור כל פקודה בנפרד.

יחד עם זאת יש לזכור להריץ כמנהל (root או sudo) פקודות שצריכות הרשאות גבוהות.

du - הצגת גדלים של קבצים ותיקיות

הפקודה du משמשת להצגת גדלים של תיקיות וקבצים במערכת לפי חיתוכים שונים.

דוגמאות שימוש:

הצגת גדלים של תיקיות וקבצים מוסתרים

הפקודה:

```
du -h .??* | sort -h
```

הערות:

- שני סימני השאלה **חיוניים** על מנת לדלג על הביטוי "..." שפירושו "כל התוכן של התיקייה הקודמת". סימן שאלה פירושו "תו בודד כלשהוא".
- -h HUMAN READABLE - מציג / ממיין גדלים קריאים בקילו, מגה, ג'יגה וכדומה במקום ב- BYTES (נכון עבור 2 הפקודות גם יחד).

הצגה ברורה של גדלי קבצים ותיקיות

הפקודה:

```
du -hs * | sort -hr
```

הערות:

- -s סיכום - מציג נתונים ברמת ה- LEVEL הנוכחי.
- -r REVERSE - הצג גדלים מהגדול לקטן.

הצגת גדלים של תיקיות בלבד

הפקודה:

```
du -hs ./*/
```

הערות:

- ./*/ הפקודה du מציגה נתיבים אל קבצים ותיקיות כמיקומים יחסיים שמתחילים בזוג התווים נקודה וסלאש. לכן נוכל לתחום ביניהם לבין סלאש נוסף את התו * וכך בעקיפין נקבל ביטוי רגולרי שמתייחס רק לתיקיות כי סלאש נוסף יופיע רק עבור תיקיות.

הצגת גדלים תוך דילוג על תיקיות מסוימות או קבצים מסוימים

הפקודה:

```
du -hs --exclude={proc,usr,tmp,dev} *
```

הערות:

- --exclude={proc,usr,tmp,dev} רשימת התיקיות שלא יבדקו. הפורמט צריך להיות כמו בדוגמא - רשימה תחומה בין סוגריים מסולסלים ופסיקים בין ביטוי לביטוי.

cut - חיתוך נתונים

הפקודה cut חותכת נתונים לשדות ומאפשרת באופן זה להדפיס רק שדות רצויים.

דוגמאות שימוש:

הצגת כתובות ה- IP בלבד על ידי חיתוך פלט

הפקודה:

```
ip addr | grep inet | cut -d' ' -f6
```

הערות:

- -d הגדרת מחלק (DELIMITER). בדוגמא זו לתו רווח. הפקודה תחלק את הפלט לפי רווחים.
- -f הדפס את שדה (FIELD) מספר 6. בדוגמא זו השדה השישי מכיל את כתובות ה- IP לאחר הדפסת 4 רווחים ריקים, והחמישי מפריד בין המילה שמגדירה את הסוג של הכתובת לבין הכתובת עצמה.

grep - חיתוך נתונים מתקדם

הפקודה אולי המוכרת ביותר שמתקשרת לעולם של לינוקס.

תפקידה למצוא נתונים בפשטות וללא מאמץ לפי חיתוכים ותנאים שונים.

דוגמאות שימוש:

חיתוך פלט על ידי הורדת שורות לא רצויות (הסתכלות הפוכה - במקום בחירת הרצויות)

נניח שיש לנו את הקובץ הבא במערכת:

```
$ cat  list.txt
one
two
three
four
```

הפקודה:

```
cat  list.txt | grep  -v  three
```

פלט:

```
one
two
four
```

הערות:

- -v חיתוך הפלט על ידי מחיקת הביטוי שבא אחרי מתג זה מרשימת התוצאות.

חיתוך פלט לפי מספר מילים במקביל

נסתכל שנית על אותו קובץ:

```
$ cat list.txt
one
two
three
four
```

הפקודה:

```
cat  list.txt | grep  'one\|two'
```

פלט:

```
one
two
```

הערות:

- חשוב להכיר את היכולת הזו למקרה שבו נרצה לסנן פלט מפקודה לפי מספר מילים במקביל.

- חשוב להקפיד על כך שהביטוי **לא** יכיל רווחים אחרת הפקודה לא תעבוד.
- התו | (PIPE) פירושו "או" וחובה ל- ESCAPE אותו עם התו \ (סלאש הפוך) אחרת הפקודה לא תעבוד.

חיפוש רקורסיבי של טקסט

יכולת זו של GREP הינה מאוד שימושית ומאוד נוחה למציאת נתונים בקלות ובמהירות בתוך אוסף של קבצים עם תוכן מבלבל או דומה.

יתר על כן, רוב המערכת מוגדרת באמצעות הגדרות שנמצאות בקבצי טקסט ולכן ניתן באמצעות פקודה זו למצוא הגדרת מערכת בקלות גם אם איננו זוכרים היכן היא נמצאת.

הפקודה:

```
grep -i -R "<ביטוי לחיפוש>" .
```

הערות:

- -i אל תבחין בין גדלי אותיות (case insensitive).
- -R חיפוש רקורסיבי - חפש בתוך כל הקבצים את הביטוי לחיפוש החל מהמיקום הנוכחי (.).
- "<ביטוי לחיפוש>" הביטוי שנרצה לחפש.
- . חפש החל מהמיקום הנוכחי. כמובן שניתן לציין גם נתיב מלא.

חיפוש תווים מיוחדים

הפקודה:

```
grep -P "[\t]*" <קובץ>
```

הערות:

- -P grep לא מחפשת תווים מיוחדים כברירת מחדל. המתג -P מרחיב ל- grep את היכולות ומוסיף לה תמיכה בביטויים רגולריים ותווים מיוחדים נוספים שהשפה PERL תומכת בהם (PERL=P).
- *[\t] חפש את התו TAB אפס פעמים או יותר.

התעלמות מתיקייה מסוימת

הפקודה:

```
grep --exclude-dir=myDir
```

הערות:

- --exclude-dir=myDir התיקייה שהגדרנו ל- GREP להתעלם ממנה.

התעלמות מתיקייה מסוימת תוך כדי העתקת קבצים

בדוגמא זו אנו משתמשים ב-grep כדי לדלג על תת-תיקייה תוך כדי העתקה של תוכן של תיקייה רצויה. על מנת ליצור רשימה של קבצים ותיקיות רצויים, אנו מריצים את הפקודה ls בשילוב עם grep –v וכך הרשימה מצטמצמת לפי הדרישה (ניתן להוסיף כמה פקודות grep שנרצה כמובן או להשתמש בביטוי רגולרי במקום בשם בודד) ולאחר שיש לנו את הרשימה הרצויה אנו שולחים אותה לפקודה cp שמעתיקה רק אותה.

הפקודה:

```
cp -R `ls /dir/to/copy | grep -v "inner_dir_to_exclude"` \
 destination
```

תזכורת!

התפקיד של הסלאש ההפוך בסוף השורה הראשונה הוא לומר ל-Bash שהפקודה טרם הסתיימה ויש לה המשך. אם נעתיק את כל הפקודה כמו שהיא ל-Bash היא תעבוד כרגיל בזכות הסלאש למרות שהיא מחולקת ל-2 שורות נפרדות. חשוב לזכור גם בהקשר הזה שאסור שיהיו רווחים בין הסלאש לבין סוף השורה (כלומר הסלאש הבודד צריך להופיע אחרון בשורה).

הערות:

- שימו לב לגרשיים ההפוכים שמאפשרים את הרצת הפקודה ls בתוך הפקודה cp.

ריבוי שורות ב- GREP

כברירת מחדל, grep עובדת על שורות. על מנת להגדיר לה להתעלם מסוף שורה ניתן להשתמש במתג z-. מתג זה יגרום ל-grep להסתכל על תוכן כבלוק אחד גדול.

הפקודה:

```
grep -z <קובץ>
```

שליפת חלק מקובץ לפי שורה תוחמת

דוגמא זו מציגה דרך לשלוף חלק מקובץ לפי שורה שחוזרת על עצמה בתוך הקובץ ומפרידה בין מקטע למקטע. בדוגמא המקטע שנשלף הוא האחרון בקובץ.

טכנית, זה מה שהפקודה עושה:

1. הפקודה grep מחפשת את השורה שמחלקת את הקובץ למקטעים ומדפיסה אותה עם המתג n– שמדפיס לצד השורה גם את מספרה.
2. הפקודה מניחה שהשורה המחלקת בנויה מ- 3 שורות. שורת כותרת עליונה, שורת תיאור ושורת כותרת תחתונה. לכן היא יכולה להיות למשל:

==========

תיאור

==========

<u>או:</u>

תאריך

הפקודה שולפת את המקטע האחרון בקובץ כולל החלק הזה על ידי שימוש בפקודות tail ו-head.

1. לאחר מכן הפקודה awk שולפת רק את מספר השורה מהפלט של grep.
2. בסוף, הפקודה xargs שולחת ל-awk את מספר השורה ו- awk מדפיסה את כל השורות החל משורה זו עד סוף הקובץ.

דוגמא זו מציגה גם את הכוח שיש בשילוב בין מספר פקודות.

<u>הפקודה:</u>

```
grep -n "patternToFind" <קובץ> | tail -2 | head -1 \
| awk -F: '{print $1}' | xargs -I {} awk 'NR>={}' <קובץ>
```

<u>הערות:</u>

- `-n` הדפס את מספר השורה לצד השורה שנמצאה.
- `"patternToFind"` שורת הכותרת שנמצאת בקובץ. זו השורה המחלקת. היא יכולה להיות למשל: ========== .
- `awk -F: '{print $1}'` שליפת מספר השורה מהפלט של grep (לפרטים נוספים אודות פקודה זו ראו בהמשך בחלק של awk).
- `xargs -I {} awk 'NR>={}'` שליחת מספר השורה באמצעות הפקודה xargs להרצה נוספת של awk שמדפיסה את השורות החל ממספר השורה שנמצאה על ידי grep ועד לסוף הקובץ. הביטוי NR מתייחס למספר השורה הנוכחי ב-awk (לפרטים נוספים אודות הביטוי NR והפקודה xargs ראו בהמשך).

awk - חיתוך נתונים לפי שדות (עמודות)

הפקודה awk מאוד משוכללת ובעלת כוח רב. בפועל ניתן לומר כי היא שפת תכנות שלמה ולא רק פקודה.

המקרה הקלאסי של שימוש בפקודה הוא חיתוך נתונים והדפסת חלקם על ידי בחירת עמודות רצויות.

הפקודה awk מתייחסת אל התוכן שהיא עובדת עליו כטבלה עם עמודות כאשר העמודה הראשונה נקראת $1, השנייה $2 וכן הלאה.

העמודות מתחלקות לפי מחלק (DELIMITER) שהינו התו רווח כברירת מחדל. ניתן לשנות את המחלק בקלות על ידי שימוש במתג F- **גדולה** ובצמוד לה את המחלק שנרצה. למשל: נקרא ל-awk עם המתג "F\- כדי להגדיר מחלק שהינו התו גרשיים כפולים (") או :F- על מנת להגדיר מחלק שהוא התו נקודתיים (:). ניתן לשנות את המחלק לכל תו וכך מאוד נוח לקבל את התוכן שרוצים על ידי משחקים עם הגדרה זו.

הביטוי $0 מתייחס לכל התוכן שנכנס אל awk.

הביטוי NF מכיל את מספר השדות הכולל שיש בשורת הפלט הנוכחית (זהו נתון שמתעדכן ברמת שורה).

הביטוי NR מכיל את מספר השורה הנוכחית.

מבנה כללי של הפקודה:

```
awk '{ סט של פקודות }' <קובץ>
```

הפקודות שנשלחות ל-awk חייבות להופיע תחומות בתוך שילוב של גרשיים **בודדים** (בהכרח) וסוגריים מסולסלים.

<u>דוגמאות שימוש:</u>

שליפת פלט חלקי 1 - הדפסת שם המשתמש וה-SHELL מתוך קובץ המערכת /etc/passwd

פקודה זו מדגימה שימוש קלאסי בפקודה awk.

<u>הפקודה:</u>

```
cat /etc/passwd | awk -F: '{print $1" "$7}'
```

<u>הערות:</u>

- :F- כברירת מחדל, awk מחלקת את הנתונים שמגיעים אליה לשדות לפי רווחים. אם רוצים להגדיר לה מחלק אחר, משתמשים במתג F- גדולה ורושמים בצמוד לו את התו שרוצים לחלק אתו. בדוגמא אנו עובדים עם קובץ המשתמשים שהמחלק בו הוא התו נקודתיים (:) ולא התו רווח ולכן עלינו להגדירו כמחלק על מנת ש-awk יחלק את הקובץ לפי השדות הנכונים.
- print $1" "$7 הדפס את השדות 1 ו- 7. במקרה שלנו מדובר בשם המשתמש וב-SHELL שהוגדר עבור אותו משתמש. print היא פונקציית ההדפסה של awk וצריך את

הרווח באמצע כדי ש-print תדפיס את השדות עם רווח ביניהם, אחרת הנתונים המודפסים יהיו צמודים ולא קריאים.

שליפת פלט חלקי 2 - בדרך הפוכה ("הכל חוץ מ-") 1

דוגמא זו מראה איך ניתן באמצעות awk לקבל חלק מטבלה בקלות.

<u>הפקודה:</u>

```
awk  '{$1=$2="" ;  print  $0}'  myFile.txt
```

<u>הערות:</u>

- `$1=$2=""` איפוס המשתנים שמייצגים את העמודות שאיננו רוצים. כך יצא שנקבל מ-awk את עמודות 3 והלאה ללא עמודות 1 ו- 2 שאותן איפסנו עם פקודה זו שמכניסה "כלום" למשתנים שמייצגים אותן.
- `;` התו נקודה פסיק מפריד בין פקודות כאשר רוצים להריץ בתוך awk יותר מפקודה אחת.
- `print  $0` כפי שהוזכר לעיל, הביטוי $0 מייצג את כל הקלט שנכנס אל awk. לאחר שאיפסנו את עמודות 1 ו- 2 נקבל את הכל חוץ מהן וזה מה שרצינו.

שליפת פלט חלקי 3 - בדרך הפוכה ("הכל חוץ מ-") 2

בדוגמא הקודמת איפסנו השדות שלא רצינו לכלול באופן ידני היות והן היו מועטות. אם מדובר בהרבה שדות יותר נוח להשתמש במנגנון אוטומטי. הדוגמא הבאה מדגימה זאת. היא לוקחת את שדה מספר 9 והלאה מהפלט של הפקודה ls –l. במקום לאפס ידנית את שדות 1-8, נשתמש בלולאת for בתוך awk. לא מספיק לקחת את שדה מספר 9 לבדו היות שאם מדובר בשמות קבצים עם רווחים, הרי ש- awk תגדיר את החלקים הנוספים של השם כשדות נפרדים ואז השם יהיה חלקי.

<u>הפקודה:</u>

```
ls  -l  |  awk  '{ s = ""; for (i = 9; i <= NF; i++)  s = s $i " "; print  s }'
```

<u>הערות:</u>

- `s = "";` הגדרת משתנה זמני בשם s ואיפוסו.
- `for (i = 9; i <= NF; i++)  s = s $i " ";` לולאת for. הלולאה רצה עם המשתנה i מהמספר 9 שמייצג את תחילת הטווח שנרצה לשלוף ועד הערך של משתנה המערכת NF שמייצג את מספר השדות הכולל שיש בשורה הנוכחית. הלולאה משרשרת למשתנה s את נתוני השדות לפי הטווח שבחרנו תוך שימוש בתו רווח להפרדת השדות.
- `print  s` בסוף הלולאה אנו מדפיסים את התוכן של המשתנה s שמכיל את הנתונים החלקיים שרצינו.

מחיקת קבצים לפי תאריך

דוגמא זו מרחיבה את הדוגמא הקודמת ומשלבת את grep ואת xargs. הדוגמא מציגה דרך למחוק קבצים לפי תאריך ספציפי.

הרעיון הכללי של דוגמא זו הוא להראות דרך להגיע אל קובץ באופן עקיף. כך אם למשל יש לנו קובץ בשפה שהשרת לא מכיר (שהועלה דרך אתר אינטרנט למשל) ונרצה לעשות אתו משהו, לא נוכל להשתמש במקלדת או בהשלמה אוטומטית במקרה זה (בהנחה שאין לנו ממשק גרפי) ונצטרך דרך עקיפה כגון זו כדי להגיע אל הקובץ. כמובן שניתן באופן דומה להגדיר קריטריונים אחרים נוספים מלבד התאריך של הקובץ על מנת למצוא אותו.

<u>הפקודה:</u>

```
ls  -l | grep  "Jul\ 10" | awk  '{ filename = ""; for (i = 9; i <= NF; i++) \
if(i==NF){filename = filename $i} else {filename = filename $i " "}; \
print  "\""filename"\"" }' | xargs  -I {} rm {}
```

<u>הערות:</u>

- `grep  "Jul\ 10"` חיתוך הפלט של ls לפי תאריך רצוי.
- `awk  '{.....}'` החלק של awk שולף את שם הקובץ מהפלט של הפקודה ls ומדפיס אותו בין גרשיים כפולים. החלק של הלולאה זהה לדוגמא הקודמת ותפקידו למצוא גם שמות קבצים בעלי רווחים (לפרטים נוספים ראו לעיל). כאן יש תוספת של תיחום השם לאחר מציאתו בין גרשיים כפולים אחרת הפקודה rm תיכשל עבור שמות קבצים שיש בהם רווחים.
- `for (i = 9; i <= NF; i++)` הלולאה זהה ללולאה מהדוגמא הקודמת. ראו שם פרטים נוספים.
- `if(i==NF){filename = filename $i} else {filename = filename $i " "};` תנאי שמונע שרשור של התו רווח כאשר הגענו לשדה האחרון (NF). התנאי הזה נדרש כי בלעדיו יהיה שרשור של תו רווח מיותר בסוף שם הקובץ ואז הפקודה rm שבהמשך תיכשל. את התו רווח עלינו להוסיף על מנת להפריד בין השדות כאשר אנו משרשרים אותם אחד לשני (אחרת כל הנתונים יופיעו מחוברים ברצף).
- `print  "\""filename"\""` הדפסת שמות הקבצים שנמצאו בין גרשיים כפולים. יש כאן נתון שנשלף ממשתנה בשילוב עם נתון שהוא מחרוזת פשוטה. המשתנה הוא filename ולכן הוא מחוץ לגרשיים המגדירים מחרוזת. היות שאנו רוצים לשרשר לו את התו גרשיים כפולים (") אנו משתמשים כאן בתו ה- ESCAPE \ (סלאש הפוך) כדי להתייחס אליו כתו טקסטואלי ואם הוא נחשב כטקסט הרי שעלינו לתחום אותו עצמו בגרשיים וכך נקבל את הביטוי "\"".
- `xargs  -I {} rm {}` מחיקת הקבצים שנמצאו (להסבר מפורט עיין בפקודה xargs).

הרצת פקודה על מספר ערכים במקביל באמצעות awk

פקודה זו מדגימה את היכולת של awk לשכפל שורות. כך ניתן בקלות להריץ פקודה על מספר ערכים במקביל על ידי שליחת השורות המשוכפלות אל Bash להרצה.

כאן ניתן לראות דוגמא להרצת הפקודה הנוחה והשימושית file, שמדפיסה תיאור של סוג הקובץ הנוכחי. הפקודה מתבצעת על כל האובייקטים בתיקייה הנוכחית ולמעשה מתבצע כאן שכפול של אותה פקודה מספר פעמים באמצעות awk עבור כל אובייקט שנמצא בתיקייה. כל השורות נשלחות לבסוף ל-Bash שמריץ אותן אחת אחת.

<u>הפקודה:</u>

```
ls | awk '{print "file " $1 " " }' | sh
```

<u>הערות:</u>

- כמו בדוגמא הקודמת, גם כאן נדרשת ומתבצעת הפרדה בין נתון טקסטואלי (המילה "file" והתו רווח) לבין מילה שמגדירה פעולה ב- awk (כגון הפקודה print והערך $1). ההפרדה מתבצעת עם גרשיים כפולים בשילוב עם רווחים שמפרידים בין הביטויים השונים.

פלט לדוגמא:

```
gnupg: directory
group: ASCII text
group-: regular file, no read permission
grub.conf: symbolic link to `../boot/grub/grub.conf'
gshadow: regular file, no read permission
gshadow-: regular file, no read permission
gtk-2.0: directory
hal: directory
host.conf: ASCII text
hp: directory
httpd: directory
```

יכולים לנחש איזו תיקייה זו?

שליפת נתונים מקובץ לפי טווח זמנים - דוגמא 1

פקודה זו מדגימה כיצד ניתן לשלוף מקובץ שבנוי בפורמט קבוע רשומות לפי טווח תאריכים ספציפי.

<u>הפקודה:</u>

```
awk '$4>"[05/May/2014:08:50:" && \
 $4<"[05/May/2014:11:05:" ' /var/log/httpd/access_log
```

התאמה לפורמט נוסף:

```
awk '$4>"12:21:33" && \
 $4<"12:21:45" ' /var/log/httpd/access_log
```

תזכורת!

התפקיד של הסלאש ההפוך בסוף השורה הראשונה הוא לומר ל-Bash שהפקודה טרם הסתיימה ויש לה המשך. אם נעתיק את כל הפקודה כמו שהיא ל-Bash היא תעבוד כרגיל בזכות הסלאש למרות שהיא מחולקת ל- 2 שורות נפרדות. חשוב לזכור גם בהקשר הזה שאסור שיהיו רווחים בין הסלאש לבין סוף השורה (כלומר הסלאש הבודד צריך להופיע אחרון בשורה).

הערות:

- על מנת שהפקודה תעבוד, חייבים להתאים אותה למבנה התוכן בקובץ.
- החלק הגאוני בפקודה הוא ש- awk מתעלמת מהחלק הטקסטואלי / תווים שאינם מספרים ויודעת לשלוף את החלק הרצוי על ידי עבודה רק על החלק המספרי וחישוב של הפרש המספרים שנבחר.
- $4 זה השדה שמכיל את השעה ולכן עליו מתבצע התנאי שבודק את הטווחים הרצויים.

שליפת נתונים מקובץ לפי טווח זמנים - דוגמא 2

גם כאן מתבצעת שליפת נתונים לפי טווח באופן מעט שונה עם שימוש ב- grep לחתוך מקדים של תוכן של קובץ ועם תנאי בתוך awk שדואג להדפיס רק הנתונים שנמצאים בטווח שהוגדר.

הפקודה:

```
grep support@org.il maillog |     \
awk '{ if ($3>"08:00:00" && $3<"08:10:00") print $0}'
```

הרצת פקודת מערכת לאחר חיתוך נתונים עם awk

הפקודה:

```
ls -ltr | awk '{if ($7==5) system("cp " $9 " /tmp")}'
```

הפקודה במילים:

אם מספר היום בחודש בתאריך של קובץ בתיקייה הנוכחית שווה ל-5, העתק אותו לתיקייה הזמנית של המערכת.

הערות:

- $7 השדה שמכיל את מספר היום בחודש לאחר הרצת הפקודה ls -l.
- $9 השדה שמכיל את שם הקובץ לאחר הרצת הפקודה ls -l.
- system פונקציית מערכת שמאפשרת להריץ פקודות מערכת מתוך awk.

- "cp " $9 " /tmp" ביחס ל-awk, הפקודה שנשלחת ל-system היא מחרוזת למעט השדה של שם הקובץ ולכן צריך לחלק את השורה לתת-מחרוזות על ידי תווי גרשיים כפולים על מנת להפריד בין נתון של awk לבין טקסט שנשלח על ידה הלאה.

xargs - הרצת פקודה על נתונים שהגיעו מפקודה אחרת

הפקודה xargs הינה שימושית למדי, פשוטה וגאונית, ומקילה מאוד על העבודה השוטפת.

תפקידה לקבל קלט כלשהו מכל מקור ולהריץ עליו פקודה אחרת.

הקלט מפורק על ידי xargs לשורה אחרי שורה ועל כל שורה בנפרד xargs מריצה את הפקודה הרצויה.

יש 2 אופציות להריץ את הפקודה. באופן כללי הדרך השנייה נדרשת רק אם הפקודה שרוצים להריץ על הקלט לא מסתיימת בו (כלומר - לא מכילה ארגומנטים נוספים מעבר לקלט הבסיסי שנכנס לפקודה xargs).

<u>מבנה כללי של הפקודה:</u>

```
קלט | xargs command
```

<u>או</u>

```
קלט | xargs -I <ביטוי מייצג> command <שימוש בביטוי>
```

ביטוי מייצג = הגדרת ביטוי מייצג עבור שורת הקלט הנוכחית כמו למשל # או {}. לאחר ההגדרה, בכל פעם שנכתוב אותו הוא יתורגם לשורה הנוכחית מהקלט שנכנס ל- xargs.

שימוש בביטוי = שימוש בביטוי לציון השורה הנוכחית בתוך הפקודה עצמה. כל ציון של הביטוי בשלב זה יתורגם לשורה הנוכחית מהקלט וכך למעשה תיבנה לנו הפקודה הרצויה עבור כל אחת משורות הקלט.

<u>דוגמאות שימוש:</u>

הסרת התקנה של מספר חבילות במקביל לפי שם (Red Hat)

<u>הפקודה:</u>

```
rpm -qa | grep php | xargs yum remove -y
```

<u>הערות:</u>

- כאן השימוש הוא במבנה הפשוט של הפקודה היות שהקלט של xargs יכול להישלח ל-yum בסוף הרצף של הפקודה ללא צורך במיקום מיוחד עבורו.

  זה אומר שבפועל רשימת החבילות שהפקודה תחזיר עד החלק של xargs תישלח שורה אחרי שורה על ידי xargs לחלק שהיא מריצה שהינו פקודת ההסרה yum remove והשליחה של הרשימה תתבצע באופן שקוף ומרומז ולא מפורשות (להשוואה ראו את הדוגמאות הבאות).

הצגת גדלים של תיקיות שנמצאו על ידי הפקודה find

הפקודה:

```
find . -type d -name "styles" | grep "files/styles" | \
xargs du -hs | sort -h
```

הערות:

- גם כאן מתבצע שימוש במבנה הקלאסי הפשוט של xargs מאותה הסיבה. הפקודה du מסתיימת בערכים ש- xargs שולחת אליה.

מציאת קבצים בני יותר מ-30 יום

הפקודה:

```
ls | xargs -I {} find {} -mtime +30
```

הערות:

- `xargs -I {}` הגדרת תו אחד (או יותר) שיגדיר ל- xargs באיזה שלב להכניס את השורה הנוכחית מהפלט שנכנס אליה. בדוגמא זו הגדרנו את צמד התווים {} כביטוי המייצג את השורה הנוכחית.
- `find {}` שימוש בתו שהוגדר לעיל בפועל בתוך הפקודה find. אם הוגדר תו אחר, כמובן שהוא צריך להופיע כאן במקום התווים {}.
- כאן, בניגוד לדוגמאות הקודמות, עלינו להגדיר באופן מפורש על ידי המתג I- וביטוי (כפי שהוסבר לעיל) באיזה שלב להשתמש בערכים ש-xargs מעבירה לפקודה הבאה היות שהפקודה הבאה לא מסתיימת בהם אלא משתמשת בהם באמצעה.

שינוי שם למספר קבצים במקביל

הפקודה:

```
ls /etc/yum.repos.d | xargs -I {} mv {} {}.dis
```

הערות:

- `xargs -I {}` כמו בדוגמא לעיל, הגדרת תו (או יותר) שיגדיר ל- xargs באיזה שלב להכניס את השורה הנוכחית מהפלט שנכנס אליה.
- `mv {} {}.dis` שימוש כפול בתו שהוגדר ל- xargs על מנת לאפשר את הרצת הפקודה mv לטובת שינוי השם של הקובץ הנוכחי.

העתקת מספר קבצים במקביל

הפקודה:

```
ls | grep .*17-8.* | xargs -I {} cp {} /dest
```

מציאת קובץ מכווץ לפי שם קובץ שנמצא בתוכו מתוך רשימה של קבצים מכווצים

הפקודה:

```
ls -t *.gz | xargs -I {} tar -tvf {} | grep "a file name"
```

ריבוי פקודות עם xargs

בדוגמא הקודמת, אם נמצא את הקובץ שחיפשנו באחד מבין הקבצים המכווצים שחיפשנו בהם, התוצאה שתופיע היא רק שם הקובץ שמצאנו.

זו תוצאה חסרה כי כל המהות של פקודה מסוג כזה היא להחזיר לנו בתוצאות גם או אפילו רק את שם הקובץ המכווץ שמכיל את הקובץ כי זהו הקובץ שבפועל חיפשנו. נוכל לתקן את התוצאה על ידי הוספת פקודת echo של שם הקובץ שאנו מחפשים בו כעת ואז אם ימצא הקובץ הפנימי נדע גם איזה קובץ הכיל אותו. לשם כך נצטרך להריץ 2 פקודות בתוך xargs.

על מנת לשמור על הקשר בין הביטוי המייצג ש- xargs משתמשת בו לציון האיבר הבא שיש לבצע עליו פקודה לבין 2 הפקודות, נתחם בתוך פקודת sh בין גרשיים בודדים את פקודת ה- echo ואת פקודת ה- tar גם יחד בין סימן נקודה פסיק (;), כך:

```
ls -t *.gz | xargs -I {} sh -c 'echo {}: ; tar -tvf {} | \
grep "a file name" '
```

הצגת פרטים של קבצים לפי שמות שנמצאים ברשימה שנמצאת בקובץ

הפקודה:

```
<list.txt | xargs -I {} ls -l {}
```

הערות:

- <list.txt שליחת הקובץ list.txt לפקודה xargs. זו הדרך היותר נכונה לשלוח קובץ לפקודה (בהשוואה ל- 'cat list.txt' - ראו הסבר על כך בהמשך הספר תחת החלק על טיפים ל-Bash).

tr - החלפת תווים

הפקודה tr מאפשרת להחליף או למחוק תווים בקלות.

דוגמאות שימוש:

המרת פסיקים לשורות חדשות

הפקודה:

```
cat csv-list.txt | tr ',' '\n' > newlines-list.txt
```

הערות:

- ',' חפש את התו הזה.
- '\n' החלף בתו הזה.

מחיקת התו "

הפקודה:

```
echo the_sign_"_will_be_removed | tr -d "\""
```

הערות:

- `-d "\""` מחק את התו "

find - חיפוש של קבצים ותיקיות

find היא פקודה מאוד חזקה ומשוכללת שניתן לעשות איתה הרבה.

התפקיד המרכזי שלה הוא למצוא קבצים אך כפי שניתן לראות מהדוגמאות להלן היא עושה קצת יותר מזה ...

כברירת מחדל, find מחפשת החל מהמיקום שהוגדר לה ופנימה באופן רקורסיבי.

ניתן להגדיר קריטריונים מסוגים שונים ומגוונים לחיפוש כפי שניתן לראות מהדוגמאות בהמשך.

מבנה כללי של הפקודה:

```
find <חפש החל ממיקום זה> <אפשרויות>
```

דוגמאות שימוש:

חיפוש לפי שם (case insensitive)

הפקודה:

```
find . -iname "*.TXT"
```

הערות:

- `"*.TXT"` חובה לכתוב ביטוי רגולרי בין זוג גרשיים כפולים אחרת הפקודה לא תעבוד כמצופה.
- `-iname` חיפוש לפי שם וללא התחשבות בגודל אות (ניתן להשתמש ב- -name על מנת לחפש עם התחשבות בגדלי אותיות).

חפש קבצי json שהשתנו ב-10 דקות האחרונות

הפקודה:

```
find . -name "*.json" -mmin -10
```

הערות:

- `-mmin -10` חפש קבצים בני פחות מ-10 דקות.
- `-mmin 10` חפש קבצים בני 10 דקות בדיוק.
- `-mmin +10` חפש קבצים בני יותר מ-10 דקות.

חפש קבצי GZIP בני יותר מ 30 יום

הפקודה:

```
find . -name "*.gz" -mtime +30
```

הערות:

- -mtime -30 חפש קבצים בני פחות מ 30 יום.
- -mtime 30 חפש קבצים בני 30 יום בדיוק.
- -mtime +30 חפש קבצים בני יותר מ 30 יום.

מציאת כל התיקיות המוגדרות עם הרשאות מלאות

הפקודה:

```
find / -perm 777 -type d
```

הערות:

- -type סוג האובייקט שמחפשים (d=directory, f=file).
- -perm איזה הרשאות לחפש.

מציאת כל הקבצים בגודל 0

הפקודה:

```
find / -size 0 -type f
```

הערות:

- -type סוג האובייקט שמחפשים (d=directory, f=file).
- -size חפש קבצים בגודל זה.

מציאת כל הקבצים בתיקייה tmp/ שבבעלות ROOT

הפקודה:

```
find /tmp -user root
```

הערות:

- -user חפש קבצים בבעלות משתמש זה.

חיפוש קבצים תוך הגבלת רמות העומק של התיקיות שיש לחפש בהן

הפקודה:

```
find / -mindepth 2 -maxdepth 4 -name index.html
```

הערות:

- -mindepth 2 חפש החל מרמת עומק שנייה ביחס למיקום שממנו החיפוש מתחיל (/ בדוגמא). אם החיפוש יתחיל מתת-תיקייה היא תוגדר כתיקיית השורש ביחס לחיפוש הנוכחי. בפועל רמה שנייה זה אומר קובץ בתוך תיקייה אחת ולא 2 תיקיות ואז הקובץ (למשל: var/index.html/ זה רמה שנייה כי רמה ראשונה זה קובץ במיקום /).
- -maxdepth 4 חפש במיקומים בעלי רמת עומק של מקסימום 4 כלומר במיקום של עד 3 תתי-תיקיות.
- -name index.html חפש קובץ בשם index.html. רצוי להגדיר את השם או הביטוי שרוצים לחפש אחרי המתגים של העומק לעיל אחרת find תדפיס אזהרה.

חיפוש קבצים לפי שם והפעלת פקודה על הקבצים שנמצאו

הפקודה:

```
find . -name ".htaccess" -exec ls -lh {} \;
```

הערות:

- -name חפש קבצים בעלי השם .htaccess .
- -exec הרץ עליהם את הפקודה הבאה (ls –lh במקרה הזה).
- {} ביטוי זה מומר לשם הקובץ הנוכחי שנמצא.
- \; החלק של exec- חייב להסתיים בתו נקודה פסיק. היות שגם Bash מתייחס לתו זה באופן מיוחד (ניתן באמצעותו להריץ מספר פקודות במקביל בשורה אחת) נדרש ל-ESCAPE אותו עם סלאש הפוך כדי שמי שיתייחס אליו יהיה find ולא Bash.

חיפוש זהה לחיפוש הקודם אך כולל גם סינון של הפלט, מיונו ועצירה כל מסך

הפקודה:

```
find . -name ".htaccess" -exec ls -lh {} \; \
| awk '{print $5" "$9}' | sort -hr | more
```

הערות:

- awk '{print $5" "$9}' הדפסת העמודה החמישית והעמודה התשיעית של הפלט (במקרה הזה של ls -lh) עם רווח ביניהן. בפועל מדובר בשם הקובץ שנמצא וגודלו. הרווח באמצע (" ") נדרש על מנת שיודפס רווח בין 2 השדות האלה.
- sort -hr מיון הפלט בסדר הפוך מהגדול לקטן (r = reverse).
- more עצור והמתן לאישור להמשך כשהמסך מתמלא.

חיפוש בתוכן של קבצים שנמצאו

הפקודה:

```
find /etc -name "*.conf" -print -exec grep -i hash {} \;
```

הערות:

- "*.conf" חובה לכתוב ביטוי רגולרי בין זוג גרשיים כפולים אחרת הפקודה לא תעבוד כמצופה.
- -exec grep -i hash חפש בתוך הקבצים שנמצאו את המילה "hash" תוך התעלמות מגדלי אותיות (-i).
- -print הדפס את שם הקובץ שנמצא. ללא מתג זה יודפסו למסך במקרה זה שורות שמכילות את המילה "hash" מתוך כל הקבצים שחיפשנו בהם ללא שמותם ואז לא נוכל לדעת איזו שורה שייכת לאיזה קובץ. החיסרון כאן הוא שהמתג הזה יגרום להדפסת כל הקבצים ש- find עברה עליהם ונצטרך לחפש מתחת לאיזה שם קובץ יש גם תוכן כדי לדעת באילו קבצים נמצאה המילה.

חיפוש קבצים תוך דילוג על רשימת תיקיות

הפקודה:

```
find / -path "/home" -prune -or -path "/backup" -prune \
-or -path "/usr" -prune -or -path "/etc" -prune -or -name \
"*.conf" -print
```

הערות:

- / חפש בכל מערכת הקבצים החל מתיקיית השורש.
- \ הפקודה ממשיכה בשורה הבאה.
- "*.conf" -name חפש את כל הקבצים בעלי סיומת conf. חובה לכתוב ביטוי רגולרי בין זוג גרשיים כפולים אחרת הפקודה לא תעבוד כמצופה.
- -print הדפס את שם הקובץ שנמצא. במקרה זה המתג חשוב כי בלעדיו find תדפיס גם את רשימת התיקיות שהגדרנו לדלג עליהן.
- -path "/home" -prune הגדרת תיקייה שרוצים לדלג עליה בחיפוש. חשוב לציין את שם התיקייה ללא הסלאש בסוף אחרת find תתעלם מהגדרה זו ***ולא*** תדלג על התיקייה (שימו לב לזה כשאתם משתמשים ב- TAB להשלמה אוטו' כי הוא כן מוסיף אותו). המתג prune- מגדיר את הדילוג. כפי שניתן לראות בדוגמא, מוגדרת כך רשימה של מספר תיקיות.

- -or ניתן להשתמש גם בביטוי o- הזהה. כאשר משתמשים בפקודת find עם בקשה לדילוג, צריך להפריד את הביטויים בפקודה שמגדירים באמצעותם את מה לחפש כגון name- ו- prune- עם מתג זה אחרת find לא תעבוד כמצופה.

הרצת מספר פקודות על תוצאות חיפוש

הפקודה:

```
find . -name page.htm -exec echo -e "{}:\n" \; \
-exec grep "my.js" {} \;
```

הערות:

- כפי שניתן לראות בדוגמא זו, ניתן לשרשר מספר פקודות עם המתג exec- בפקודת find בודדת. שימו לב שכל פקודה עם exec- בפני עצמה צריכה להסתיים בנפרד עם הביטוי הסוגר הרגיל שלה \;.

שינוי שם למספר קבצים במכה אחת

הפקודה:

```
find . -name "*.TXT" -exec bash -c 'mv $0 ${0/TXT/txt}' {} \;
```

הפקודה במילים:

חפש קבצים עם סיומת TXT באותיות גדולות, ושנה לקבצים שנמצאו את הסיומת ל- txt באותיות קטנות.

הערות:

- -exec sh -c הפעלת פקודה בחלון Bash נפרד. find מתקשה בהרצת פקודות מורכבות באופן ישיר ולכן ניתן להורות לה על ידי המתג exec– להריץ שוב את Bash ובתוכו נוכל להריץ איזו פקודה שנרצה. הפקודה בתוך ה- Bash הפנימי צריכה להימצא בין גרשיים בודדים (sh –c 'command').
- $0 שם הקובץ הנוכחי כמו ש- Bash מכיר אותו.
- {} שם הקובץ הנוכחי כמו ש- find מכירה אותו.
- ${0/TXT/txt} ביטוי שהינו בפועל פקודה של החלפת מחרוזות. הביטוי מחפש בתוך משתנה בשם $0 את המחרוזת TXT ואם הוא מוצא אותה הוא מחליף אותה במחרוזת txt ושומר את המשתנה עם השינוי שלאחר ההחלפה.
- \; החלק של exec- חייב להסתיים בתו נקודה פסיק. היות שגם Bash מתייחס לתו זה באופן מיוחד (ניתן באמצעותו להריץ מספר פקודות במקביל בשורה אחת), עלינו להשתמש בתו ה- ESCAPE סלאש הפוך כדי שמי שיתייחס אליו יהיה find ולא Bash.

הרצת פקודה מורכבת על קבצים שנמצאו בחיפוש

הפקודה:

```
DATE='date +%Y-%m-%d_%H-%M-%S'
export  DATE

find  .  -name  settings.php*?  -exec  sh  -c  'mv {} \
`get-backup-folder.sh`/$(basename  {})-`$DATE`' \;
```

הפקודה במילים:

חפש קבצים בעלי שם עם תו אחד לפחות אחרי התווים "settings.php". אם נמצא קובץ כזה, הזז אותו לתיקייה שמיקומה יתקבל כתוצאה מהרצת סקריפט ושנה את שמו לשמו הנוכחי בתוספת תאריך שמוגדר במשתנה הסביבה DATE (שהוגדר לעיל לפני הרצת החיפוש).

הערות:

- `-name  settings.php*?` חפש קבצים עם שם שמכיל **לפחות תו אחד נוסף** (זה מה שהסימן ? אומר) לאחר התווים settings.php (כלומר השם settings.php לא יכלל בחיפוש אך שמות כגון settings.php.1 ו- settings.php.bak כן).
- `-exec sh -c` הפעלת פקודה בחלון Bash נפרד. find מתקשה בהרצת פקודות מורכבות באופן ישיר ולכן ניתן להורות לה על ידי המתג exec– להריץ שוב את Bash ובתוכו נוכל להריץ איזו פקודה שנרצה. הפקודה בתוך ה- Bash הפנימי צריכה להימצא בין גרשיים בודדים (sh –c 'command').
- `` `get-backup-folder.sh` `` הפעלת סקריפט בתוך פקודת ה- Bash שבתוך החיפוש. הסקריפט צריך להימצא בין גרשיים הפוכים על מנת שירוץ תוך כדי התהליך.
- `$(basename  {})` שימוש בשם הקובץ ללא הנתיב שלו (על ידי הרצת הפקודה basename). הביטוי {} מכיל את שם הקובץ הנוכחי. הקלדת פקודה בין סוגריים ו- $ כמו בדוגמא, היא עוד דרך להריץ פקודה פנימית על מנת להשתמש בפלט שלה לפני שהפקודה החיצונית מסתיימת כמו הגרשיים ההפוכים לעיל.
- `` `$DATE` `` שימוש בתוכן של משתנה סביבה כחלק מפקודת החיפוש. גם כאן על מנת לקבל את תוכן המשתנה ולא את שמו נדרש לתחום אותו בין גרשיים הפוכים.

sed - ביצוע שינויים ועדכונים בקבצי טקסט

sed הינה פקודה מאוד חשובה ומאוד שימושית. היא מאפשרת לבצע בקלות שינויים ועדכונים בקבצי טקסט.

השימוש הקלאסי והבסיסי שלה, הוא find and replace ויש דוגמא לכך בהמשך.

sed עובדת ברמת שורה. היא מסתכלת כל פעם על שורה אחת.

בנוסף יש נספח בסוף הספר המפרט את כל סוגי הפעולות ש- sed יודעת לבצע.

דוגמאות שימוש:

חיפוש והחלפה

הפקודה:

```
sed -i "s/FindThisLine/ReplaceItWithThisLine/g" <קובץ>
```

למשל:

```
sed -i "s/127.0.0.1/127.1.1.1/g" /etc/hosts
```

הערות:

- -i in-place - עדכן את הקובץ בפועל. כברירת מחדל, הפקודה sed רק מדפיסה למסך את השינוי שיתבצע. על מנת לשמור את השינוי בקובץ, יש להשתמש במתג זה. לחילופין ניתן לבצע REDIRECT רגיל באמצעות הסימן גדול מ- (למשל כך: sed file1 > file2) אל קובץ בשם אחר שייווצר (אך לא אל אותו הקובץ - זה **לא** יעבוד). לכן רצוי להוסיף את המתג הזה רק לאחר שווידאנו שהפקודה עובדת כמו שרצינו.
- s - substitute - בצע פעולה מסוג החלפה.
- /find/replace/ תוכן הפקודה בנוי משני ביטויים התחומים בין שלושה סלאשים. הסלאש הראשון מגדיר את תחילת הפקודה. לאחריו מופיע ביטוי ראשון (רגיל או רגולרי) ש- sed יחפש עבורנו (המילה find בדוגמא). הוא נסגר עם סלאש שני. לאחר הסלאש השני מופיע הביטוי השני ש- sed יחליף עם הביטוי הראשון (את המילה replace עם המילה find בדוגמא). בסוף יש את הסלאש השלישי שסוגר את הפקודה.
- g - greedy - חמדני - בצע את ההחלפה בכל פעם שהביטוי לחיפוש נמצא. ללא הגדרה זו תתבצע החלפה במציאה ראשונה בלבד (החלפה אחת לכל שורה).

חיפוש והחלפה של שורה שלמה

הפקודה:

```
sed -i --follow-symlinks "/FindALineWithThisExpression/c
        ReplaceItEntirelyWithThisLine" <קובץ>
```

הערות:

- --follow-symlinks אם מדובר בלינק אל קובץ, עבוד על הקובץ ולא על הלינק. אפשרות זו חשובה מאוד אם מריצים sed על לינק ולא על קובץ כי בלעדיה הלינק יתחלף בקובץ חדש עם השינויים במקום הקובץ שהלינק מפנה אליו (ואז נישאר עם קובץ במקום לינק וגם הקובץ שרצינו שיתעדכן לא ישתנה - בעיה שעלולה להיות מסוכנת אם אנחנו בונים על הלינק הזה).

- c - range_change - התייחס לשורה שלמה ולא לביטוי. אם ימצא הביטוי שחיפשנו באותה שורה, כל השורה תתחלף בביטוי שהגדרנו להחלפה ולא רק החלק בשורה שמכיל את הביטוי שנמצא.
- /find/c replace במקרה זה בניגוד לדוגמא הקודמת, המבנה קצת משתנה. כאן יש סלאש ראשון, שורה לחיפוש, פקודה שמגדירה את הביטוי כשורה (c), רווח (אחד או יותר, אפשר גם שורה חדשה, חובה על מנת להבחין בפקודה c) וביטוי להחלפה (ללא צורך בסלאש סוגר).

מחיקה של שורה שלמה לפי ביטוי שהיא מכילה

הפקודה:

```
sed "/searchLinesThatContainThisText/d" <קובץ>
```

הערות:

- d - delete - מחק את השורה.
- /find/d גם כאן, היות שמדובר במחיקה המבנה של הפקודה משתנה. כאן כל מה שצריך זה סלאש פותח, ביטוי לחיפוש בתוך שורה ופקודת מחיקת שורה.

מחיקה של שורות לפי מספר שורה

הפקודה:

```
sed "5,10d;12d" <קובץ>
```

הערות:

- 5,10d מחק את השורות 5-10.
- ; מפריד בין 2 פקודות שונות.
- 12d מחק גם את שורה מספר 12.

הוספת שורה שלמה לפי חיפוש

הפקודה:

```
sed "/searchForThisLine/a lineToAppend" <קובץ>
```

הערות:

- a - append - הוספת שורה חדשה.
- /find/a lineToAppendAfter מבנה זהה להחלפת שורה שלמה (ראו לעיל דוגמא 2). השורה החדשה תתווסף לאחר השורה שנמצאה.

הוספת שורה שלמה לפי מספר שורה

הפקודה:

```
sed '8i This is new line 9' <קובץ>
```

הערות:

- 8i הוסף שורה חדשה לאחר שורה מספר 8.
- This is new line 9 זו השורה שתתוסף לקובץ ותהפוך לשורה מספר 9.

החלפת תוכן של שורה בודדת

הפקודה:

```
sed '34s/AAA/BBB/' <קובץ>
```

הערות:

- 34s עבוד על שורה מספר 34.
- /AAA/BBB/ החלף את הביטוי AAA ב- BBB.

שליפת נתונים לפי טווח של שורות רצויות

הפקודה:

```
sed -n 4,12p <קובץ>
```

הערות:

- -n שלוף שורות לפי טווח.
- 4,12p טווח השורות הרצויות. בדוגמא 4-12.

החלפת שורה רק אם היא לא מכילה את התו *

הפקודה:

```
sed '/\*/b; s/one/two/g' <קובץ>
```

הערות:

- /*/b; ביטוי שפירושו: "אם השורה הנוכחית מכילה את התו *, דלג עליה". היות שלתו * יש תפקיד מיוחד ואנו צריכים אותו כטקסט, הוא מופיע עם תו ה- ESCAPE סלאש הפוך.
- s/one/two/g ביטוי רגיל של sed שמחליף (s) את המילה 'one' במילה 'two' בכל פעם שהיא מופיעה (g).

החלפת ביטוי בביטוי שחלק ממנו הוא הביטוי שחיפשנו

לפעמים נרצה לשלב את הביטוי שחיפשנו עם ההחלפה. למשל אם נרצה להחליף ערך של משתנה בערך חדש, יהיה הכי נוח לחפש ולהחליף ביטוי שמכיל גם את שמו וגם את הערך שלו בביטוי שמכיל את שמו עם ערך חדש. במקרה כזה נשתמש בחלק מהערך שחיפשנו (שם המשתנה) בתור חלק מהערך להחלפה.

אז ככה זה עובד:

ראשית יש לסמן את החלק בביטוי שנרצה להשתמש בו בשלב ההחלפה. סימון זה מתבצע על ידי סוגריים עגולים עם סלאש הפוך לפניהם. ניתן לסמן יותר מחלק אחד. כל חלק שסומן ממופה על ידי sed ומוקצה לו שם שכולל סלאש הפוך בשילוב עם מספר לפי סדר הסוגריים בהתאמה. כך החלק הראשון שסומן יוגדר עבור הביטוי \1, החלק השני יוגדר כ-\2 וכן הלאה.

<u>הפקודה:</u>

```
echo "MY_VAR=444" | sed "s/\(MY_VAR=\)[0-9]*/\1888/"
```

הפקודה מחליפה תוכן של משתנה בשם MY_VAR בערך 888 ולא משנה מה היה הערך הקודם שלו.

<u>הערות:</u>

- `\(MY_VAR=\)` תחימה באמצעות סוגריים וסלאש הפוך של החלק שאנחנו רוצים להשתמש בו בהחלפה. הוא יקבל את הסימון \1.
- `*[0-9]` ביטוי רגולרי שפירושו "ביטוי שמכיל אפס או יותר ספרות בשורה".
- `\1888` השתמש בביטוי שסומן קודם והוסף לו את המספר 888. בדוגמא זו כפי שציינו למעלה, הביטוי "\1" שווה לביטוי "MY_VAR= ".

הוספת הסימן # לפני שורה

הפקודה הבאה מדגימה איך לשים בהערה שורה בקובץ.

<u>הפקודה:</u>

```
sed "/searchForThisLine/s/^/#/" <קובץ>
```

<u>לדוגמא:</u>

```
cat /home/ohad/examples.desktop | sed "/Comment/s/^/#/"
```

<u>הערות:</u>

- `searchForThisLine` השורה שאנו מחפשים. זו השורה שאנו רוצים לשים בהערה. גם חלק משורה נחשב ולכן לא חייבים לכתוב את השורה המדויקת.
- `/s/^/#/` החלף (s) את השורה שחיפשנו באותה שורה אבל עם # בתחילתה (^).

החלפת שורה בשורה אחרת רק אם השורה לא מתחילה בתו

הפקודה:

```
sed '/#/!{s/.*user.*/newLine/g}' <קובץ>
```

הערות:

- הפקודה בדוגמא זו חייבת להיות תחומה דווקא בין גרשיים **בודדים** אחרת Bash יעבד את סימן הקריאה שבפקודה במקום sed.
- `/#/` חפש את התו #.
- `!{ }` סימן הקריאה פירושו NOT. כל ביטוי עם ! בין גרשיים מסולסלים פירושו "בצע את מה שנמצא כאן אך רק אם לא ...". בפועל הפעולה בתוך הביטוי לא תתבצע אם מצאנו את הביטוי לחיפוש שיופיע לפני חלק זה.
- `s/.*user.*/newLine/g` מבנה רגיל של sed. ביטוי לחיפוש וביטוי להחלפה.
- המבנה של פקודת sed מסוג זה הוא:

```
sed '/expression_to_ignore/!{a_standard_sed_expression}' <file>
```

הוספת התו # בתחילת שורה רק אם הוא לא נמצא

דוגמא זו משלבת בין דוגמא מספר 10 לדוגמא מספר 12. היא משתמשת בביטוי משלב החיפוש בשלב ההחלפה ובנוסף היא משתמשת בביטוי השלילה.

הפקודה:

```
sed '/#/!{s/\(.*user.*\)/#\ \1/g}' <קובץ>
```

הערות:

- `\(.*user.*\)` שמירת הביטוי שחיפשנו בצד לשימוש בשלב ההחלפה (ראו הסבר מפורט בדוגמא 10).
- `\1` שימוש בביטוי ששמרנו בשלב ההחלפה (ראו הסבר מפורט בדוגמא 10).
- `/#\ \1/` החלף את הביטוי שחיפשנו בתו '#' + הביטוי שחיפשנו.
- `'/#/!{s/\(.*user.*\)/#\ \1/g}'` בצע את ההחלפה רק אם לא מצאנו את התו # (ראו הסבר מפורט בדוגמא 12).

ריבוי שורות ב- sed

כברירת מחדל, sed עובדת ברמת שורה. בפועל היא מחפשת את התו שורה חדשה (\n). לכן על מנת לציין לה לחפש ביטוי בחלק שמורכב מיותר משורה אחת, יש צורך בהגדרה מיוחדת.

הגדרה זו היא בפועל הביטוי N; (N גדולה ונקודה פסיק), ופירושה הוא: "דלג על \n אחד". במקרה זה sed יחפש ב-2 שורות במכה אחת. ניתן להכפיל אותו כמה שצריך כדי לחפש ביותר שורות (פעמיים שימוש בביטוי זה אומר חיפוש ב-3 שורות בפעם וכו').

לדוגמא,

הפקודה:

```
sed 'N;N; /.*user.*/d' <קובץ>
```

הערות:

- N;N; חפש ב 3 שורות בפעם. שימו לב שיש רווח בין ביטוי זה לבין הסלאש שפותח את הביטוי שאנו רוצים לחפש. **רווח זה הינו חובה.**
- /.*user.*/ חפש את הביטוי "מכיל את המילה user".
- d מחיקת 3 השורות שבהן נמצא הביטוי לחיפוש.

Head - הצגת חלק מתוכן ביחס להתחלתו

הפקודה head ואחותה tail הינן בין הפקודות המוכרות והשימושיות שיש. תפקידן להציג חלק מתוכן או מקובץ ביחס לתחילתו (head) או לסופו (tail) בהתאמה.

דוגמאות שימוש:

הצגת 8 השורות הראשונות מתחילת קובץ

הפקודה:

```
head -8 /etc/httpd/conf.d/php.conf
```

הערות:

- -8 מספר השורות שרוצים לקחת מהקובץ. בדוגמא זו מדובר ב-8 שורות. ברירת המחדל היא 10 וזהו מספר השורות שנקבל אם לא נציין מספר כלל.

הצגת השורה הראשונה מפלט של פקודה - הצגת קוד שגיאה מדף אינטרנט

הפקודה:

```
curl -I google.com | head -1
```

הערות:

- head -1 שליפת שורה אחת בלבד מהפלט של הפקודה curl. במקרה זה השורה הזו תכיל את קוד השגיאה שחזר מהדף המבוקש.
- curl -I הצגת הכותרות בלבד (HEADERS) שמחזיר דף אינטרנט.

tail - הצגת חלק מתוכן ביחס לסופו

כאמור לעיל, תפקידה של הפקודה tail להציג חלק מתוכן או מקובץ ביחס לסופו.

tail יותר שימושית מ- head היות שהיא מאפשרת מבט מהיר וקל על שינויים שהתבצעו לאחרונה בקבצים כגון קבצי לוג.

דוגמאות שימוש:

הצגת 6 השורות האחרונות מסוף קובץ

הפקודה:

```
tail -6 /var/log/messages
```

הערות:

- -6 מספר השורות שרוצים לקחת מהקובץ. בדוגמא זו מדובר ב-6 שורות. ברירת המחדל היא 10 וזהו מספר השורות שנקבל אם לא נציין מספר כלל.

הצגת כל התוכן החל משורה מספר 11 (כלומר למעט 10 השורות הראשונות)

הפקודה:

```
tail -n +11 /var/log/messages
```

הערות:

- -n +11 המתג n- מציין ל- tail מאיזו שורה לעבוד והפלוס (+) מציין "הכל החל מ-" (זאת בניגוד לשימוש הדיפולטיבי של "קח מהסוף את" שמוגדר על ידי הביטוי המלא - n -11 או בקיצור על ידי הביטוי 11- כמו בדוגמא הקודמת).

הצגת שינויים בקובץ בזמן אמת

הפקודה:

```
tail -f /var/log/httpd/access.log
```

הערות:

- זוהי אחת הפקודות היותר שימושיות ביום יום. היא מאפשרת לצפות רק בשינויים שמתבצעים החל מזמן הרצת הפקודה על הקובץ. כך ניתן לדעת בדיוק מתי הקובץ מתעדכן בזמן אמת ועם איזה תוכן.
- ניתן להשתמש גם בפקודה המקוצרת tailf במקום השימוש במתג f-.

vi - עורך טקסט

vi הוא עורך הטקסט הוותיק בלינוקס. ממבט ראשון הוא נראה מגושם אך כשמתעמקים מגלים יכולות מתקדמות רבות שניתן להשתמש בהן ביעילות רבה ובמהירות. הבעיה היחידה אתו היא הקיצורים הרבים שיש לזכור על מנת להשתמש בו כראוי.

בפועל, Ubuntu ו-CentOS עושות שימוש בגרסה המשופרת של vi ששמה הוא VIM שזה קיצור של **Vi Im**proved, אך ניתן להשתמש עדיין בפקודה vi הרגילה (היא בפועל קיצור דרך) ולכן נתייחס אל שמו המקורי של העורך על מנת למנוע בלבול מיותר.

הרשימה להלן היא בתקווה תשתית מספיק איתנה לשימוש אפקטיבי ויומיומי בעורך.

לרוצה להעמיק קצת יותר (או להיזכר בפקודות בסיסיות יותר), יש פירוט מורחב יותר על הפקודות והאפשרויות של העורך בנספח בסוף הספר.

<u>דוגמאות שימוש:</u>

הצגת נתונים בסיסיים על קובץ תוך כדי עריכתו

<u>הפקודה:</u>

Ctrl + g (קטנה)

הקיצור הזה יציג בתחתית המסך את שם הקובץ שכרגע פתוח ואת מספר השורה שאנו נמצאים בה כרגע.

גזור + הדבק / העתק + הדבק

<u>התהליך:</u>

1. מגיעים לתחילת הטקסט שרוצים להעתיק או לגזור.
2. לוחצים על v קטנה כדי להתחיל לסמן טקסט לפי תווים או V גדולה כדי לסמן טקסט לפי שורות.
3. מסמנים עם החיצים את הטקסט שרוצים להעתיק או לגזור.
4. לוחצים על y כדי להעתיק או על d כדי לגזור את הטקסט שנבחר.
5. זזים למיקום שבו רוצים להדביק את הטקסט.
6. לוחצים על p קטנה כדי להדביק את הטקסט לפני הסמן או על P גדולה כדי להדביק את הטקסט אחרי הסמן.

UNDO ו- REDO

<u>הפקודות:</u>

:UNDO

u (קטנה)

:REDO

Ctrl+r (קטנה)

Find and Replace

<u>התהליך:</u>

1. מקלידים את התו נקודתיים (:) כדי להגיע ל-"מצב פקודה" (COMMAND MODE).
2. מקלידים בתחתית החלון ביטוי במבנה הבא (מאוד דומה למבנה של פקודת ההחלפה ב-sed):

%s/findMe/replaceWithMe/g

<u>הסבר:</u>

%s	הגדרת מצב של 'מצא והחלף'
/	גרש פותח
findMe	ביטוי לחיפוש (הביטוי שיוחלף)
replaceWithMe	החלף ב- (הביטוי שמחליף)
/	גרש סוגר
g	greedy - חמדן - ללא הגדרה זו, תתבצע החלפה של טקסט אם הוא נמצא <u>פעם אחת בלבד</u> בכל שורה גם אם הוא נמצא יותר מפעם אחת בשורה אחת. עם הגדרה זו, תתבצע ההחלפה בכל פעם שנמצא הטקסט לחיפוש.

3. לאחר האישור עם ENTER, ההחלפה תתבצע.

חיפוש טקסט

<u>התהליך:</u>

1. מקלידים את התו סלאש רגיל (\) ולאחריו את הביטוי שרוצים לחפש.
2. כדי למצוא את הבא בתור מקלידים את התו n **קטנה**.
3. כדי למצוא את הקודם, מקלידים את התו N **גדולה**.

עריכת יותר מקובץ אחד במקביל

התהליך:

1. פותחים יותר מקובץ אחד על ידי שרשור של יותר מקובץ אחד לפקודה vi (שם של קובץ לא קיים יגרור יצירת של קובץ ריק חדש):

vi <קובץ 1> <קובץ 2> … <קובץ N>

2. בתוך vi, על מנת לעבור לקובץ הבא מקלידים את התו נקודתיים ולאחריהם n קטנה (:n) + ENTER. n=next. כמובן שניתן כך לנוע קדימה בין כל הקבצים שנפתחו.

3. על מנת לעבור לקובץ הקודם יש להקליד את התו נקודתיים ולאחריו את המילה prev בקטן (:prev) + ENTER. כמובן שניתן כך לנוע אחורה בין כל הקבצים שנפתחו.

הערה:

ניתן כמובן לשלב את היכולת הזו עם העתקה והדבקה מקובץ לקובץ (ראו סעיף 2 לעיל).

קפיצה לשורה ספציפית בקובץ

התהליך:

מקלידים את התו נקודתיים (:) ולאחריו מספר השורה הרצוי (למשל - 334:) + ENTER.

הצגת מספרי שורות בקובץ

התהליך:

מקלידים את התו נקודתיים (:) ולאחריו את הביטוי set number (או את חלקו. למשל - set nu:) + ENTER.

על מנת לבטל את המספרים ניתן להקליד את הפקודה ההפוכה set nonumber: (או את גרסתה המקוצרת) + ENTER.

תאימות לגרסה המקורית

Ubuntu ONLY

הסבר בקצרה:

זה יותר תיקון באג מאשר הפעלת תכונה.

בעיה זו קיימת רק ב-Ubuntu.

כברירת מחדל, מוגדר ל-Ubuntu להפעיל את VIM עם תאימות אחורנית ל-vi המקורי. הגדרות תאימות אלה גורמות לחיצים לא לעבוד במצב עריכה ושימוש בהם במצב זה גורם להקלדת אותיות במקום לזוז בתוך הקובץ וזה מאוד מציק.

ניתן בקלות להתגבר על הבעיה ולפתור אותה על ידי ביטול תכונת תאימות זו.

התהליך:

שינוי זמני (להפעלה הנוכחית בלבד):

1. נכנסים ל- vi
2. מקלידים את התו נקודתיים (:) ולאחריו את הביטוי:

set nocompatible

3. מאשרים עם ENTER.

שינוי קבוע:

1. עורכים קובץ בשם:

/etc/vim/vimrc.tiny

2. מאתרים את השורה:

set compatible

3. משנים אותה ל:

set **no**compatible

4. שומרים את השינוי.

קבצים ותיקיות

בפועל בלינוקס, כל המערכת מנוהלת דרך קבצים. לכן החלק העיקרי של סעיף זה נמצא בפרק "כלי מערכת" לעיל. יחד עם זאת, ישנם מספר פרטים נקודתיים בהקשר של ניהול מערכת הקבצים שכדאי להכיר.

הגדרות מיוחדות לקבצים

Ubuntu / CentOS

הסבר בקצרה:

קיימות 3 הגדרות נוספות שניתן לשייך לקבצים ותיקיות מעבר להגדרות הרגילות (של קריאה, כתיבה והרצה).

כאשר מתעסקים עם הרשאות, בדרך כלל משתמשים באופציה של שימוש במספרים כדי לציין הרשאות היות שזה יותר נוח. מדובר ב 3 ספרות ליתר דיוק (לדוגמא 777) אך בפועל יש מקום להגדרת ביט אחד נוסף (כברירת מחדל הוא 0 ולכן ניתן להשתמש רק ב-3 ספרות ברוב המקרים) שגם אותו ניתן להגדיר באותו אופן על ידי ספרות מתאימות או באמצעות אותיות מיוחדות המשויכות להגדרות אלה.

ביט רביעי זה הוא הביט הראשון מבין הארבעה והוא מאפשר להגדיר כאמור 3 הגדרות נוספות מיוחדות על קובץ או תיקייה:

1. sticky bit - "t" באותיות - "1" בספרות

לאחר הגדרה זו ה- x הרגיל של המקטע **השלישי** של ההרשאות (ההרשאות של "כל השאר" או "others") יהפוך ל-T גדולה בקבצים ול- t קטנה בתיקיות.

משמעות ההגדרה:

בקבצים:

מגדירה למערכת לשמור את הקובץ ב-RAM (לטובת שיפור ביצועים).

בתיקיות:

מונעת ממשתמשים שאינם הבעלים של התיקייה מלבצע שינויים כלשהם בתיקייה או בתוכנה גם אם יש להם הרשאות.

<u>פקודות לדוגמא:</u>

chmod +t <שם קובץ>
chmod 1755 <שם תיקייה>

2. SUID - משויך למשתמש - "s" באותיות - "4" בספרות

לאחר הגדרה זו ה-x הרגיל של המקטע **הראשון** של ההרשאות (ההרשאות של המשתמש) יהפוך ל-S גדולה בקבצים ול-s קטנה בתיקיות.

בקבצים:

מגדירה לקובץ לרוץ עם ההרשאות של הבעלים שלו במקום עם ההרשאות של המשתמש שהריץ אותו.

בתיקיות:

כנ"ל.

<u>פקודות לדוגמא:</u>

chmod u+s <שם קובץ או תיקייה>
chmod 4755 <שם קובץ או תיקייה>

3. GUID - משויך לקבוצה - "s" באותיות - "2" בספרות

לאחר הגדרה זו ה- x הרגיל של המקטע **השני** של ההרשאות (ההרשאות של הקבוצה) יהפוך ל- S גדולה בקבצים ול- s קטנה בתיקיות.

בקבצים:

מגדירה לקובץ לרוץ עם ההרשאות של הקבוצה שלו במקום עם ההרשאות של המשתמש שהריץ אותו.

בתיקיות:

כנ"ל.

<u>פקודות לדוגמא:</u>

chmod g+s <שם קובץ או תיקייה>
chmod 2755 <שם קובץ או תיקייה>

פקודות קשורות נוספות:

איפוס הגדרות (מחיקת הביטים המיוחדים):

chmod 0755 <שם קובץ או תיקייה>

הגדרת 2 הביטים האחרונים בפקודה אחת:

chmod +s <שם קובץ או תיקייה>

הצגת מספר קבצי טקסט זה לצד זה כעמודות בטבלה

<u>Ubuntu / CentOS</u>

<u>הפקודה:</u>

paste <2 קובץ> <1 קובץ> | pr -t -e24

<u>הערות:</u>

- הפקודה pr משמשת לסידור פלט. בדוגמא זו היא גורמת להצגת 2 הקבצים כעמודות בטבלה. המספר 24 מייצג סוג של מרחק יחסי בין העמודות שאיננו נמדד בערך כלשהו. ניתן להעלות או להוריד את ערכו כדי להתאים לעצמכם את המרחק הרצוי בין העמודות (למען האמת התיעוד של המתג הזה לא ממש ברור ולכן ההסבר כאן קצת מעורפל).
- במקור pr נועדה להכין פלט להדפסה. המתג t– מסיר הוספות דיפולטיביות של הפקודה כגון "מספר עמוד" ומשאיר רק את התוכן (את הפלט המקורי).

הפעלת פקודות / תכניות שמצריכות הרשאות ROOT ממשתמש רגיל שאין לו הרשאות SUDO

<u>הסבר בקצרה:</u>

קיימות פקודות שדורשות הרשאות מעל הרמה של משתמש רגיל.

אחת הדוגמאות לכך היא גישה לחומרה שדורשת הרשאות ROOT. לכן פקודה שעושה שימוש בחומרה לא תעבוד ללא הרשאות גבוהות. יוצא מכאן שקיימת בעיה כי לא יתכן שנגדיר את כל משתמשי המערכת כמנהלים רק על מנת שיוכלו להריץ פקודות רשת יומיומיות כגון PING שמשתמשות בחומרה. כדי לפתור את הבעיה משתמשים בהגדרה ברמת FILE SYSTEM שנקראת 'SETUID BIT' או 'SETUID PERMISSION' (נושא זה הוסבר גם לעיל בטיפ

נפרד ביתר פירוט). כאשר מגדירים אותה עבור קובץ בינארי, הוא ירוץ עם הרשאות **הבעלים** של הקובץ (שהינו ROOT בד"כ) במקום עם הרשאות המשתמש שהריץ אותו.

הביט מודלק על ידי הפקודה chmod ושמו הוא S. בשיטת ניהול הרשאות "777" הוא מוגדר על ידי ביט רביעי שלא משתמשים בו בד"כ והוא הביט הראשון מתוך הארבעה.

Ubuntu / CentOS

הפקודה:

chmod u+s <קובץ>

או

chmod 4755 <קובץ>

הצגת המיקום האמיתי של קובץ או תיקייה דרך לינק סימבולי

הסבר בקצרה:

לפעמים נרצה להגיע אל הנתיב האמיתי של קובץ או תיקייה דרך לינק סימבולי שיצרנו. לינקים עוזרים למשל עם המרת מיקום ארוך לקצר או המרת שם עם מספר גרסה לשם כללי או ליצירת אחידות של סביבת העבודה שלנו (כגון יצירה של לינק במיקום קבוע במערכות שונות אל קובץ שמיקומו משתנה בין מערכת למערכת).

במקרים כאלה יתכן שנרצה לעדכן את הקובץ או התיקייה שהלינק מצביע עליהם או להשתמש בהם (כגון להעתיקם) ואז נצטרך להגיע דרך הלינק אל המיקום האמיתי של התוכן.

Ubuntu / CentOS

הפקודה:

readlink -f <לינק סימבולי>

הצגת המיקום האמיתי של תיקייה דרך לינק סימבולי

הסבר בקצרה:

זו דוגמא נוספת להצגת המיקום האמיתי של לינק סימבולי אך רק עבור תיקיות.

במקרה הזה אנו משתמשים בפקודה pwd עם מתג מה שאומר שאופציה זו עובדת רק כאשר אנו נמצאים בתוך תיקייה שהיא בפועל לינק סימבולי ואנו רוצים לדעת מהו המיקום האמיתי שאליו הלינק מצביע (שהרי זה התפקיד של הפקודה pwd - למצוא את התיקייה הנוכחית שבה אנו נמצאים כעת).

Ubuntu / CentOS

הפקודה:

pwd -P

עדכון ערך של לינק סימבולי

הסבר בקצרה:

על מנת לעדכן ערך של לינק סימבולי בערך חדש נוכל להשתמש בפקודה הבאה.

Ubuntu / CentOS

הפקודה:

```
ln -sfn <נתיב חדש> <שם של לינק סימבולי קיים>
```

הערות:

- -f כפה עדכון. ללא מתג זה נקבל שגיאה שהלינק כבר קיים ולא ניתן לכתוב עליו.
- -n התייחס ללינק כאל קובץ רגיל. ללא מתג זה הפקודה תנסה לבצע פעולה על היעד שאליו מצביע הלינק במקום על הלינק עצמו ותיכשל.
- -s צור לינק סימבולי ולא לינק קשיח.

חומרה

זיהוי דיסק קשיח חדש במערכת

Ubuntu / CentOS

הסבר בקצרה:

אם הוספנו דיסק קשיח חדש למערכת ההפעלה כאשר היא עדיין פועלת, עלינו לבצע סריקה של ההתקנים על מנת שהמערכת תוכל לזהות אותו ולהשתמש בו.

ניתן לבצע זאת באופן מסורבל על ידי שליחת פרמטרים לקבצי מערכת שונים שנמצאים בתיקייה proc/ אך אין צורך להסתבך סתם. במקום זה, ניתן פשוט להתקין חבילה שמכילה סקריפט מסודר שמבצע את כל הפעולות הנדרשות ופשוט להריץ אותו.

תהליך:

1. ב- **Ubuntu** יש להתקין את החבילה:

scsitools

2. ב- **CentOS** יש להתקין את החבילה:

scsi-target-utils

3. לאחר ההתקנה יש להריץ עם הרשאות ROOT את הסקריפט:

rescan-scsi-bus.sh

4. הסקריפט יסרוק ויזהה אוטומטית את הדיסק החדש ויציג לנו את תוצאות הסריקה.

5. על מנת לוודא שהדיסק אכן התווסף בהצלחה לרשימת הדיסקים נוכל להריץ את הפקודה lsblk שמציגה התקני אחסון.

6. כעת נוכל להגדיר מחיצות חדשות בדיסק החדש. פרטים בטיפ הבא...

<u>הערות:</u>

- במקרים מסויימים הסקריפט לא יצליח לעדכן בזמן אמת את מצב הדיסקים ולא תהיה ברירה אלא להפעיל מחדש באופן מלא את המערכת על מנת שהדיסק החדש יזוהה על ידה.

קנפוג ראשוני של דיסק קשיח חדש

<u>Ubuntu / CentOS</u>

<u>הסבר בקצרה:</u>

הכנת דיסק לשימוש כוללת מספר שלבים.

כמובן שכל הפעולות צריכות להתבצע עם הרשאות ROOT.

נניח שאנו רוצים ליצור בדיסק מחיצה אחת.

להלן פירוט השלבים:

1. איתור הדיסק

על מנת לאתר את הדיסק נוכל להשתמש בפקודה lsblk. היא מציגה תמונת עץ של הדיסקים כולל הגדלים שלהם והמחיצות. נוכל לאתר את הדיסק החדש לפי גודלו ולפי העובדה שאין לו עדיין מחיצות.

דוגמא של פלט מהפקודה:

NAME	MAJ:MIN	RM	SIZE	RO	TYPE	MOUNTPOINT
sda	8:0	0	20G	0	disk	
├─sda1	8:1	0	300M	0	part	/boot
├─sda2	8:2	0	17.8G	0	part	/
└─sda3	8:3	0	2G	0	part	[SWAP]
sdb	8:16	0	5G	0	disk	
sr0	11:0	1	2.5G	0	rom	

ניתן לראות שהדיסק החדש הוא sdb היות שאין לו מחיצות.

2. יצירת מחיצות

תהליך:

1. מפעילים את fdisk על הדיסק החדש:

```
fdisk  /dev/sdb
```

2. לוחצים על:

n - ליצירת מחיצה חדשה (n=new).

p - ליצירת מחיצה ראשית (p=primary).

1 - בחירת מספר המחיצה (מחיצה ראשונה).

enter - אישור גודל מינימלי דיפולטיבי.

enter - אישור גודל מקסימלי דיפולטיבי.

w - כתיבת ההגדרות לדיסק בפועל (w=write).

במערכת עם GUI ניתן להשתמש בתוכנה הגרפית הנוחה GPARTED (ב- Ubuntu התוכנה זמינה דרך ה- REPO הראשי או לאחר הפעלת המחשב דרך דיסק ההתקנה במצב של LIVE CD וב- CentOS היא זמינה דרך REPO בשם EPEL שיש להתקין קודם).

3. פירמוט המחיצות

מפרמטים את המחיצה באמצעות הפקודה mkfs.ext4:

```
mkfs.ext4   /dev/sdb1
```

ניתן לבחור מערכת קבצים אחרת על ידי שימוש בפקודת mkfs עם סיומת אחרת לפי שם מערכת הקבצים. שימוש בפקודה mkfs ללא תוספת שהיא תפרמט את המחיצה ל- ext2.

4. הגדרת mount points ב- /etc/fstab

1. מציאת ה- UUID של המחיצה החדשה:

הפקודה blkid מציגה את המספר המזהה (UUID) של מחיצות. באמצעותה ניתן לזהות ולהגדיר מיפויים ב- /etc/fstab .

פלט לדוגמא:

```
/dev/sda2: UUID="4b0eaf4d-f177-4986-b642-d4c190a309b4" TYPE="ext4"
/dev/sda1: UUID="c0ebdc9b-f7cb-41bb-9d37-bfd0a6009e20" TYPE="ext4"
/dev/sda3: UUID="2ac7ea45-3101-4799-b740-a2e392e688ef" TYPE="swap"
/dev/sdb1: UUID="c16651f7-201d-47f0-978e-529b1b521ef6" TYPE="ext4"
```

ניתן לראות בפלט את ה- UUID של המחיצה החדשה sdb1 שיצרנו.

2. בוחרים מיקום במערכת הקבצים ויוצרים תיקייה חדשה שתהיה ה- mount point של המחיצה החדשה:

```
mkdir  /sdb1
```

3. עריכת /etc/fstab והוספת שורה עבור המחיצה החדשה תוך שימוש ב- UUID שלה ובתיקייה שיצרנו בשלב הקודם:

UUID=c16651f7-201d-47f0-978e-529b1b521ef6 /sdb1 ext4 defaults 0 2

הסבר האפשרויות הנוספות:

defaults = הכל מאופשר.

0 = אל תגבה את המחיצה אוטומטית.

2 = בדוק שגיאות במחיצה אוטו' בעת עליית המערכת עם עדיפות נמוכה (אחרי מחיצה השורש).

לפרטים נוספים אודות שאר האפשרויות השונות ראו נספח בסוף הספר.

4. טעינת /etc/fstab מחדש על מנת להחיל את השינוי ומיפוי המחיצה:

```
mount -a
```

לאחר שלב זה המחיצה מוכנה לשימוש וניתן לכתוב אליה קבצים דרך ה- mount point שהגדרנו עבורה (התיקייה /sdb1).

ניתן לוודא שמדובר ב- mount point של מחיצה ולא בתיקייה רגילה על ידי וידוא שתיקיית המערכת lost+found מופיעה בתוך ה- mount point. זו תיקייה שנוצרת אוטומטית בתוך כל mount point של מחיצה.

בדיקה ותיקון שגיאות בדיסק הקשיח

Ubuntu / CentOS

הסבר בקצרה:

ניתן להשתמש בפקודה e2fsck לבדיקה ותיקון שגיאות בדיסק הקשיח.

יש לפקודה גרסה עבור כל מערכת קבצים נתמכת.

אם חסרה פקודה עבור מערכת קבצים מסוימת והיא נתמכת על ידי המערכת, צריך פשוט להתקין את הפקודה המתאימה (לדוגמא: להתקנת גרסת הפקודה עבור מערכת הקבצים xfs, ניתן להתקין את החבילה "xfsprogs").

לא ניתן להריץ את הפקודה על מחיצה שהיא mounted. ניתן להריץ אותה למשל לאחר הפעלת המחשב מ LIVE CD.

פקודה:

```
e2fsck /dev/sdb1
```

הגדלת דיסק קשיח

הסבר בקצרה:

בעולמנו הווירטואלי קורה לא אחת שיש צורך בהגדלה דינמית של הדיסק הקשיח.

לאחר ההגדלה ברמת "החומרה", צריך להגדיר גם למערכת ההפעלה שהדיסק גדל.

כאן התהליך מתחלק לפי סוג המחיצה שנרצה להגדיל.

כברירת מחדל, CentOS ו-Ubuntu משתמשות במערכות שונות עבור ניהול המחיצות.

Ubuntu משתמשת במחיצות רגילות המפורמטות ישירות ל-ext4 בעוד ש-CentOS בונה את מערך המחיצות שלה מתחת למנהל המחיצות הלוגי LVM (Logical Volume Manager) שיודע לנהל את המחיצות באופן מתוחכם יותר.

Ubuntu

באובונטו, התהליך פשוט וקלאסי היות שמדובר במחיצות רגילות.

התהליך:

1. סריקה ותיקון שגיאות בדיסק (לפעמים נדרשת על ידי פקודת ההגדלה בהמשך כתנאי מקדים):

פקודה:

```
e2fsck  -f  /dev/sdb1
```

2. הרחבת המחיצה:

פקודה:

```
resize2fs  /dev/sdb1
```

CentOS

ב-CentOS לעומת זאת, התהליך טיפה יותר מורכב.

כאשר עובדים עם LVM מתבצעת הפרדה בין ההגדרה הלוגית של המחיצה לבין הגדרתה הפיזית זאת על מנת לאפשר ניהול מתקדם של מחיצות כגון חיבור מספר מחיצות פיזיות למחיצה לוגית אחת וכדומה.

קיימים 3 סוגים של אובייקטים:

1. הגדרה פיזית - Physical Volume - PV - מגדיר את פרטי המחיצה כיחידה פיזית.
2. הגדרה לוגית - Volume Group - VG - מגדיר קבוצה לוגית שיכולה להשתמש ביחידה פיזית אחת או יותר.

3. הגדרה לוגית - Logical Volume - LV - מחיצה לוגית המשויכת לקבוצה לוגית (הגדרה מספר 2). היא המחיצה השמישה שבפועל מפורמטת למערכת קבצים סטנדרטית כגון ext4 ואליה מקשרים mount point.

לכל אובייקט כזה יש סט של פקודות שמנהל אותו (לדוגמא הפקודות pvs, vgs ו- lvs המציגות נתונים כלליים אודות האובייקטים בהתאמה).

מכאן יוצא שכדי להגדיל מחיצה ב- LVM נדרש לבצע את הפעולות הבאות:

<u>תהליך:</u>

1. הרחבת ה- PHYSICAL VOLUME:

<u>פקודה:</u>

```
lvm pvresize /dev/sdb1
```

2. הרחבת ה- VOLUME GROUP ושיוך המקום החדש ל- LOGICAL VOLUME ספציפי:

<u>פקודה:</u>

```
lvextend -r /dev/mapper/VolumeGroup-lv_root /dev/sdb1
```

<u>הערות:</u>

- הפקודה pvresize עובדת רק עם LVM בגרסה 2.
- r- הרחב אף את מערכת הקבצים עצמה. אם לא הרצנו את הפקודה עם המתג r-, מערכת הקבצים שבה משתמשים בפועל דרך אותו LV (זו שיש עבורה mount point, למשל ext4) לא תורחב אלא רק ה-LV עצמו. אם נריץ את הפקודה df למשל, לא נראה את ההגדלה עדיין. על מנת להרחיב את המערכת במקרה כזה נצטרך פשוט להריץ את פקודת ההרחבה הרגילה ולהתייחס ל-LV (שהינה /dev/mapper/VolumeGroup-lv_root בדוגמא) כמחיצה רגילה כפי שצוין במקרה של אובונטו לעיל. כלומר הפקודות במקרה הזה הן:

```
1. e2fsck -f /dev/mapper/VolumeGroup-lv_root
2. resize2fs /dev/mapper/VolumeGroup-lv_root
```

הצגת נתוני החומרה של המערכת

<u>Ubuntu / CentOS</u>

<u>הסבר בקצרה:</u>

ניתן להציג את פרטי החומרה השונים של המערכת על ידי שילוב בין הרצת פקודות וצפייה בתוכן של קבצי מערכת.

להלן טבלה המכילה את הפרטים:

סוג	שם	מציג/ה פרטים אודות	הערות
קובץ	proc/cpuinfo/	המעבד	מכיל גם את כל הדגלים שמציינים את יכולות המעבד (instruction sets)
פקודה	lscpu	המעבד	
פקודה	free	הזיכרון	h- לגדלים קריאים
פקודה	lspci	התקני PCI	
פקודה	lsblk	דיסקים וכוננים	פלט בצורת עץ, נוחה לשימוש
פקודה	lsusb	התקני USB	
פקודה	lshw	כל החומרה במחשב	יתכן שתידרש התקנה שלה
פקודה	udevadm	דיסקים ומחיצות	מציגה פרטים מתקדמים. פקודה לדוגמא: udevadm info -q all -n /dev/sda
פקודה	vmstat	כל החומרה במחשב	פקודה לדוגמא: vmstat -S M

בדיקת הביצועים של המערכת

Ubuntu / CentOS

הסבר בקצרה:

לפעמים נרצה לבדוק את הביצועים של רכיבי המערכת השונים. להלן רשימה של כלים.

מה נרצה לבדוק	הפקודה / הכלי הבודק	הערות
עומס כללי על המערכת	uptime	ערך שגדול מ-1 כבר מעיד על עומס חשוד
cpu	top	הפקודה הטובה והמוכרת לבדיקת ביצועים
ram	top	
ביצועי רשת	iptraf	לבחור ב- "general interface statistics" מהתפריט
ביצועי רשת	iftop	פקודה נוספת לבדיקת ביצועי רשת, לא נמצאת כחבילה ויש להורידה מהרשת
ביצועי דיסק	iotop	יתכן שתידרש התקנה שלה, דומה מאוד ל-top

מערכת הפעלה

איך מתקינים את לינוקס באופן אוטומטי (unattended)?

הסבר בקצרה:

לכל מערכת הפעלה באשר היא, יש מנגנון שמאפשר להתקין אותה באופן אוטומטי לחלוטין ללא צורך בהקלקה על תפריט התקנה כזה או אחר (מה שנקרא unattended installation). בשיטה זו, מגדירים מראש בקובץ שנקרא "קובץ תשובות" (או answer file) את ההגדרות הרצויות ומפעילים את דיסק ההתקנה תוך ציון מיקומו של הקובץ. משלב זה המערכת תקרא את הקובץ ואם הוא תקין, היא תתקין את עצמה לבד ולא תעצור לבקש פרטים מהמשתמש.

CentOS

ב-CentOS מנגנון זה נקרא בשם "kick start" וקל למדיי להשתמש בו.

למעשה, בכל פעם שמתקינים את המערכת באופן ידני רגיל, נוצר קובץ תשובות בשם "anaconda-ks.cfg" עבור אותה התקנה ספציפית בתיקיית הבית של המשתמש - ROOT.

המשמעות היא, שעל מנת להתקין את אותה מערכת כפי שהיא באופן אוטומטי, כל שעלינו לעשות הוא לקחת את הקובץ הזה ולהפעיל את דיסק ההתקנה עם פרמטר שמגדיר אותו כקובץ התשובות עבור ההתקנה הנוכחית.

כמובן שנוכל להוסיף ולשנות ערכים לפני כן כפי שנרצה לפי הפורמט של הפקודות שהמנגנון מכיר.

תהליך:

1. בפעם הראשונה מתקינים כרגיל באופן ידני את המערכת.
2. שומרים את קובץ התשובות שנוצר כ- /root/anaconda-ks.cfg בתוך דיסק ההתקנה של המערכת בתיקיית השורש.

להלן דוגמא לקובץ תשובות מסוג kick start:

```
# Kickstart file automatically generated by anaconda.

#version=DEVEL
install
cdrom
lang en_US.UTF-8
keyboard us
network --onboot yes --device eth0 --bootproto dhcp
rootpw  --iscrypted $1$eh40Hxjp$o8pTwXuKL5XXupSsgxWxU1
# Reboot after installation
```

```
reboot
firewall --service=ssh
authconfig --useshadow --enablemd5
selinux --enforcing
timezone --utc America/Los_Angeles
bootloader --location=mbr --driveorder=sda --append="crashkernel=auto rhgb quiet"
# The following is the partition information you requested
# Note that any partitions you deleted are not expressed
# here so unless you clear all partitions first, this is
# not guaranteed to work
#clearpart --all --initlabel

#part /boot --fstype=ext4 --size=300
#part / --fstype=ext4 --grow --size=3000
#part swap --grow --maxsize=1984 --size=1984

repo --name="CentOS"  --baseurl=cdrom:sr1 --cost=100

%packages
@Base
@Core
@Desktop
@Fonts
@General Purpose Desktop
@Internet Browser
@Printing client
@X Window System
binutils
gcc
kernel-devel
make
patch
python

%end

%post
cp /boot/grub/menu.lst /boot/grub/grub.conf.bak
```

```
sed -i 's/ rhgb//' /boot/grub/grub.conf
if [ -f /etc/rc.d/rc.local ]; then cp /etc/rc.d/rc.local /etc/rc.d/rc.local.backup; fi
cat >>/etc/rc.d/rc.local <<EOF
#!/bin/bash
echo
echo "Installing VMware Tools, please wait..."
if [ -x /usr/sbin/getenforce ]; then oldenforce=\$(/usr/sbin/getenforce);
/usr/sbin/setenforce permissive || true; fi
mkdir -p /tmp/vmware-toolsmnt0
for i in hda sr0 scd0; do mount -t iso9660 /dev/\$i /tmp/vmware-toolsmnt0 && break;
done
cp -a /tmp/vmware-toolsmnt0 /opt/vmware-tools-installer
chmod 755 /opt/vmware-tools-installer
cd /opt/vmware-tools-installer
mv upgra32 vmware-tools-upgrader-32
mv upgra64 vmware-tools-upgrader-64
mv upgrade.sh run_upgrader.sh
chmod +x /opt/vmware-tools-installer/*upgr*
umount /tmp/vmware-toolsmnt0
rmdir /tmp/vmware-toolsmnt0
if [ -x /usr/bin/rhgb-client ]; then /usr/bin/rhgb-client --quit; fi
cd /opt/vmware-tools-installer
for s in sr0 sr1; do eject -s /dev/\$s; done
./run_upgrader.sh
if [ -f /etc/rc.d/rc.local.backup ]; then mv /etc/rc.d/rc.local.backup /etc/rc.d/rc.local; else
rm -f /etc/rc.d/rc.local; fi
rm -rf /opt/vmware-tools-installer
sed -i 's/3:initdefault/5:initdefault/' /etc/inittab
mv /boot/grub/grub.conf.bak /boot/grub/grub.conf
if [ -x /usr/sbin/getenforce ]; then /usr/sbin/setenforce \$oldenforce || true; fi
if [ -x /bin/systemd ]; then systemctl restart prefdm.service; else telinit 5; fi
EOF
chmod 755 /etc/rc.d/rc.local
if [ -x /bin/systemd ]; then systemctl enable rc-local.service; fi
/usr/sbin/adduser ohadm
/usr/sbin/usermod -p '$1$eh40Hxjp$o8pTwXuKL5XXupSsgxWxU1' ohadm
/usr/bin/chfn -f "o m" ohadm
%end
```

3. מפעילים את הדיסק תוך ציון הקובץ על ידי עריכת תפריט האתחול של הדיסק והוספת פרמטר בשם KS לשורה של initrd.

<u>תהליך:</u>

1. עורכים את הקובץ isolinux/isolinux.cfg.
2. מוסיפים את הקריאה לקובץ לאחר השורה "append initrd=initrd.img"

כך:

```
label linux
  menu label ^Install or upgrade an existing system
  menu default
  kernel vmlinuz
  append initrd=initrd.img   ks=cdrom:/anaconda-ks.cfg
```

3. שומרים את השינוי ומפעילים את הדיסק. ברגע שהמערכת תזהה שיש קובץ KS, היא תשתמש בו ותתקין את עצמה באופן אוטומטי לפי הפרטים שמופיעים בו.

<u>Ubuntu</u>

Ubuntu תומכת אף היא במנגנון kick start של CentOS אך באופן חלקי.

ניתן להשתמש בו או במנגנון שלה שנקרא "preseeding".

Ubuntu לא יוצרת קובץ הגדרות אוטומטי כמו CentOS ולכן ההמלצה במקרה שלה היא לעבוד עם קובץ תשובות לדוגמא שנמצא באתר הרשמי של אובונטו ולעדכן אותו לפי הצורך.

<u>דוגמא לקובץ תשובות מסוג preseed:</u>

```
# Text mode installer settings
d-i finish-install/reboot_in_progress note

# Language selection
ubiquity languagechooser/language-name    select    English
ubiquity countrychooser/shortlist    select    US
ubiquity time/zone          select    America/Los_Angeles
ubiquity debian-installer/locale    select    en_US.UTF-8
ubiquity localechooser/supported-locales    multiselect          en_US.UTF-8
ubiquity keyboard-configuration/modelcodestring    SKIP
console-setup    console-setup/layoutcode  string    us
console-setup    console-setup/layout      select    U.S. English
console-setup    console-setup/variantcode select    U.S. English
console-setup    console-setup/codeset     select    . Combined - Latin; Slavic Cyrillic;
Hebrew; basic Arabic
```

```
# Partitioning
ubiquity partman-auto/init_automatically_partition   select    Guided - use entire disk
ubiquity partman-auto/disk          string    /dev/sda
ubiquity partman-auto/method        string    regular
ubiquity partman-auto/choose_recipe          select    All files in one partition
(recommended for new users)
ubiquity partman/confirm_write_new_label boolean true
ubiquity partman/choose_partition select    Finish partitioning and write changes to disk
ubiquity partman/confirm boolean true
ubiquity partman/confirm_nooverwrite        boolean true

# User name and password
ubiquity passwd/user-fullname       string    o m
ubiquity passwd/usernamestring      ohad
ubiquity passwd/user-password-crypted password
$1$LqVh.G3z$LKFnRxAefIJq6dbsqsyY20
ubiquity user-setup/encrypt-home boolean false

# Miscellaneous
ubiquity ubiquity/reboot boolean true
ubiquity migration-assistant/partitions          multiselect
ubiquity ubiquity/summary           note
ubiquity apt-setup/security-updates-failed    note
ubiquity ubiquity/success_command           string \
apt-get update;\
apt-get install -q -y --force-yes --no-install-recommends open-vm-tools;
```

יש דוגמא עם הסברים מפורטים ב-

https://help.Ubuntu.com/lts/installation-guide/example-preseed.txt

דוגמא לקריאת הקובץ דרך תפריט האתחול של isolinux:

```
label  linux
kernel  vmlinuz.efi
append  initrd=initrd.lz  file=/cdrom/preseed/Ubuntu.seed  boot=casper  quiet  splash -
debian-installer/custom-installation=/custom  find_preseed=/preseed.cfg  auto
preseed/file=/floppy/preseed.cfg  automatic-ubiquity  noprompt  priority=critical
locale=en_US  console-setup/modelcode=evdev
```

הערות עבור 2 המערכות:

- זו הערה חשובה מאוד! וודאו שאתם מפעילים את ההתקנה האוטומטית על מחשב שאין צורך בנתונים שיש לו בדיסק הקשיח. תהליך ההתקנה ימחק לחלוטין את כל המחיצות בדיסק הקשיח ולכן יש לשים לב לכך ולהשתמש במנגנון בזהירות מרבית.
- יש תוכנה שמאפשרת ליצור באמצעות תפריט גרפי קובץ מסוג kick start. התוכנה זמינה להתקנה גם ב- CentOS וגם ב- Ubuntu ושמה הוא:

system-config-kickstart

- עבור עבודה מול המנגנון של Ubuntu, ניתן להתקין ולהשתמש בכלים שנמצאים בחבילה בשם debconf-utils.
- יש לחברת vmware תוכנה חינמית בשם vmware player שמאפשרת להריץ מכונות וירטואליות ויודעת ליצור לחלקן מנגנון התקנה ב-UNATTENDED שכולל מיני מערכת הפעלה שמכילה את קובץ התשובות ומשתמשת בדיסק ההתקנה המקורי של המערכת. זו דרך מאוד נוחה להשתמש במנגנון ההתקנה האוטומטי ללא צורך בשינוי המבנה של דיסק ההתקנה המקורי של המערכת. כמובן שהגרסה הלא חינמית של התוכנה הזו שנקראת vmware workstation גם מאפשרת להשתמש באותה תכונה.

התהליך ליצירת קובץ ה- ISO של מערכת זו הוא כזה:

1. יוצרים עם התוכנה מכונת לינוקס וירטואלית חדשה.
2. ממלאים פרטים (לא משנה איזה כי אנחנו מחליפים את קובץ התשובות).
3. לאחר התחלת תהליך ההתקנה נוכל לראות שמחובר למכונה החדשה שיצרנו קובץ ISO קטן בשם autoinst.iso. הקובץ הזה מכיל את המיני מערכת שהוזכרה לעיל ואת קובץ התשובות שיצרנו עם האשף של יצירת המכונה. ניתן בשלב זה לעצור את תהליך ההתקנה כי אנחנו רוצים הגדרות אחרות.
4. כעת נוכל לערוך את קובץ ה- ISO הזה ולהחליף את קובץ התשובות הקיים בקובץ תשובות משלנו תוך שינוי שמו לפי השם המופיע בדיסק (שמו צריך להיות ks.cfg).
5. על מנת להפעיל את ההתקנה האוטומטית לאחר שעדכנו את ה- ISO צריך לחבר **במקביל** את דיסק ההתקנה המקורי של המערכת לצד קובץ ה- ISO ולהפעיל את המכונה / המחשב ממנו. המיני מערכת הפעלה שנמצאת בקובץ autoinst.iso כבר תדע לטעון את קובץ התשובות ואף להשתמש בדיסק ההתקנה ותדאג להתקין את המערכת באופן אוטומטי.

איך מגדירים לשירות שיפעל אוטומטית לאחר עליית מערכת ההפעלה?

Ubuntu

הפקודה:

sudo update-rc.d <שם שירות> defaults

ביטול הפעולה:

sudo update-rc.d <שם שירות> disable

CentOS 6

הפקודה:

chkconfig <שם שירות> on

ביטול הפעולה:

chkconfig <שם שירות> off

CentOS 7

הפקודה:

systemctl enable <שם שירות>

ביטול הפעולה:

systemctl disable <שם שירות>

איך מריצים פקודה באופן אוטומטי לאחר שכל המערכת סיימה לעלות?

הסבר בקצרה:

לפעמים נרצה להפעיל פקודה או תכנית באופן אוטומטי מיד לאחר שהמערכת סיימה לעלות.

על מנת לבצע זאת, נוכל להשתמש בקובץ הסקריפט /etc/rc.local .

כל פקודה שנגדיר בסוף קובץ זה, תרוץ מיד לאחר העליה של המערכת.

Ubuntu

יש להוסיף את הפקודות הרצויות בשורה **לפני האחרונה** של הקובץ.

השורה האחרונה מכילה פקודת יציאה שמחזירה למערכת קוד שגיאה ויוצאת מהסקריפט ולכן אם נרשום את הפקודות שלנו בסופו הן לא ירוצו כי הסקריפט יסגר מיד לאחר פקודת היציאה.

CentOS 6

ניתן להוסיף את הפקודות הרצויות כרגיל בסוף הקובץ.

CentOS 7

ניתן להוסיף את הפקודות הרצויות כרגיל בסוף הקובץ, אך כברירת מחדל, הסקריפט **לא פעיל** ולכן על מנת להשתמש בו ראשית יש להפעילו על ידי הוספת הרשאות הרצה עבורו:

chmod +x /etc/rc.local

איך מתחברים למערכת הפעלה קיימת שלא עולה בכוחות עצמה מדיסק כדי לתחזק / לתקן אותה / לאפס סיסמאות?

הסבר בקצרה:

לינוקס ממפה גם התקנים לקבצים. בזכות תכונה זו, ניתן להתחבר בקלות למערכת הפעלה שלא עולה ולהשתמש בה כאילו היא עלתה כרגיל לאחר טעינה של מערכת חלופית מדיסק ומיפוי ההתקנים הנדרשים. כך ניתן לבחון את התקלה ולפתור אותה.

הערה חשובה לפני שמתחילים:

חובה להתאים את ה- LIVECD שממנו מתחברים אל המערכת התקולה לפי הפלטפורמה - 32 או 64 ביט - אחרת פקודת החיבור לא תעבוד.

התהליך בקצרה:

עושים MOUNT למחיצת השורש של הדיסק שמכיל את המערכת התקולה (/), מקשרים את ההתקנים הפיזיים של המערכת אל המחיצה ולאחר מכן מתחברים אל המערכת המקומית על ידי הפקודה chroot. לאחר הרצת הפקודה chroot בהצלחה נקבל טרמינל של Bash שמקושר למערכת התקולה ונוכל להשתמש בו כאילו המערכת עלתה כרגיל. כך נוכל לאפס סיסמאות, לתקן הגדרות וכדומה על מנת לפתור את הבעיה שמונעת את הפעלת המערכת באופן תקין.

Ubuntu

הפקודות:

```
mkdir  /mnt/myOS

mount   /dev/sda1   /mnt/myOS

mount  --bind  /dev  /mnt/myOS/dev
mount  --bind  /proc  /mnt/myOS/proc
mount  --bind  /sys  /mnt/myOS/sys

chroot /mnt/myOS
```

CentOS

הפקודות:

כברירת מחדל, CentOS מנהלת דיסקים ומחיצות באמצעות LVM ולכן לפני החיבור אל הדיסק יש לאקטב את ה- LVM. לאחר מכן התהליך זהה לאובונטו למעט שם המחיצה ששונה קצת בעקבות השימוש ב- LVM.

הפקודות:

הפעלת המחיצות של LVM:

```
vgchange  -a  y
```

שאר הפקודות:

```
mkdir   /mnt/myOS

mount  /dev/VolGroup/lv_root   /mnt/myOS

mount  --bind  /dev  /mnt/myOS/dev
mount  --bind  /proc  /mnt/myOS/proc
mount  --bind  /sys  /mnt/myOS/sys

chroot  /mnt/myOS
```

הערות:

- ניתן להשתמש בפקודה lsblk על מנת לאתר את המחיצה הרצויה.
- ב- CentOS, ה- LIVE CD מכיל אפשרות אוטומטית לחיבור למערכת קיימת. ניתן להשתמש בה במקום באופציה זו אך יש לקחת בחשבון שבמקרה זה יש סיכוי שמערכת הקבצים המקומית תאותחל לקריאה בלבד ולכן יש לעדכן לה את ההגדרה הזו על מנת לאפשר את הכתיבה לדיסק לטובת ביצוע השינויים הנדרשים לתיקון התקלה (ראו בהמשך טיפ בשם "REMOUNT למערכת הקבצים").

איך מתקינים את GRUB ?

הסבר בקצרה:

GRUB הוא מנוע האתחול של לינוקס. יש לו מספר גרסאות. בהקשר לינוקסי יש לו 2 - גרסה 1 וגרסה 2. יש לו גם גרסה לווינדוס שנקראת GRUB4DOS. ההבדלים בגרסאות נוגעים בעיקר ביכולות כגון תמיכה במערכות קבצים כאלה ואחרות וכדומה אך ברוב המקרים כל הגרסאות תדענה להפעיל את כל סוגי המערכות המוכרות בשוק (ווינדוס, לינוקס ועוד).

התהליך בקצרה:

מתחברים למחיצה שעליה רוצים להתקין את GRUB ומריצים את פקודת ההתקנה.

הערות:

יש לשים לב שההתקנה מתבצעת על הדיסק כולו (sda בדוגמא) ולא על מחיצה ספציפית היות ש- GRUB מותקן בראש הדיסק.

Ubuntu

הפקודות:

משתמשת ב- GRUB בגרסה 2.

```
mkdir  /mnt/myDisk
mount  /dev/sda1  /mnt/myDisk
grub-install  /dev/sda  --boot-directory=/mnt/myDisk
```

CentOS 6

הפקודות:

משתמשת ב- GRUB בגרסה 1.

```
mkdir  /mnt/myDisk
mount  /dev/sda1  /mnt/myDisk
grub-install  /dev/sda  --boot-directory=/mnt/myDisk
```

CentOS 7

CentOS בגרסה 7 עודכנה והיא משתמשת ב- GRUB בגרסה 2, ולכן פקודות ההתקנה זהות להתקנה על Ubuntu (ראו לעיל).

שינוי שם למחשב

Ubuntu

התהליך:

1. עורכים את הקובץ

/etc/hostname

2. מגדירים בו שם חדש ושומרים.
3. השינוי יקבל תוקף לאחר הפעלת מערכת ההפעלה מחדש.
4. על מנת לקבל את השם החדש מיידית נדרש להריץ את הפקודה:

```
hostname <שם_חדש>
```

CentOS

התהליך:

1. עורכים את הקובץ:

/etc/sysconfig/network

2. מגדירים בשורה HOSTNAME שם חדש ושומרים.
3. השינוי יקבל תוקף לאחר הפעלת מערכת ההפעלה מחדש.
4. על מנת לקבל את השם החדש מיידית נדרש להריץ את הפקודה:

```
hostname <שם_חדש>
```

הערות:

- כדי להימנע מבעיות, רצוי מאוד להוסיף את השם החדש גם לקובץ /etc/hosts .

REDIRECT OUTPUT

טבלה מקדימה:

סוג ערוץ	שם ערוץ	שם נוסף
קלט	STDIN	0
פלט	STDOUT	1
שגיאות	STDERR	2

הסבר בקצרה:

שגיאות מפקודות נשלחות לערוץ נפרד במערכת. בפועל הן גם מודפסות למסך שהינו התקן הפלט הדיפולטי אך על מנת לנתב אותן למיקום אחר כגון לקובץ, ניתוב רגיל עם הסימן '>' ינתב רק פלט שנשלח למסך וזה לא כולל שגיאות. על מנת לנתב גם את השגיאות לקובץ יש להשתמש באחת מהפקודות הבאות:

Ubuntu / CentOS

הפקודות:

אופציה א':

```
command   &>myFile.txt
```

אופציה ב':

```
command>myFile.txt   2>&1
```

הערות:

- אסור שיהיו רווחים בין התו '>' לבין 2 הביטויים שבצדדיו.

הצגת רשימת הספריות (LIBRARIES) שקובץ בינארי משתמש בהן

הסבר בקצרה:

הקבצים הבינאריים במערכת משתמשים בספריות משותפות שהינן בפועל קבצים בסיומת SO (קיצור של Shared Object), בדומה לקבצי DLL בווינדוס.

על מנת לדבג בעיות שקשורות לגרסאות של ספריות אלו, ניתן באמצעות פקודה בשם ldd לראות באילו ספריות משתמש קובץ בינארי נתון.

Ubuntu / CentOS

הפקודה:

```
ldd  <נתיב אל קובץ בינארי>
```

דוגמת שימוש:

```
ldd  `which mv`
```

פלט:

```
linux-vdso.so.1 =>  (0x00007ffcbc9e9000)
libselinux.so.1 => /lib64/libselinux.so.1 (0x0000003649e00000)
librt.so.1 => /lib64/librt.so.1 (0x0000003648e00000)
libacl.so.1 => /lib64/libacl.so.1 (0x0000003653200000)
libattr.so.1 => /lib64/libattr.so.1 (0x0000003658600000)
libc.so.6 => /lib64/libc.so.6 (0x0000003648600000)
libdl.so.2 => /lib64/libdl.so.2 (0x0000003648200000)
/lib64/ld-linux-x86-64.so.2 (0x0000003647e00000)
libpthread.so.0 => /lib64/libpthread.so.0 (0x0000003648a00000)
```

שינוי העורך הדיפולטיבי של המערכת

הסבר בקצרה:

כברירת מחדל, מוגדר למערכת עורך טקסט דיפולטיבי לטובת האפשרות לערוך קבצי מערכת של שירותים כגון CRONTAB שמנוהלים על ידי פקודה, ולא ניתן לגשת ישירות אל קובץ ההגדרות שלהם.

ב-CentOS העורך הדיפולטיבי הוא VI וב- Ubuntu ניתנת אפשרות לבחור מבין מספר אפשרויות לאחר הרצה ראשונה והבחירה נשמרת מאז.

ניתן לשנות את העורך הדיפולטיבי על ידי הגדרת משתנה סביבה בשם EDITOR עם המיקום של העורך שבו נרצה להשתמש.

לא לשכוח לוודא שהעורך שהגדרנו מותקן כמובן ...

Ubuntu / CentOS

הפקודה:

```
export  EDITOR=`which nano`
```

REMOUNT למערכת הקבצים

הסבר בקצרה:

אם המערכת נתקלת בקשיים בעלייה, היא עשויה להגדיר את מערכת הקבצים לקריאה בלבד, ואז לא יהיה ניתן לבצע כל שינוי (זהו סוג של מנגנון הגנה).

דוגמא לכך היא הגדרת מיפויים שגויים עבור מחיצות בקובץ /etc/fstab .

על מנת לאפשר את הכתיבה, ניתן להריץ את הפקודה הבאה:

Ubuntu / CentOS

הפקודה:

mount -o remount,rw <מחיצה שנרצה לאפשר כתיבה אליה>

דוגמא:

mount -o remount,rw /

הערות:

- ניתן להשתמש בפקודה זו גם כדי להסיר הקשחות באופן זמני ממחיצה שהקשחנו לטובת עדכונה (דוגמאות להקשחות מסוג זה ניתן למצוא בנספח בסוף הספר שמציג את האפשרויות השונות של הקובץ fstab).

איפוס סיסמת ROOT

טיפ זה מתייחס במקביל לשתי מערכות ההפעלה היות שקיימות חפיפות ביניהן אך הן חלקיות ולכן כותרות זה לא מספיק טוב במקרה זה. כאשר קיים שוני בין המערכות הוא יצוין במפורש.

קיימות 3 אופציות עיקריות לאיפוס סיסמת ROOT:

1. SINGLE MODE
2. דרך LIVE CD או RECOVERY MODE של אותה מערכת
3. יצירת HASH במערכת זהה והטמעתו

1. SINGLE MODE

הסבר בקצרה:

SINGLE MODE הוא מצב הפעלה של המערכת שבו היא עולה עם SHELL של ROOT וללא צורך בהכנסת סיסמא. מצב זה נועד לאפשר גישה מהירה לטובת פתרון בעיות.

על מנת להפעיל אותו עורכים את אפשרויות ההפעלה של המערכת על ידי עריכת התפריט של GRUB בעת עלייתה ומוסיפים את המילה single לאפשרויות הקרנל. לאחר עליית המערכת עם ROOT נוכל פשוט לאפס לו את הסיסמא כרגיל עם הפקודה passwd.

התהליך עובד כברירת מחדל ב-CentOS 6 וב- Ubuntu אך ב-CentOS 7 נדרשת סיסמא על מנת להיכנס למצב זה.

כמו כן, ניתן לבטל אפשרות זו על ידי הקשחת הגדרות המערכת כך שגם בהפעלת מצב זה המשתמש יתבקש להכניס סיסמא. במקרה זה יש לבחור באופציה אחרת לאיפוס הסיסמא.

תהליך:

1. מפעילים את המערכת.

2. מיד לאחר שהמסך של ה-bios עובר, לוחצים על אחד מהחיצים במקלדת עד שמופיע תפריט.
3. מאתרים בתפריט את השורה שמפעילה את המערכת (בד"כ היא תכיל את שמה), מוודאים שהיא זו שמסומנת ולוחצים על e על מנת לערוך אותה.
4. מאתרים את השורה שטוענת את הקרנל (חפשו את המילה KERNEL או את המילה LINUX).
5. מגיעים לחציה או לקצה שלה (הכי בטוח להוסיף את האופציה באמצע כדי שהיא תהיה בטוח במיקום הנכון) וכותבים שם את המילה **single**.
6. מפעילים את המערכת לאחר עדכון ההגדרה (תמיד יש מפה שמסבירה איך באחד מהצדדים על המסך. בגדול זה b ל- GRUB1 ו- Ctrl+X ל- GRUB2).
7. כעת המערכת צריכה לעלות ב- SINGLE MODE.

2. דרך LIVE CD או RECOVERY MODE של אותה מערכת

<u>הסבר בקצרה:</u>

לכל מערכת יש אפשרות עצמית להיכנס למצב שחזור.

ב- CentOS מתוך דיסק ההתקנה (ה-DVD, לא ה- LIVE CD) ניתן לבחור באופציה להציל מערכת קיימת. במקרה זה חשוב מאוד להקפיד על הפעלת הדיסק המדויק שממנו הותקנה המערכת (בפועל ברוב המקרים מספיק להקפיד על גרסת ביט זהה - 64 ביט למערכת 64 ביט ו- 32 ביט לגרסת 32 ביט) משם יהיה אפשרי לבחור מתפריט נוסף שנפתח ROOT SHELL ולקבל גישה מלאה אל המערכת. משם כל שנותר הוא לאפס את הסיסמא של ROOT כרגיל עם הפקודה passwd.

ב-Ubuntu ניתן לבחור מהתפריט לאחר ההתקנה את מצב RECOVERY שהינו מצב הפעלה בפני עצמו. לאחר הפעלתו יופיע תפריט שאחת מהאפשרויות בו היא קבלת ROOT SHELL ומשם יהיה ניתן לאפס את הסיסמא.

<u>הערות:</u>

- יש סיכוי שמערכת הקבצים המקומית תאותחל לקריאה בלבד לאחר הפעלת המערכת בשיטה זו. לכן יש לעדכן לה את ההגדרה הזו על מנת לאפשר את הכתיבה לדיסק לטובת ביצוע האיפוס לסיסמא (ראו טיפ בשם "REMOUNT למערכת הקבצים").

3. יצירת HASH במערכת זהה והטמעתו

<u>הסבר בקצרה:</u>

הסיסמאות במערכת שמורות באופן מוצפן כרצף תווים ואותיות בקובץ טקסט שנגיש ל- ROOT בלבד בשם /etc/shadow . סיסמא כזו נקראת HASH.

בשיטה זו, על מנת לאפס את סיסמת ROOT של מערכת ההפעלה המותקנת, נפעיל מדיסק חיצוני את המחשב, נאתר את ההגדרה של האלגוריתם שבאמצעותו נוצרים ה-HASHים של הסיסמאות וניצור בצד HASH חדש עם סיסמא חדשה שיצרנו. לאחר מכן נטמיע אותו בקובץ SHADOW של המערכת המותקנת, נפעיל אותה מחדש ונוכל להיכנס עם הסיסמא שזה עתה יצרנו.

תהליך:

1. מפעילים את המערכת מ- LIVE CD כלשהו.
2. מגיעים לקובץ ההגדרות של המערכת המותקנת ומוצאים את ההגדרה שקובעת באיזה אלגוריתם HASHING המערכת משתמשת. הקובץ הוא /etc/login.defs השורה היא ENCRYPT_METHOD והאלגוריתם הדיפולטיבי הוא sha-512.
3. מגדירים את אותו אלגוריתם במערכת שעלתה מהדיסק.
4. יוצרים בה משתמש חדש (useradd).
5. מגדירים לו סיסמא חדשה (passwd).
6. שולפים את ה- HASH של הסיסמא של המשתמש החדש שיצרנו מהקובץ /etc/shadow של המערכת שעלתה מהדיסק.
7. מחליפים אותו ב-HASH של המערכת המותקנת בשורה של המשתמש ROOT (ניתן להסתכל בקובץ SHADOW במשתמש שנוצר בסעיפים 4-5 במערכת שעלתה מהדיסק על מנת להבין היכן לשים את ה-HASH החדש. אם יש בשדה הזה * או ! יש להחליפו ב- HASH או לחילופין לאפס את ה-HASH של המשתמש שיש לו הרשאות ניהול במערכת (SUDOER).
8. מפעילים את המערכת מחדש ונכנסים עם המשתמש ROOT והסיסמא שנבחרה בסעיף 5 (או המשתמש SUDOER שבחרנו לאפס במקום).

שינוי צבע לקבצים ותיקיות

Ubuntu / CentOS

הסבר בקצרה:

ניתן לשנות צבע וסגנון לקבצים ותיקיות.

בעיה ידועה ב- CentOS 6 היא, שהתיקיות מופיעות ב- Bash בצבע כחול כהה ומאוד לא קריא ולכן כדאי לדעת איך ניתן לשנות את הצבעים.

הצבעים מוגדרים על ידי משתנה מערכת בשם LS_COLORS.

תהליך:

1. עורכים את קובץ ההגדרות של המשתמש הרצוי.

הקובץ הוא:

/home/<שם_המשתמש>/.bashrc

<u>הערות:</u>

שימו לב לתו נקודה בשם הקובץ (קובץ שמתחיל בנקודה פירושו קובץ מערכת והוא מוסתר כברירת מחדל).

2. יוצרים הגדרה רצויה לפי המבנה הבא:

<סוג>=<סגנון>;<צבע>

למשל:

di=0;35

פירוש הדוגמא הוא: שנה את הצבע של כל התיקיות לצבע סגול עם סגנון דיפולטיבי.

3. את ההגדרה החדשה משרשרים למשתנה שהוזכר לעיל:

LS_COLORS=$LS_COLORS:'di=0;35:'

4. מוסיפים פקודה נוספת שמגדירה את העדכון כמשתנה סביבה על מנת שהוא יקבל תוקף:

export LS_COLORS

5. שומרים את השינויים.

מעתה לאחר פתיחת חלון SHELL חדש על ידי המשתמש שעודכן, ההגדרות החדשות יופיעו.

לצבעים, סגנונות, וסוגי קבצים נוספים ראו נספח בסוף הספר.

<u>הערות:</u>

- ב-CentOS, ניתן להגדיר הגדרה זו לכל המערכת באופן גלובאלי על ידי עריכת הקובץ /etc/LS_COLORS .
- החל מגרסה 6.8, CentOS תיקנה את הצבע והוא כעת כחול בהיר וקריא.
- ניתן להחיל את השינויים באופן מיידי על ידי שימוש בפקודה source. למשל:

source /home/<שם_המשתמש>/.bashrc

הצגת פרטים אודות מערכת ההפעלה

<u>הסבר בקצרה:</u>

כדי לקבל פרטים אודות מערכת ההפעלה ניתן להשתמש ב-2 פקודות וקובץ.

הקובץ /etc/issue

הקובץ /etc/issue מציג את השם והגרסה של מערכת ההפעלה.

Ubuntu / CentOS

פקודה:

cat /etc/issue

פלט לדוגמא:

CentOS release 6.7 (Final)
Kernel \r on an \m

או

Ubuntu 16.04 LTS \n \l

הפקודה uname

הפקודה uname מציגה פרטים הכוללים את שם המחשב, השם הכללי של מערכת ההפעלה, הגרסה של הקרנל והארכיטקטורה של מערכת ההפעלה.

כך ניתן לדעת למשל האם מדובר במערכת בתצורה של 64 ביט או לא. הביטוי "x86_64" מציין מערכת של 64 ביט.

Ubuntu / CentOS

פקודה:

uname -a

פלט לדוגמא:

Linux localhost.localdomain 2.6.32-573.el6.x86_64 #1
SMP Thu Jul 23 15:44:03 UTC 2015 x86_64 x86_64 x86_64
GNU/Linux

או

Linux localhost 4.4.0-24-generic #43-Ubuntu SMP Wed Jun
8 19:27:37 UTC 2016 x86_64 x86_64 x86_64 GNU/Linux

הפקודה lsb_release

הפקודה lsb_release קיימת רק באובונטו ומציגה פרטים אודות הגרסה של המערכת.

Ubuntu ONLY

פקודה:

lsb_release -a

פלט לדוגמא:

No LSB modules are available.
Distributor ID: Ubuntu

Description: Ubuntu 16.04 LTS
Release: 16.04
Codename: xenial

הערה:

בפועל הפקודה lsb_release שולפת את הנתונים מקובץ בשם

/etc/lsb-release ולכן ניתן פשוט להציג אותו במקום להשתמש בפקודה (שימו לב שהקובץ הוא עם מקף ולא עם קו תחתון):

פקודה:

```
cat /etc/lsb-release
```

פלט:

```
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=16.04
DISTRIB_CODENAME=xenial
DISTRIB_DESCRIPTION="Ubuntu 16.04 LTS"
```

הגדרת ערכים גלובאליים

הסבר בקצרה:

ניתן להגדיר ערכים שישמרו ברמת כל המערכת ויחולו על כל המשתמשים ובכל מצב. בפועל מדובר בקובץ גלובאלי שנקרא בכל פעם שמשתמש נכנס למערכת או פותח טרמינל חדש.

לכל מערכת הפעלה יש קובץ גלובאלי משלה (בפועל מדובר בכמה קבצים אך מספיק להשתמש באחד מהם).

אל הקובץ מתייחסים כאל סקריפט ולא כאל קובץ הגדרות ולכן כותבים בו פקודות בדיוק כמו שכותבים בסקריפט או בטרמינל.

כלומר על מנת להגדיר למשל משתנה סביבה נכתוב את הפקודה:

```
export  MY_VAR="myValue"
```

כך ניתן להגדיר משתנים שנרצה שיוגדרו לכל משתמשי המערכת כגון הגדרות פרוקסי ארגוניות (ראו בחלק "רשת ותקשורת" אודות הגדרת פרוקסי דרך הטרמינל) או ALIASים וכדומה.

Ubuntu

הקובץ הגלובאלי:

/etc/bash.bashrc

CentOS

הקובץ הגלובאלי:

/etc/profile.d/custom.sh

הערות:

- אם הקובץ לא קיים ניתן ליצור אותו והמערכת אוטומטית תחפש אותו ותקרא אותו.
- ניתן להחיל את השינויים באופן מיידי על ידי שימוש בפקודה source. למשל:

```
source  /etc/bash.bashrc
```

סגירת כל התהליכים לפי שם משתמש

הסבר בקצרה:

לפעמים ה-SHELL נתקע ברמה כזו שצריך לסגור את כל התהליכים של המשתמש שפתח אותו כדי לאפשר לו לחזור לעבוד.

לשם כך יש פקודה מיוחדת בשם pkill.

Ubuntu / CentOS

הפקודה:

```
pkill  -9  -u  <שם משתמש>
```

הערות:

- לשים לב שאם סרוויס כלשהו רץ על שם המשתמש שהרגנו לו את כל התהליכים, יש לזכור להפעיל אותו מחדש.
- אם הרגנו את ה- SHELL שנתקע והוא שייך למשתמש ROOT, צריך לזכור להפעיל מחדש את כל התהליכים שרצו ברקע עם משתמש ROOT (כגון crond למשל).

סגירת חלון גרפי תקוע

Ubuntu / CentOS

הסבר בקצרה:

לפעמים חלון גרפי ספציפי נתקע ומסרב להגיב. ניתן לסגור אותו בכפייה על ידי שימוש בפקודה בשם xkill.

תהליך:

1. פותחים חלון terminal. החלון צריך לרוץ עם הרשאות של משתמש רגיל ולא של root (ב- Ubuntu אין בעיה להריץ כ- ROOT, ב- CentOS מופיעה שגיאה).
2. מקלידים את הפקודה xkill ללא פרמטרים (יתכן שנצטרך להתקינה):

```
xkill
```

3. תופיע בטרמינל בקשה לבחירת חלון רצוי. לוחצים עם העכבר על החלון שלא מגיב והוא ייסגר.

סגירת הממשק הגרפי דרך ה-SHELL

Ubuntu / CentOS

הסבר בקצרה:

מי שמגיע מעולם הווינדוס, עלול לשכוח שבלינוקס בניגוד לווינדוס הממשק הגרפי הוא הטפל ושורת הפקודה היא העיקר.

המשמעות היא שלא קרה כלום אם הממשק הגרפי נתקע פתאום בלינוקס. בווינדוס הוא העיקרי (בפועל גם חלון שורת הפקודה עצמו - ה- CMD - הוא חלון גרפי בווינדוס) ולכן צריך להפעיל מחדש את כל המחשב. בלינוקס לעומת זאת, ניתן פשוט לפתוח סשן חיבור אחר ללא GUI (חלון tty), להרוג משם את Xorg (זהו השם של התהליך שמריץ את הממשק הגרפי) ולהפעילו מחדש. זה הכל.

תהליך:

1. עוברים לסשן (tty) אחר על ידי צירוף המקשים Ctrl+Alt+F2 (ניתן לבחור במקש F אחר בין F1 ל- F6 אך יש לזכור שהממשק הגרפי ב- CentOS מוגדר על F1 ולכן לא ניתן לבחור בו. באובונטו אין בעיה כי הממשק הגרפי עובד על F7).
2. נכנסים עם משתמש בעל הרשאות root.
3. מריצים כ- root את הפקודה (הפקודה תגרום לסגירה בכפייה של הממשק הגרפי):

```
killall  Xorg
```

4. יתכן שהממשק הגרפי יפעיל את עצמו מחדש. אם זה לא יקרה ניתן להפעילו ידנית באמצעות הפקודה:

```
startx
```

INODES

Ubuntu / CentOS

הסבר בקצרה:

אחסון קבצים בלינוקס מורכב משני חלקים.

החלק הראשון נקרא inode והוא שדה המכיל פרטים אודות הקובץ המאוחסן כגון שמו, גודלו, מיקומו וכדומה.

החלק השני הוא הנתונים שמהווים את הקובץ עצמו.

לעתים אנו עלולים להיתקל במצב "מוזר" שבו איננו מצליחים ליצור במערכת הקבצים קבצים חדשים למרות שבמבט על הפלט של הפקודה df נראה שיש עוד מקום פיזי בדיסק.

מצב זה יכול לקרות כאשר טבלת ה- inodes מתמלאת כי לה יש הגבלה נפרדת בפני עצמה שאיננה קשורה למקום שנשאר על הדיסק. אם נוצרו במערכת המון קבצים קטנים הטבלה הזו עשויה להתמלא ואז לא תהיה יכולת ליצור קבצים חדשים גם אם נותר מקום פיזי.

על מנת לבדוק את מצב ה- inodes ניתן להשתמש בפקודה df, אך עם המתג **-i** :

<u>פקודה:</u>

```
df -i
```

<u>פלט לדוגמא:</u>

Filesystem	Inodes	IUsed	IFree	IUse%	Mounted on
/dev/sda2	1164592	104977	1059615	10%	/
tmpfs	364176	4	364172	1%	/dev/shm
/dev/sda1	76912	39	76873	1%	/boot

שימו לב ששמות השדות בכותרת של הטבלה מכילים את האות I לציין שמדובר במצב ה- inodes ולא באחוז השימוש בדיסק.

אם טבלת ה- inodes התמלאה, נוכל לקבל אינדיקציה למיקום הבעייתי במערכת הקבצים (זה שמכיל יותר מדיי קבצים קטנים) על ידי הרצת הפקודה du החל מתיקיית השורש. ברגע שנגיע לתיקייה שיש בה הרבה קבצים, פשוט נראה ש- du מתעכבת בחישוב גודלה הרבה יותר זמן מהרגיל וכך נדע שמדובר כנראה בתיקייה עם מספר חשוד של קבצים.

בנוסף, אנו עשויים להיתקל בתופעת לוואי כאשר ננסה למחוק את אותם קבצים כדי לפנות בטבלה מקום מחדש. ישנה הגבלה למספר הארגומנטים (הנתונים שנכנסים) ש- bash יכול לשמור בצד לפני שהוא מבצע עליהם פעולה כלשהי. אם ננסה למחוק עם התו * תוכן של תיקייה שמכילה הרבה מדיי קבצים, אנו עלולים להיכשל עם השגיאה:

"argument list too long!"

אם זה קורה, נוכל להתחכם ולאפשר בכל זאת את מחיקת הקבצים על ידי שליפתם אחד אחד באמצעות הפקודה find. למשל:

```
find /var/spool/postfix/maildrop -exec rm {} \;
```

מחיקת קבצים לא מחזירה את המקום שהם תפסו בדיסק

<u>הסבר בקצרה:</u>

לעתים קורה שהדיסק מתמלא מקבצים שנוצרים על ידי שירותים שונים שרצים במערכת כגון קבצי לוג, אך כשרוצים לפנות מקום ומוחקים אותם, המקום לא משתחרר.

זה קורה כי השירות ממשיך לנעול את הקובץ ולכן מערכת ההפעלה עדיין לא יכולה לעדכן את הגודל החדש של הקובץ ואת מצב הדיסק הנוכחי לאחר הניקוי.

על מנת לפתור את הבעיה צריך פשוט להפעיל מחדש את השירות שמשתמש בקובץ ואז המערכת תתעדכן עם הפרטים הנכונים.

מנהלי חבילות והתקנות

הסבר בקצרה:

הפצות הלינוקס השונות מורכבות בפועל מאוסף של חבילות עם יחסי תלות ביניהן.

קבוצת חבילות יכולה להגדיר תכונה קטנה או גדולה או את כל המערכת כולה.

לכן יש לכל מערכת מנהל חבילות שיכול להתקין ולנהל את החבילות השונות.

היות שהתקנת חבילה עלולה להסתבך בעקבות חבילות רבות שהיא תלויה בהן ושהן בעצמן תלויות באחרות וכן הלאה, בד"כ בנוסף למנהל החבילות הבסיסי שעובד ברמת חבילה בודדת, ישנו מנהל חבילות מתקדם יותר שנמצא מעליו והוא יודע להסתכל יותר מגבוה על המערכת ולספק תכונות מתקדמות כגון ניהול אוטומטי של היחסים בין החבילות והתלות ביניהן והתקנה אוטומטית של חבילות, אפשרות התקנת חבילות דרך האינטרנט ואפשרות התקנה של קבוצה של חבילות בפעם אחת שלהן תכונה משותפת (כגון: התקנת ממשק גרפי).

Ubuntu ו- CentOS אינן יוצאות דופן.

ב-Ubuntu, מנהל החבילות הבסיסי נקרא dpkg, מנהל החבילות המתקדם נקרא apt והחבילות הן קבצים בסיומת deb (חלק מהמילה Debian שהינה ההפצה שממנה Ubuntu נוצרה).

ב-CentOS, מנהל החבילות הבסיסי נקרא rpm, מנהל החבילות המתקדם נקרא yum והחבילות הן קבצים בסיומת rpm.

להלן דוגמאות שימוש במנהלי החבילות השונים.

Ubuntu

שם קובץ deb = שם קובץ ממש

שם חבילה = שם חבילה כפי שהיא מופיעה ב- apt (שם בלבד - ללא סיומת, מספר גרסה או ארכיטקטורה).

dpkg

פקודות על חבילות מותקנות או על קבצי deb

1. התקנת חבילה דרך קובץ deb:

```
dpkg -i <שם קובץ deb>
```

2. איפוס הגדרות של חבילה:

```
dpkg-reconfigure <שם חבילה>
```

3. הצגת תוכן של חבילה:

```
dpkg --listfiles <שם חבילה>
```

או

```
dpkg-query -L <שם חבילה>
```

4. הצגת רשימת כל החבילות המותקנות:

dpkg --list

<u>טיפ!</u>

ניתן לשלב את הפקודה עם grep כדי לבדוק האם חבילה מסוימת מותקנת ובאיזו גרסה.

5. מציאת חבילה לפי קובץ שהיא התקינה:

dpkg --search <קובץ>

6. קבלת נתונים מורחבים אודות חבילה (כגון גרסתה, תיאורה, רשימת החבילות שהיא תלוייה בהן ועוד):

dpks -s <שם חבילה>

apt

<u>פקודות על חבילות שעדיין לא הותקנו (למעט פקודת ההסרה)</u>

1. התקנה או עדכון חבילה:

apt-get install <שם חבילה או שם קובץ deb>

<u>הערה:</u>

אם רוצים להתקין קובץ deb עם apt, חייבים להשתמש בנתיב מלא אליו (למשל: apt-get install ./myDebFile.deb).

2. הסרת חבילה:

apt-get remove <שם חבילה>

3. הסרת חבילה וקבצי קונפיג (נדרשת לפעמים כשיש בעיות):

apt-get purge <שם חבילה>

4. חיפוש חבילה:

apt-cache search <שם חבילה>

5. הצגת תוכן של חבילה:

apt-file list <שם חבילה>

6. מציאת חבילה לפי קובץ שהיא מכילה:

apt-get install apt-file
apt-file update

apt-file search <קובץ>

7. הצגת רשימה של חבילות שחבילה תלויה בהן:

apt-cache showpkg <שם חבילה>

שם קובץ rpm = שם קובץ ממש

שם חבילה = שם חבילה כפי שהיא מופיעה ב- yum (שם בלבד - ללא סיומת, מספר גרסה או ארכיטקטורה). ב 'שם חבילה' נכלל גם שם של קובץ rpm בודד שהותקן דרך yum (לאחר ההתקנה yum שומר את שמו בלבד).

rpm

פקודות על חבילות מותקנות או על קבצי rpm

שם קובץ rpm = שם קובץ ממש

שם חבילה קיימת = שם חבילה כפי שהיא מופיעה לאחר התקנה במערכת

1. התקנת חבילה דרך קובץ rpm:

```
rpm -ivh <שם קובץ rpm>
```

2. עדכון חבילה:

```
rpm -Uvh <שם קובץ rpm>
```

3. התקנת מחדש של חבילה:

```
rpm -ivh --replacepkgs <שם קובץ rpm>
```

4. הסרה של חבילה:

```
rpm -e <שם חבילה קיימת>
```

טיפ!

ניתן לשלב את הפקודה עם rpm -qa ו- xargs כדי להסיר מספר חבילות בפקודה אחת.

5. הצגת רשימת כל החבילות המותקנות:

```
rpm -qa
```

טיפ!

ניתן לשלב את הפקודה עם grep כדי לבדוק האם חבילה מסוימת מותקנת ובאיזו גרסה.

6. הצגת רשימה של חבילות שחבילה תלויה בהן:

```
rpm -qpR <שם קובץ rpm>
```

7. הצגת תהליך התקנה של חבילה מבלי להתקין אותה בפועל (לטובת בדיקת תקינות החבילה למשל):

```
rpm -Kvv <שם קובץ rpm>
```

8. הצגת רשימה של קבצים שיש בחבילה:

```
rpm -ql <שם חבילה קיימת>
```

9. התקנת מפתח של repository מסוים על מנת שנוכל להתקין ממנו חבילות עם yum:

```
rpm --import <קובץ שמכיל מפתח PGP>
```

yum/rpm

<u>פקודות על חבילות שעדיין לא הותקנו (למעט פקודת ההסרה)</u>

חלק מהפקודות של yum תומכות בשתי האופציות.

1. התקנת חבילה:

yum install <שם חבילה או שם קובץ rpm>

2. הסרת חבילה:

yum remove <שם חבילה>

3. שנמוך חבילה לגרסה קודמת:

yum downgrade <שם חבילה>

4. חיפוש חבילה:

yum search <שם חבילה>

5. הצגת רשימת קבצים שיש בקובץ RPM:

rpm -qlp <שם קובץ rpm>

הידור קוד

<u>Ubuntu / CentOS</u>

<u>הסבר בקצרה:</u>

היות שהידור (Compilation) הוא יכולת מאוד בסיסית בעולם הקוד הפתוח, חשוב לפחות להזכיר אותו גם אם זה רק בקצרה.

אחד מהיתרונות הגדולים ביותר של עבודה בסביבה פתוחה הוא היכולת לשנות מה שרוצים ואיך שרוצים. קוד המקור של כל מערכת ההפעלה על כל חלקיה זמין לשימוש ולשינוי ומאפשר ליצור גרסאות חדשות וייחודיות בכל רמה.

לפעמים נרצה לשלוט ברכיבים שאנו מתקינים במערכת ברמת דיוק גבוהה מהרגיל ולא להסתמך על הגרסה הבינארית של אותו רכיב. סיבה אחת היא אבטחת מידע - במקרה זה רק תכונות שנשתמש בהן בפועל נכניס לחישוב כדי לצמצם כמה שניתן את טווח הפגיעות של המערכת.

סיבה שנייה היא שינוי מותאם אישית של הקרנל עצמו. היכולת לשנות את הליבה של מערכת ההפעלה איך שנרצה פותחת פתח לעולם שלם של מוצרים בעלי מטרה ספציפית. הקרנל כידוע מכיל את הדרייברים המאפשרים לחומרות שונות לעבוד ולכן אם נוכל לשלוט במה נכנס לשם ומה לא נוכל לעצב מכשירים מיוחדים מבוססי לינוקס למטרות מוגדרות מאוד כגון מערכות הפעלה רזות למחשבים רזים, נתבים, סטרימרים, רכיבי אבטחת מידע, טלפונים חכמים ועוד וזה מה שקורה כיום בפועל בעולם.

על מנת להשיג גרסה כזו המותאמת אישית לצרכינו, נצטרך לבצע הידור של קוד המקור לקבצים בינאריים של אותו רכיב מאפס בעצמנו, לאחר ביצוע השינויים הנדרשים בקוד עצמו או בהגדרות ההידור שיאפשרו לנו לקבל את הגרסה שרצינו.

תהליך ההידור משתנה בין מערכת למערכת ואפילו בין רכיב תוכנה אחד לאחר. לכן עם כל חבילת קוד מגיע קובץ עם הוראות התקנה ספציפיות לאותו מקרה. למרות הייחודיות של כל חבילת קוד כזו, על מנת ליצור בכל זאת איזו שהיא אחידות, בד"כ משתמשים בתהליך הידור די קבוע ודי מוגדר בזכות הצורך לעשות סדר בבלאגן שעלול להיגרם מקוד רב ומגוון שכפה יצירה של כלים ייעודיים שמפשטים את התהליך.

תהליך הידור של הקרנל עצמו בנוי מעט אחרת מהידור רגיל ומתבסס על התקנת קוד המקור של הקרנל ומספר כלים ייעודיים כגון הכלי menuconfig (בשילוב הפקודה make) שמציג תפריט ניהול מסודר שמאפשר לבחור את חלקי הקרנל הרצויים בקלות.

בכל אופן,

<u>תהליך קלאסי של הידור בנוי כך:</u>

1. השגת הקוד - משיגים את חבילת הקוד של רכיב התוכנה הרצוי.
2. קנפוג ראשוני - נכנסים לתיקיית הקוד ומריצים סקריפט בשם configure. הסקריפט נועד לזהות את המאפיינים הייחודיים לאותה מערכת, לוודא שכל דרישות הקדם של החבילה קיימות ולייצר קבצי הגדרות בהתאם לנתונים שנאספו אוטומטית בשילוב עם ההגדרות האישיות שהמשתמש בחר עבור ההידור. בשלב זה אנו עלולים להיתקע על קבצי מקור חסרים עקב דרישות קדם של החבילה. למשל: אם נרצה לקמפל את HTTPD עם יכולות SSL, נצטרך כדרישת קדם גם את הקוד של החבילה openssl. גם ב-CentOS וגם ב- Ubuntu יש חבילות בשמות דומים גם עבור הגרסה הבינארית של אותו רכיב תוכנה וגם עבור גרסת הקוד הלא מקומפלת. לשמות החבילות לא מקומפלות תתווסף מילה המסמנת אותן. מילה זו היא "devel" ב-CentOS ו-"dev" ב-Ubuntu. בהקשר של הידור לרוב נצטרך להתקין דווקא את החבילות הלא מקומפלות של רכיבים נדרשים אלא אם כן החבילה מצריכה גם את החבילה הבינארית. בכל מקרה נצטרך להתקין גם את הפקודות הבינאריות שמפעילות את הקמפול עצמו כגון הפקודה make ולעתים פקודות נוספות. לכן יש סיכוי טוב ככל שהרכיב מורכב יותר, שניתקע מספר פעמים בשלב זה כי בכל פעם ניתקע על דרישת קדם שונה ונצטרך להתקין עוד חבילה חסרה. לאחר שתהליך הקנפוג הראשוני באמצעות הסקריפט configure הסתיים בהצלחה, נוכל להמשיך לשלב הבא.
3. הידור עם הפקודה make - בשלב זה נריץ לרוב את הפקודה make ללא פרמטרים נוספים והיא תדאג להפוך את הקוד לחבילה בינארית מקומפלת. לפעמים נצטרך להריץ פקודות make נוספות כגון make depends לדוגמא שתפקידה לקמפל את החלקים הבסיסיים שרכיב התוכנה הנוכחי צריך כדי להתקמפל בעצמו.

4. התקנה עם make install - זהו השלב האחרון בתהליך. הרצת הפקודה make install תגרום להעתקת רכיבי התוכנה המקומפלים השונים למספר מקומות במערכת על מנת שנוכל להפעילה. לאחר שלב זה התוכנה מוכנה לשימוש ותהליך ההידור הסתיים.

<u>הערות:</u>

- חשוב לזכור שבכל שלב המערכת יוצרת קבצים לפי תצורת ההידור הנוכחית בתיקיית הפרויקט. אם שכחנו הגדרה מסוימת והוספנו אותה לאחר שכבר ביצענו ניסיון אחד לקמפל, רוב הסיכויים שנשארו קבצים בפרויקט שנוצרו בתהליך העלולים לשבש את הניסיונות הבאים ולהחזיר שגיאות משונות ולא ברורות. על מנת להיפטר מהבעיה הזו, ניתן להריץ את הפקודה make clean והיא תדאג לנקות קבצים שנוצרו בהידורים קודמים.
- תפקידו של הפרמטר prefix-- בתהליך של קונפיגורציה ראשונית באמצעות סקריפט configure הוא לציין היכן אנו רוצים להתקין את הרכיב שאנו מקמפלים (יעד). חשוב לא להתבלבל ולא להגדיר אותו עם ערך של המיקום של תיקיית הפרויקט עצמו (מקור) כי זה כמובן יצור קונפליקטים ושגיאות.
- יתכן ששמו של סקריפט הקנפוג יהיה שונה מהשם הדיפולטיבי configure (כגון "config"). לכן יש לשים לב להגדרות הקמפול הספציפיות לרכיב שרוצים לקמפל.

קבצי pid או socket למניעת הרצת תהליך פעמיים

<u>Ubuntu / CentOS</u>

<u>הסבר בקצרה:</u>

ישנם שירותים שרק עותק אחד שלהם יכול לעבוד בזמן נתון.

על מנת לוודא שרק עותק אחד רץ, הם יוצרים קובץ זמני במיקום כלשהו (משתנה בין תוכנה לתוכנה) וכך לפני הרצה השירות בודק האם הקובץ הזה קיים ורק אם הוא לא קיים השירות יעלה בהצלחה.

קובץ זה יכול להיות בעל שם וסוג מגוונים. סוג קלאסי לקובץ כזה הוא pid או socket ומיקום קלאסי של קבצי pid הוא var/run/. מיקומים אפשריים נוספים הם המיקום שבו הותקן השירות או התיקייה הזמנית של המערכת tmp/ אך זה לא קבוע.

הבעיה היא שאם השרת לא נכבה באופן מסודר או נתקע או שוכפל וכדומה יצא שיהיה לנו קובץ קיים והשירות יהיה למטה. ולא רק שהוא יהיה למטה אלא הוא יסרב לעלות מחדש כי הוא יזהה שהקובץ הזה קיים.

יש לזכור את המקרה הזה ואם זה קורה ניתן פשוט למחוק בבטחה את הקובץ ולהפעיל את השירות מחדש ללא שגיאות.

משתני סביבה

Ubuntu / CentOS

הסבר בקצרה:

באופן אוטומטי מוגדרים במערכת מספר לא מבוטל של משתני סביבה.

משתני הסביבה הינם משתנים שמוגדרים ברמת המערכת כולה או ברמת המשתמש הנוכחי ונגישים דרך ה- SHELL.

חלקם מכילים נתוני מערכת יבשים כגון שם המשתמש הנוכחי, שם המחשב, שם ה- SHELL שכרגע בשימוש ועוד וחלקם מכילים הגדרות אופרטיביות.

ההגדרות מחולקות ל 2: ישנן הגדרות ברמת מערכת וישנן ברמת אפליקציה.

ברמת המערכת ישנן הגדרות כגון: רשימת מיקומים שהמערכת תחפש בהם קבצי הרצה (המשתנה PATH - הטיפ הבא מרחיב אודותיו), איך לנהל את היסטוריית הפקודות (כמה פקודות לשמור, האם לשמור פקודות כפולות וכדומה), מראה ה- SHELL ועוד.

ברמת האפליקצייה - אפליקציות רבות עושות שימוש במשתני סביבה על מנת להגדיר מיקומים של ספריות והגדרות נוספות והן עשויות שלא לעבוד כראוי אם משתנים אלו לא יוגדרו או יוגדרו באופן לא נכון. דוגמא אחת מבין רבות היא סביבת ההרצה ו/או הפיתוח של JAVA שעובדת עם משתני סביבה כגון JAVA_HOME ו- CLASSPATH.

רשימת משתני הסביבה נגישה דרך הפקודה **env**.

על מנת להוסיף משתנה סביבה חדש יש להשתמש בפקודה **export** ולאחריה לציין את שם המשתנה שרוצים להוסיף לרשימה. חשוב לציין שהמשתנה החדש לא ישמר שם לתמיד אלא רק עבור ה-SHELL הנוכחי אלא אם כן נגדיר אותו באחד מקבצי המערכת שנטענים עם הפעלת SHELL חדש בין ברמת המשתמש ובין ברמת המערכת (ברמת מערכת ראו לעיל טיפ נפרד בנושא בשם "הגדרת ערכים גלובאליים". ברמת המשתמש יש את קבצי המערכת ברמת הפרופיל כמו הקובץ bashrc. שניתן להוסיף לו הגדרות כאלה).

איך מפעילים קבצי הרצה (פקודות, קבצים בינאריים או סקריפטים) מכל מקום

Ubuntu / CentOS

הסבר בקצרה:

אחד ממשתני הסביבה החשובים של המערכת (ראו הסבר אודות משתני סביבה בטיפ הקודם) נקרא PATH.

משתנה זה מכיל רשימה של נתיבים במערכת הקבצים. נתיבים אלה מגדירים למערכת היכן לחפש קבצי הרצה (כגון פקודות, סקריפטים או קבצי הרצה אחרים) כאשר קראנו לקובץ כזה ללא ציון מפורש של מיקומו.

ברגע שנגדיר במשתנה זה מיקום מסוים, נוכל להפעיל את קובץ ההרצה שקיים באותו מיקום מכל מקום במערכת ולא משנה באיזו תיקייה אנו נמצאים כעת.

תכונה זו מאוד מקילה ונוחה לשימוש כי היא מאפשרת לנו להריץ מה שנרצה מבלי להצטרך להגיע קודם אל אותו קובץ הרצה או לספק נתיב מלא אליו.

גם נתיבים של קבצי מערכת ופקודות מוגדרים על ידי משתנה זה ולכן אם נתקלנו במצב שבו לא הצלחנו להפעיל פקודה שאנו בטוחים שהיא קיימת במערכת זה כנראה בגלל שהנתיב אליה לא הוגדר כראוי עבור המשתמש הנוכחי.

הנתיבים מוגדרים כרשימה כאשר בין נתיב לנתיב מופיע התו נקודתיים (:).

למשל:

```
PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin:/home/ohadm/bin
```

על מנת להוסיף נתיב חדש, יש לשרשר את הנתיב הנוסף לרשימה הקיימת על מנת שלא לפגוע בגישה אל שאר קבצי ההרצה (שחלקם כאמור ברמת מערכת) כך:

```
PATH=$PATH:/my/new/path/to/bin/files

export PATH
```

שינוי זה יחול רק על ה- SHELL הנוכחי אלא אם כן נטמיע אותו בקובץ הגדרות בין ברמת המשתמש בין ברמת המערכת.

מה עושים אם משהו לא עובד מסיבה לא ברורה?

<u>הסבר בקצרה:</u>

לעתים קורה שלמרות שאנו בטוחים שהגדרנו משהו כמו שצריך הוא עדיין לא עובד.

במקרים כאלה יש לזכור שגם CentOS וגם Ubuntu משתמשות ב-2 רכיבים של אבטחת מידע שעשויים לגרום לבעיה ולכן כדאי לנסות לנטרל אותם על מנת לוודא שהם לא הגורם.

<u>CentOS</u>

<u>הרכיבים הם:</u>

1. FIREWALL - אם חומת האש של לינוקס הלא היא iptables דלוקה, היא עשויה להיות הגורם לבעיה. ניתן לנטרל אותה על ידי הורדת השירות:

<u>CentOS 6</u>

```
service iptables stop
```

CentOS 7

service firewalld stop

2. SELINUX - מערכת זו הינה תוסף אבטחה ברמת קרנל. המערכת מוסיפה הגדרות מערכת שמונעות לבצע פעולות שעשויות להיות מסוכנות או שונות מברירת המחדל.

בפועל קורה לא מעט שהיא הגורם לבעיות לא מובנות במערכת לאחר התקנה או קינפוג של שירות חדש שהגדרותיו שונות מברירת המחדל.

ניתן לנטרל אותה באופן זמני על ידי הקלדת הפקודה:

setenforce 0

לאחר הרצת הפקודה selinux תהיה מנוטרלת עד האתחול הבא או עד הקלדת הפקודה ההפוכה:

setenforce 1

על מנת לנטרל אותה באופן קבוע ניתן לערוך את קובץ ההגדרות שלה /etc/sysconfig/selinux ולשנות את הערך של המשתנה SELINUX ל- disabled.

Ubuntu

הרכיבים הם:

1. FIREWALL - אם חומת האש של לינוקס הלא היא iptables דלוקה, היא עשויה להיות הגורם לבעיה. ניתן לנטרל אותה על ידי הורדת השירות:

service ufw stop

2. APPARMOR - מערכת זו הינה תוסף אבטחה ברמת קרנל. המערכת מוסיפה הגדרות מערכת שמונעות לבצע פעולות שעשויות להיות מסוכנות.

ניתן לנטרל אותה על ידי הורדת השירות:

service apparmor stop

על מנת לבטלה לחלוטין ניתן להשתמש בפקודה:

sudo update-rc.d apparmor disable

הפעלת מודולים של הקרנל באופן דינאמי

Ubuntu / CentOS

הסבר בקצרה:

ניתן להפעיל מודולים של הקרנל גם לאחר עליית מערכת ההפעלה על מנת להשמיש תכונות שלא תמיד צריך אותן בשימוש השוטף ולכן הן לא מופעלות ישר עם עליית המערכת. מודול כזה לדוגמא הוא LOG מתקדם של IPTABLES שיופעל רק אם נטען את המודול המתאים לתוך הקרנל.

איך רואים איזה מודולים פעילים כרגע?

על מנת לראות את רשימת המודולים הפעילים במערכת ניתן להריץ את הפקודה lsmod שהינה קיצור של "list modules":

```
lsmod
```

פלט לדוגמא (רשימה חלקית):

```
[root@localhost Downloads]# lsmod
Module                  Size  Used by
nls_utf8               1455  1
udf                   84419  1
crc_itu_t              1717  1 udf
rfcomm                71079  4
sco                   17493  2
bridge                84537  0
bnep                  16370  2
l2cap                 54306  16 rfcomm,bnep
bnx2fc                90711  0
cnic                  55733  1 bnx2fc
uio                   10430  1 cnic
fcoe                  23298  0
libfcoe               56759  2 bnx2fc,fcoe
libfc                108606  3 bnx2fc,fcoe,libfcoe
scsi_transport_fc     55075  3 bnx2fc,fcoe,libfc
scsi_tgt              12077  1 scsi_transport_fc
8021q                 20362  0
garp                   7152  1 8021q
stp                    2218  2 bridge,garp
llc                    5418  3 bridge,garp,stp
```

איך רואים אילו מודולים קיימים במערכת?

ניתן לראות את רשימת המודולים המלאה על ידי שימוש בפקודה modprobe עם המתג -c (והרשימה של המודולים מופיעה בסוף הפלט בחלק שמכיל את השורות שמתחילות ב- " alias symbol"):

```
modprobe  -c
```

פלט לדוגמא (חלק מרשימת המודולים):

```
alias symbol:fat_time_unix2fat fat
alias symbol:nf_nat_follow_master nf_nat
alias symbol:fw_iso_buffer_init firewire_core
```

```
alias symbol:srp_rport_add scsi_transport_srp
alias symbol:fc_remote_port_rolechg scsi_transport_fc
alias symbol:fuse_dev_release fuse
alias symbol:p9_idpool_check 9pnet
alias symbol:snd_use_lock_sync_helper snd_seq
alias symbol:snd_jack_new snd
alias symbol:cm_class ib_cm
alias symbol:dst_comm_init dst
alias symbol:cx8802_register_driver cx8802
alias symbol:v4l2_event_free videodev
alias symbol:kvm_mmu_load kvm
alias symbol:rpc_peeraddr sunrpc
alias symbol:l2cap_load l2cap
alias symbol:lib80211_crypt_delayed_deinit lib80211
alias symbol:FsmInitTimer hisax
alias symbol:il_clear_ucode_stations iwlegacy
```

איך טוענים מודול נוסף לקרנל?

כדי לטעון מודול נוסף ניתן להשתמש שוב בפקודה modprobe:

modprobe <שם של מודול>

ניתן להשתמש גם בפקודה insmod אך היא מקבלת כפרמטר נתיב מלא אל קובץ מסוג מודול ו- modprobe מסתפקת בשם של מודול קיים בלבד ולכן modprobe נוחה יותר לשימוש.

איך מסירים מודול מהקרנל?

כדי להסיר מודול ניתן להשתמש בפקודה rmmod:

rmmod <שם של מודול>

כך ניתן למשל להסיר את המודול שמאפשר את השימוש ברמקול הפנימי הקטן שהיה קיים בעבר במחשבים ־ ה- PC SPEAKER וכיום הוא סתם גורם להשמעת צלילים לא נעימים (הבעיה קיימת בגרסאות ישנות של המערכות):

rmmod pcspkr

קבצי הגדרות מוסתרים

Ubuntu / CentOS

הסבר בקצרה:

היות שכל דבר במערכת מתנהל באמצעות קבצים, לכל רכיב כמעט ישנו קובץ הגדרות משלו.

בנוסף לקבצי ההגדרות המוכרים שמגדירים שירותים ידועים כגון שרת APACHE או שרת DHCP וכדומה, ישנם קבצי הגדרות מסוג נוסף שמתחילים בתו נקודה (.).

קבצים אלה הינם מוסתרים וניתן לראותם על ידי הוספת המתג a- לפקודה ls.

לעתים מדובר בתיקייה שלמה של הגדרות ולא רק בקובץ בודד ולכן גם היא תתחיל בתו נקודה.

כאמור לרכיבים רבים במערכת מקושר קובץ הגדרות כזה ולכן יתכן מאוד שמשהו לא יעבוד במערכת עקב קובץ הגדרות שאיננו מוגדר נכון או שלא הוגדרו עבורו הרשאות מתאימות ויש לשים לכך לב.

דוגמאות לקבצים ותיקיות כאלה (יש הרבה יותר ...):

.gconf
.bashrc
.Xauthority
.bash_logout

רשת ותקשורת

איך מבקשים כתובת IP משרת DHCP (הגדרות רשת דינאמיות זמניות)?

Ubuntu / CentOS

הפקודה:

dhclient <שם של כרטיס רשת>

הערות:

- כרטיס הרשת עשוי שלא להופיע בהרצה רגילה של הפקודה ifconfig היות שהוא עדיין לא מקונפג ולכן "כבוי" מבחינת המערכת. לכן על מנת לראותו כדי לאתר את שמו עבור הרצת הפקודה dhclient, נוכל להריץ את הפקודה ifconfig -a (a=all). ב- CentOS גרסה 7, הפקודה ifconfig נחשבת כמיושנת וכבר לא מותקנת כברירת מחדל (ניתן בכל זאת להתקינה אם רוצים) ולכן עדיף להשתמש במקום בפקודה ip address.
- יש לשים לב שגם המוסכמה של מתן שמות לכרטיסי רשת השתנתה ב- CentOS גרסה 7 והיא כבר לא ethX (X=מספר החל מאפס).
- הפקודה dhclient לאחר הפעלתה רצה ברקע כל הזמן. לכן אם שינינו לכרטיס כלשהו את החיבור הפיזי שלו וקישרנו אותו לשרת dhcp אחר, הרצה נוספת של הפקודה לעיל תחזיר שגיאה שאומרת שהפקודה כבר עובדת ולא ניתן להריצה שנית. על מנת להתגבר על הבעיה ניתן פשוט להרוג את המופע הנוכחי של הפקודה (עם הפקודה 'killall dhclient' למשל) ואז נוכל להפעילה שנית.

- חשוב לציין שהגדרה זו תהיה תקפה כל עוד המערכת פועלת והיא לא תישמר לאחר ריסטרט. על מנת שההגדרות ישמרו יש לקבען בקובץ ההגדרות של כרטיס הרשת שרוצים לקנפג. הטיפ הבא מסביר כיצד ניתן לעשות זאת.

איך קובעים לכרטיס רשת שיעבוד במצב DHCP באופן קבוע (הגדרות רשת דינאמיות קבועות)?

הסבר בקצרה:

Ubuntu

אובונטו מנהלת 2 שירותים (SERVICES) שונים עבור הגדרות רשת.

האחד נקרא network-manager והוא מנהל את השינויים שמתבצעים להגדרות הרשת על ידי המשתמש דרך הממשק הגרפי (GUI).

השני נקרא networking והוא מנהל את השינויים שמתבצעים להגדרות הרשת על ידי המשתמש דרך קובץ הגדרות בשם /etc/network/interfaces .

השירות networking בסיסי יותר והוא פעיל בכל מקרה (בין אם המערכת הותקנה עם ממשק גרפי ובין אם לאו) במקביל ל- network manager. לדוגמא: ה- loopback interface שמאזין על 127.0.0.1/8 מוגדר דרכו גם במערכת עם GUI. לכן אם נרצה נוכל להגדיר דרך שירות ה- networking את כל הגדרות הרשת.

בפועל, כמובן שניתן להגדיר הגדרות רשת של network-manager גם ללא GUI כי כל ההגדרות נשמרות לקבצים. הקבצים ש- network-manager מנהל באופן כללי נמצאים בתיקייה /etc/NetworkManager כאשר ההגדרות של חיבורי הרשת הסטנדרטיים נמצאות בקבצים בתוך תת-תיקייה בשם system-connections.

דרך NetworkManager (GUI)

תהליך:

1. עורכים את חיבור הרשת הרצוי (דרך האייקון של החיצים שליד השעון).
2. נכנסים לכרטיסייה "ipv4 settings".
3. מהרשימה העליונה (method) בוחרים ב- automatic.
4. שומרים.

דרך networking (/etc/network/interfaces)

תהליך:

1. מאתרים את שמו של כרטיס הרשת שרוצים להגדיר (עם הפקודה 'ip addr' למשל).
2. עורכים את הקובץ /etc/network/interfaces ומוסיפים לו את השורות:

```
auto <השם של כרטיס הרשת>
iface <השם של כרטיס הרשת> inet dhcp
```

למשל עבור כרטיס רשת ששמו הוא "ens38" נכתוב:

```
auto ens38
iface ens38 inet dhcp
```

3. מפעילים מחדש את השירות של הרשת:

```
service networking restart
```

CentOS

תהליך:

1. מאתרים את שמו של כרטיס הרשת שרוצים להגדיר (עם הפקודה 'ip addr' למשל).
2. עורכים או יוצרים קובץ במיקום:

```
/etc/sysconfig/network-scripts
```

ובשם (שימו לב למקף):

```
ifcfg-<השם של כרטיס הרשת>
```

למשל:

```
ifcfg-eth0
```

3. כותבים בקובץ את השורות הבאות:

```
DEVICE="<השם של כרטיס הרשת>"
BOOTPROTO="dhcp"
ONBOOT="yes"
```

אם הקובץ כבר קיים ניתן להתעלם משורות אחרות אך יש להחליף את השורות החופפות בשורות לעיל.

4. מפעילים מחדש את השירות של הרשת:

CentOS 6

```
/etc/init.d/network restart
```

CentOS 7

```
systemctl restart network
```

הערה:

אם יש במערכת יותר מכרטיס רשת אחד, יתכן ונקבל שגיאה לאחר הניסיון להפעיל מחדש את שירות הרשת. אם זה קורה, יש לערוך שנית את קובץ ההגדרות לעיל ולהוסיף לו שורה נוספת עם כתובת ה MAC של כל כרטיס רשת בהתאמה על ידי שימוש במילה "HWADDR".

לדוג':

```
HWADDR="00:0c:29:56:f1:37"
```

איך מגדירים כרטיס רשת עם כתובת IP סטאטית (הגדרות רשת סטאטיות)?

Ubuntu

ראו הקדמה אודות ניהול הגדרות הרשת באובונטו בטיפ הקודם.

דרך NetworkManager (GUI)

תהליך:

1. עורכים את חיבור הרשת הרצוי (דרך האייקון של החיצים שליד השעון).
2. נכנסים לכרטיסייה "ipv4 settings".
3. מהרשימה העליונה (method) בוחרים ב- manual.
4. תיפתח האפשרות להגדיר הגדרות רשת סטאטיות באופן ידני.
5. מגדירים את ההגדרות הרצויות.
6. שומרים.

דרך networking (/etc/network/interfaces)

תהליך:

1. מאתרים את שמו של כרטיס הרשת שרוצים להגדיר (עם הפקודה 'ip addr' למשל).
2. עורכים את הקובץ /etc/network/interfaces ומוסיפים לו את השורות:

```
iface <השם של כרטיס הרשת> inet static
    address <כתובת IP>
    netmask <NETMASK>
    gateway <כתובת GW>
```

למשל עבור כרטיס רשת ששמו הוא "ens38" נכתוב:

```
iface ens38 inet static
    address 192.168.100.1
    netmask 255.255.255.0
    gateway 192.168.100.254
```

3. מפעילים מחדש את השירות של הרשת:

```
service networking restart
```

CentOS

תהליך:

1. מאתרים את שמו של כרטיס הרשת שרוצים להגדיר (עם הפקודה 'ip addr' למשל).
2. עורכים או יוצרים קובץ במיקום:

/etc/sysconfig/network-scripts

ובשם (שימו לב למקף):

ifcfg-<השם של כרטיס הרשת>

למשל:

ifcfg-eth0

3. כותבים בקובץ את השורות הבאות:

```
DEVICE="<השם של כרטיס הרשת>"
BOOTPROTO="static"
ONBOOT="yes"
IPADDR=192.168.100.2
NETMASK=255.255.255.0
GATEWAY=192.168.100.254
```

אם הקובץ כבר קיים ניתן להתעלם משורות אחרות אך יש להחליף את השורות החופפות בשורות לעיל.

4. מפעילים מחדש את השירות של הרשת:

CentOS 6

/etc/init.d/network restart

CentOS 7

systemctl restart network

הערה:

אם יש במערכת יותר מכרטיס רשת אחד, יתכן ונקבל שגיאה לאחר הניסיון להפעיל מחדש את שירות הרשת. אם זה קורה, יש לערוך שנית את קובץ ההגדרות לעיל ולהוסיף לו שורה נוספת עם כתובת ה MAC של כל כרטיס רשת בהתאמה על ידי שימוש במילה "HWADDR".

לדוג':

HWADDR="00:0c:29:56:f1:37"

איך מגדירים ניתובי רשת סטאטיים?

ניתובים שלא נשמרים לאחר אתחול ניתן להגדיר באמצעות הפקודה route בשתי המערכות.

לדוגמא:

route add -net 1.2.3.4 netmask 255.255.255.255 gw 1.1.1.1

ניתובים קבועים שנשמרים באופן קבוע גם לאחר אתחול המחשב מגדירים כך:

Ubuntu

תהליך:

1. עורכים קובץ הגדרות בשם /etc/network/interfaces.
2. מוסיפים את הניתוב הרצוי כפקודת route בתוספת הגדרה לבצע אותה לאחר הפעלת הרשת (post-up).

לדוגמא:

```
post-up route add -net 1.2.3.4 netmask 255.255.255.255 gw 1.1.1.1
```

3. מאתחלים את שירות הרשת:

```
service networking restart
```

CentOS

תהליך:

1. יוצרים קובץ הגדרות בשם:

/etc/sysconfig/network-scripts/route-<השם של כרטיס הרשת>

(שימו לב למקף. את שם הכרטיס ניתן למצוא באמצעות הפקודה ip addr)

למשל:

/etc/sysconfig/network-scripts/route-eno16777736

2. מגדירים את הניתובים הרצויים בפורמט הבא:

```
1.2.3.4 /32 via 1.1.1.1
2.2.2.2/24 via 192.168.1.1
```

3. מאתחלים את שירות הרשת:

```
service networking restart
```

איך מגדירים Default Gateway?

Ubuntu / CentOS

הפקודה:

route add default gw < כתובת IP>

איך מוודאים שכתובת IP לא נמצאת כבר בשימוש?

Ubuntu / CentOS

הסבר בקצרה:

לפני שמנסים להגדיר כתובת IP סטאטית באופן ידני, כדאי לוודא שהיא פנויה ולא נמצאת בשימוש על ידי התקן אחר ברשת אחרת יהיו קונפליקטים שיצרו בעיות.

דרך פשוטה לבצע את הבדיקה היא להריץ פקודה כלשהי שעושה שימוש ברשת (כגון ping, telnet, wget, curl וכדומה) ממחשב אחר שכבר נמצא ברשת ולאחר מכן לבדוק את טבלת ה- arp. אם בטבלה מופיעה הכתובת שרצינו להקצות לצד כתובת MAC זה אומר שהיא תפוסה ואם הסטאטוס של ה- IP מופיע כ- "incomplete" זה אומר שהכתובת פנויה וניתן להשתמש בה.

למשל:

נתבונן בטבלת ה- ARP הבאה על ידי הרצת הפקודה **arp**:

```
root@aaa:/home/ohad# arp

Address          HWtype  HWaddress          Flags Mask     Iface
192.168.40.2       ether   00:50:56:f5:96:8c  C              ens38
192.168.40.254     ether   00:50:56:fa:a6:29  C              ens38
```

ניתן לראות בפלט של הפקודה את הכתובות שכבר מוכרות לנו.

עכשיו נניח שנרצה לבדוק האם הכתובת **192.168.40.28** פנויה לשימוש.

אז נריץ למשל את PING ונקבל:

```
root@aaa:/home/ohad# ping 192.168.40.28

PING 192.168.40.28 (192.168.40.28) 56(84) bytes of data.
From 192.168.40.135 icmp_seq=1 Destination Host Unreachable
From 192.168.40.135 icmp_seq=2 Destination Host Unreachable
From 192.168.40.135 icmp_seq=3 Destination Host Unreachable
^C
--- 192.168.40.28 ping statistics ---
4 packets transmitted, 0 received, +3 errors, 100% packet loss, time 3015ms
pipe 3
```

ברקע, המערכת תנסה להשיג את כתובת ה- MAC שמקושרת לכתובת ה- IP הזו על ידי שימוש בפרוטוקול ARP. אם היא לא תמצא, נקבל סטאטוס של incomplete על הכתובת כפי שהוזכר לעיל:

```
root@aaa:/home/ohad# arp
```

```
Address          HWtype  HWaddress          Flags Mask      Iface
192.168.40.2       ether   00:50:56:f5:96:8c  C               ens38
192.168.40.28            (incomplete)                       ens38
192.168.40.254     ether   00:50:56:fa:a6:29  C               ens38
```

וזה אומר מבחינתנו שכרגע הכתובת הזו פנויה לשימוש.

הערות:

- לא לשכוח שהבדיקה רק מעידה על כך **שכרגע** אין שימוש בכתובת הנבדקת. אם יש מחשב כבוי ברשת שגם הוא מחזיק את הכתובת שבדקנו אנו עדיין נהיה בבעיה כשהוא יופעל.
- בנוסף יתכן שיש ברשת שרת DHCP שמקצה כתובות ברשת זו ומוגדר לו טווח כתובות להקצאה הכולל את הכתובת שבחרנו והוא פשוט לא הגיע עדיין לשלב שבו עליו להקצות אותה.

הצגת הפורטים שכרגע בשימוש על ידי המערכת

Ubuntu / CentOS

הפקודה:

```
lsof  -i  -n  -P
```

פלט לדוגמא:

```
COMMAND   PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
cupsd    2275 root    7u  IPv6  14025      0t0  TCP [::1]:631 (LISTEN)
cupsd    2275 root    8u  IPv4  14026      0t0  TCP 127.0.0.1:631 (LISTEN)
cupsd    2275 root   10u  IPv4  14029      0t0  UDP *:631
dhclient 2278 root    6u  IPv4  14007      0t0  UDP *:68
sshd     2512 root    3u  IPv4  14690      0t0  TCP *:22 (LISTEN)
sshd     2512 root    4u  IPv6  14699      0t0  TCP *:22 (LISTEN)
master   2641 root   12u  IPv4  15002      0t0  TCP 127.0.0.1:25 (LISTEN)
master   2641 root   13u  IPv6  15004      0t0  TCP [::1]:25 (LISTEN)
```

הערות:

- יתכן שהפקודה לא תהיה קיימת ויהיה צורך להתקין אותה.
- הפקודה מציגה רשימה של קבצים פתוחים לפי חיתוכים שונים (ראשי התיבות של הפקודה הם "list open files"). בלינוקס זה אומר "את הכל" כי כל המערכת היא אוסף של קבצים. על מנת להבין זאת, נסו להריץ אותה ללא פרמטים כלל...
- -i הצג רק קבצים שקשורים לתקשורת ב- IP גרסה 4 או 6.

- -n אל תציג HOSTNAMES.
- -P אל תציג את הפורטים לפי שם אלא לפי מספר.

הגדרת שרת PROXY עבור חלון ה- SHELL הנוכחי

Ubuntu / CentOS

הסבר בקצרה:

בסביבה ארגונית שבה היציאה לאינטרנט מתבצעת על ידי פרוקסי, על מנת להשתמש בו גם דרך ה- SHELL, ניתן להגדירו באמצעות יצירת משתני הסביבה הבאים בהתאמה לתפקידם.

לאחר הגדרתם, פקודות כמו CURL ו- WGET יעברו דרכו אל האינטרנט.

פקודות:

```
export  http_proxy=http://192.168.11.2:3128
export  https_proxy=http://192.168.11.2:3128
export  ftp_proxy=http://192.168.11.2:3128
export  no_proxy="127.0.0.1,192.168.22.5"
```

הערות:

- היות שהמימוש משתנה מפקודה לפקודה, יתכן שהגדרות אלה לא יעבדו על כולן. במקרה כזה, צריך לעיין בתיעוד של אותה פקודה שלא עוברת פרוקסי למרות הגדרות אלה ולבדוק כיצד ניתן להגדיר לה פרוקסי באופן פרטני.
- הרצת פקודות אלה בחלון טרמינל, תגרום להגדרתן אך ורק באותו חלון נוכחי בלבד. ברגע שנסגור אותו, ההגדרות תלכנה לאיבוד.
- ניתן להגדיר הגדרות אלה ואחרות בקובץ גלובאלי במערכת ההפעלה וכך הם ישמרו לתמיד ועבור כל המשתמשים במערכת. ראו טיפ בשם "הגדרת ערכים גלובאליים" בחלק "מערכת הפעלה".

משתמשים ופרופילים

איפוס פרופיל של משתמש

Ubuntu / CentOS

הפקודות:

1. cp -R /etc/skel/* /home/<שם משתמש>
2. cp -R /etc/skel/.??* /home/<שם משתמש>
3. chown -R <שם משתמש>:<שם משתמש> *
4. chown -R <שם משתמש>:<שם משתמש> .??*

הערות:

- /etc/skel הינה תיקיית מערכת שמכילה את הפרופיל הכללי הבסיסי או ליתר דיוק את הקבצים והתיקיות שמהווים את הגדרות הפרופיל הדיפולטיביות של המשתמש. המילה SKEL הינה קיצור של המילה SKELETON שפירושה "שלד". בכל פעם שיוצרים משתמש חדש במערכת, המערכת מעתיקה אל תוך תיקיית הבית החדשה שלו את תוכן התיקייה הזו. לכן למעשה איפוס פרופיל ידני מתבצע על ידי העתקה פשוטה של תוכנה של התיקייה הזו אל תוך תיקיית הבית של המשתמש שרוצים לאפס לו את הפרופיל ומתן הרשאות מתאימות. ניתן גם להוסיף לשם קבצי הגדרות נוספים ואז כל משתמש חדש שניצור במערכת יקבל גם את ההגדרות הנוספות.
- <שם משתמש> מתייחס בהתאמה גם לשם הקבוצה של המשתמש שזהה לשם המשתמש.
- פקודות מספר 2 ו- 4 מתייחסות לקבצים ותיקיות מוסתרים שהינם קבצי ההגדרות של הפרופיל. בפועל במקרה הזה מדובר בקבצים החשובים ביותר ולכן חשוב לאפס אותם. התו כוכבית מתייחס לכל תו אך לא כולל קבצים מוסתרים (ששמם מתחיל בתו נקודה) ולכן פקודות מספר 1 ו- 3 תתעלמנה מקבצים מוסתרים. הביטוי "*??." חייב להיכתב בדיוק כמו שהוא כי הביטוי הפשוט *. כולל בתוכו ביטויי מערכת נוספים חוץ מקבצים רגילים כגון הביטוי "..." שפירושו "כל מה שיש בתיקייה הקודמת" ואת זה אנחנו לא רוצים. שני סימני שאלה מבטיחים שם של קובץ כי הם מחייבים שני תווים רגילים לאחר התו נקודה (סימן שאלה בודד גם לא מספיק היות שכוכבית זה גם "כלום" ואז שוב נכלל לנו הביטוי "...").

לוגין אוטומטי (GUI)

הסבר בקצרה:

ניתן להגדיר כניסה אוטומטית למערכת

Ubuntu

תהליך:

1. עורכים קובץ בשם /etc/lightgdm/lightgdm.conf
2. תחת המקטע [SeatDefaults], מגדירים את המשתמש הרצוי על ידי הוספת השורה הבאה:

autologin-user=<שם משתמש>

CentOS

תהליך:

1. עורכים קובץ בשם /etc/gdm/custom.conf
2. תחת המקטע [daemon], מגדירים את המשתמש הרצוי על ידי הוספת השורות הבאות:

AutomaticLoginEnable=True
AutomaticLogin=<שם משתמש>

יצירת תיקיית בית למשתמש

הסבר בקצרה:

במקרה שבו לא נוצרה למשתמש תיקיית בית או שהשתבשו בה קבצי הפרופיל, ניתן להשתמש בפקודה הבאה כדי ליצור למשתמש את תיקיית הבית שלו מחדש עם קבצי ההגדרות הדיפולטיביים שלה.

Ubuntu / CentOS

הפקודה:

mkhomedir_helper <1 - שם משתמש> <2 - הרשאות>
<3 - המיקום של קבצי הפרופיל הדיפולטיביים>

לדוגמא:

mkhomedir_helper myuser 0077 /etc/skel

הגדרת פוליסת סיסמא למשתמש

הסבר בקצרה:

ניתן להגדיר או להציג פוליסת סיסמא של משתמש כגון הגדרת / הצגת תוקף של סיסמא וכדומה.

ההגדרות מתבצעות באמצעות פקודה בשם chage.

למשל: על מנת לצפות בהגדרות של משתמש מסוים ניתן להשתמש בפקודה להלן שמשתמשת במתג l- כדי להציג (l = LIST) סטאטוס נוכחי של משתמש.

Ubuntu / CentOS

פקודה:

chage -l ohadm

פלט לדוגמא:

```
Last password change                                    : Jun 21, 2016
Password expires                                        : never
Password inactive                                       : never
Account expires                                         : never
Minimum number of days between password change          : 0
Maximum number of days between password change          : 99999
Number of days of warning before password expires       : 7
```

הגבלת משתמש למספר סופי של ניסיונות חיבור

הסבר בקצרה:

לפעמים נרצה להגביל את משתמשי המערכת למספר סופי של ניסיונות כושלים רצופים להתחבר אל המערכת למשל כמנגנון אבטחה נגד ניסיונות חיבור אוטומטיים.

לשם כך נוכל להשתמש במנגנון האימות pam.d ולשנות בו ערכים מתאימים.

נוכל להחליט האם לסגור זמנית את החשבון במקרה שבו המשתמש עבר את מספר הניסיונות או לסגור אותו לגמרי.

בדוגמא זו אנו נועלים את המשתמש ל-15 דקות לאחר שביצע 5 ניסיונות כושלים להתחבר עקב שימוש בפרטים מזהים לא נכונים.

CentOS

תהליך:

עורכים את קובץ הגדרות האימות הראשי שהינו /etc/pam.d/**system-auth** ומוסיפים מתחת לשורה שפונה למודול **pam_env.so** במקטע ההגדרות הראשון (auth) את השורה הבאה:

```
auth  required  pam_tally2.so  deny=5  onerr=fail  unlock_time=900
```

Ubuntu

תהליך:

עורכים את קובץ הגדרות האימות הראשי שהינו /etc/pam.d/**common-auth** ומבצעים את השינויים הבאים:

1. מאתרים את השורה:

```
auth requisite                    pam_deny.so
```

ומחליפים בה את המילה requisite במילה required כך שהשורה תיראה כך:

```
auth required                     pam_deny.so
```

2. מוסיפים בסוף הקובץ את השורה הבאה:

```
auth  required  pam_tally2.so  deny=5  onerr=fail  unlock_time=900
```

הערות:

- שינויים בהגדרות של pam.d חלים באופן מיידי על המערכת ואין צורך לבצע אתחול כלשהו בשום רמה.

פתיחת משתמש שננעל לאחר מספר רב מדיי של ניסיונות חיבור כושלים

הסבר בקצרה:

אם השתמשנו ביכולת להגביל משתמש למספר סופי של ניסיונות חיבור רצופים (ראו טיפ קודם) והמשתמש ננעל לאחר שהם אזלו נוכל להשתמש בפקודה pam_tally2 על מנת לשחרר את המשתמש מנעילה.

Ubuntu / CentOS

פקודה:

pam_tally2 --user=<שם משתמש> --reset

הערות:

- --reset שחרור המשתמש מנעילה. בפועל מדובר באיפוס מונה שנמצא בתוך קובץ מערכת שמכיל את מספר הניסיונות הכושלים הרצופים שהיו למשתמש.
- יתכן שנמצא גם את הפקודה בגרסתה הקודמת - pam_tally.

בדיקת סטאטוס של משתמש

הסבר בקצרה:

ניתן לקבל סטאטוס כללי על משתמש באמצעות הרצת הפקודה הבאה כ- ROOT:

passwd -S <שם משתמש>

פלט לדוגמא:

ohadm PS 2016-02-17 0 99999 7 -1 (Password set, MD5 crypt.)

השדה השני בפלט מציג את מצב הסיסמא של המשתמש:

L = הסיסמא נעולה

NP = אין סיסמא

PS = מוגדרת סיסמא (CentOS)

P = מוגדרת סיסמא (Ubuntu)

עזרים

חלק זה מציג רשימה של פקודות עזר שונות.

כל הפקודות בחלק זה קיימות ב-2 המשפחות של מערכות ההפעלה ולכן אין צורך לציין זאת עבור כל פקודה בנפרד.

openssl – ניהול תעודות והגדרות SSL

הפקודה openssl משמשת לניהול של כל הקשור ב-SSL. זה כולל יצירת תעודות ובקשות לתעודות, בדיקה חיבור ל- SSL, בדיקת CIPHERים ועוד.

להלן דוגמאות הממחישות את יכולותיה השונות של הפקודה.

דוגמאות שימוש:

הצגת הגרסה של OPENSSL

הפקודה:

```
openssl  version
```

הצגת נתונים אודות תעודה

הפקודה:

```
openssl  x509  -in  /path/to/cert/file   -noout  -issuer  -subject  -dates
```

הערות:

- אם יש בקובץ שמכיל את התעודה שרשור (CHAIN) של תעודות ROOT ואמצע הפקודה תציג רק את הפרטים של התעודה הראשונה שקיימת בקובץ.

בדיקת תקינות של תעודה (כולל CHAIN)

הפקודה:

```
openssl  s_client  -connect  gmail.com:443
```

ייצוא של תעודה מפורמט PEM ו- KEY ל- PFX (איחוד חלקי התעודה - המרה מלינוקס לווינדוס)

הפקודה:

```
openssl  pkcs12  -export  -in  /path/to/public/cert.crt \
-inkey  /path/to/private/key.key  -name  "cert friendly name" \
 -chain  -out  pfx_with_chain.pfx
```

הערות:

- יש לבחור סיסמא עבור התעודה שתיווצר היות שהיא תכיל גם מפתח פרטי.
- אם מדובר בתעודה מסוג SELF SIGNED אין לה CHAIN ולכן יש להוריד במקרה זה את המתג chain- מהפקודה אחרת תופיע שגיאה והתהליך יכשל.

הוצאת זוג קבצי תעודה - פרטית וציבורית - KEY ו- PEM מתוך קובץ PFX לשימוש על ידי שרת WEB מבוסס קוד פתוח כגון HTTPD (המרה מווינדוס ללינוקס)

<u>התהליך:</u>

1. שליפת המפתח הפרטי לקובץ:

```
openssl pkcs12 -in <שם קובץ ה- PFX>
-out <שם רצוי עבור הקובץ שיוצר שיכיל את המפתח הפרטי>
-nocerts -nodes -passin pass:<הסיסמא של קובץ ה- PFX>
```

<u>הערה:</u>

המתג nodes- גורם לכך שהמפתח הפרטי ייווצר ללא סיסמא. אם נריץ את הפקודה בלעדיו נתבקש לספק סיסמא עבור המפתח (ניתן להוסיף את המתג :passout pass- בצירוף הסיסמא של קובץ ה- PFX בדומה למתג passin- לעיל על מנת שהפעולה לא תיעצר עם בקשת הסיסמא - שימושי לסקריפטים).

2. שליפת המפתח הציבורי לקובץ:

```
openssl pkcs12 -in <שם קובץ ה- PFX>
-out <שם רצוי עבור הקובץ שיוצר שיכיל את המפתח הציבורי>
-nokeys -clcerts -passin pass:<הסיסמא של קובץ ה- PFX>
```

בדיקת יכולת חיבור בפרוטוקול ספציפי

<u>הפקודה:</u>

עבור בדיקה של הפרוטוקול ssl v3 (לוודא שהוא דווקא <u>לא</u> עובד כי הוא לא מאובטח):

```
openssl s_client -connect mysite.co.il:443 -ssl3 | more
```

עבור בדיקה של הפרוטוקול tls v1.2 (<u>שכן</u> עובד. זהו הפרוטוקול המומלץ לשימוש):

```
openssl s_client -connect mysite.co.il:443 -tls1_2 | more
```

<u>הערות:</u>

- אם התשובה מחזירה בראשה את התעודה של האתר, זה אומר שהחיבור עבד בהצלחה או במילים אחרות שכרגע האתר והשרת תומכים בפרוטוקול הזה.

בדיקה באילו תעודות אתר מסוים משתמש

<u>הפקודה:</u>

```
openssl s_client -connect mysite.co.il:443 \
-servername mysite.co.il -showcerts
```

הערות:

- -servername פרמטר זה שמגדירים בו את כתובת האתר, הכרחי כדי לקבל את התעודה הנכונה במקרה שהשרת מארח יותר מאתר SSL אחד ויותר מתעודת SSL אחת. במקרה זה, היות שיש לשרת יותר מתעודה אחת והחיבור מוצפן, גם הכתובת המבוקשת עצמה נשלחת מוצפנת על ידי הלקוח ואז אין יכולת לשרת לזהות באיזה אתר מדובר ואיזו תעודה להציג לו אלא אם כן נשלח בנפרד את הכתובת באופן בלתי מוצפן וזה מתבצע על ידי פרמטר זה. המונח המקצועי הוא SNI (קיצור של Server Name Indication).
- \ הפקודה ממשיכה בשורה הבאה.

בדיקה באיזה CIPHER אתר SSL משתמש כרגע

הפקודה:

```
openssl s_client -connect mysite.co.il:443 | grep -i cipher
```

בדיקת שורת CIPHERים ספציפית כדי לדעת איזה CIPHERים יתמכו על ידי שרת ה- WEB

הפקודה:

```
openssl ciphers -v

'ALL:!ADH:!EDH:!DH:!NULL:RSA:MEDIUM:!LOW:!SSLv2:!EXP:!SSLv3:TLSv1'
```

בדיקת תאריך תפוגה של תעודה בעקיפין דרך האתר שהיא מוטמעת בו ולא דרך שמה ישירות

הפקודה:

```
echo "" | openssl s_client -connect mysite.co.il:443 2>/dev/null \
| openssl x509 -noout -dates
```

פלט לדוגמא:

```
notBefore=Jul 13 13:28:41 2016 GMT
notAfter=Oct 5 13:17:00 2016 GMT
```

הערות:

- echo "" בחלק מהגרסאות של OPENSSL, פקודת החיבור s_client מצפה לקלט מהמשתמש ולכן בגרסאות אלה פקודה מסוג זה של openssl שמהווה את החלק הראשון בדוגמא לעיל תיתקע אם לא נשלח אליה משהו. לכן נמצאת כאן פקודת echo זו לפניה.
- openssl s_client -connect mysite.co.il:443 החלק הראשון של הפקודה. מחזיר פרטי תעודה של אתר. התחביר של המתג -connect הוא **בהכרח** בתצורה של url:port ולא ניתן להשתמש בתצורה אחרת כגון https://url.

- 2>/dev/null התעלמות משגיאות.
- openssl x509 -noout -dates חלק זה שהינו החלק השני של הפקודה, מקבל מהאתר הנבדק דרך החלק הראשון של הפקודה את פרטי התעודה ומחזיר את טווח התאריכים של התעודה (יש גם תאריך תוקף מינימלי - כלומר "החל מ" - וגם תאריך תוקף מקסימלי - כלומר "ועד").

יצירת קובץ בקשה (קובץ CSR) לשליחה ל-CA לטובת הוצאת תעודה מוכרת

<u>התהליך:</u>

1. יצירת מפתח פרטי עבור הבקשה:

```
openssl genrsa -aes256 -out /tmp/my_csr_key 2048
```

יש לבחור סיסמא עבור המפתח שהפקודה יוצרת.

2. יצירת פרטי הבקשה עצמה:

```
openssl req -new -key /tmp/my_csr_key -out /tmp/my_csr
```

לאחר הרצת פקודה זו נצטרך להכניס את הסיסמא שהגדרנו בשלב 1 לעיל ואת פרטי הבקשה עצמה.

לדוגמא:

Country Name (2 letters) = IL
State = <ריק>
City = Jerusalem
Organization Name = myOrganization
Organizational Unit Name = <ריק>
Common Name (CN) = **mysite.org.il**
Email = <ריק>
A challenge password [] = <ריק>
An optional company name [] = <ריק>

השדה החשוב ביותר בפרטים הוא ה- CN שבו מגדירים את כתובת האתר שאליה התעודה תשויך.

את קובץ ה- CSR שנוצר בסיום התהליך שולחים ל- CA מוכר לחתימה.

<u>הערה:</u>

ניתן לוודא שהבקשה נוצרה בהצלחה על ידי הרצת הפקודה:

```
openssl req -noout -text -in /tmp/my_csr
```

פלט תקין יכיל את פרטי הבקשה עם הפרטים שבחרנו.

יצירת תעודה פנימית בעלת חתימה עצמית לצורך בדיקות (SELF SIGNED CERTIFICATE):

התהליך:

1. מריצים את הפקודה:

```
openssl req -x509 -newkey rsa:2048 -keyout /tmp/my_cert.key \
-out /tmp/my_cert.pem -days 365
```

יש לבחור סיסמא עבור המפתח שהפקודה יוצרת.

ממלאים את הפרטים שהפקודה מבקשת (ראו דוגמא בסעיף הקודם "יצירת קובץ בקשה (קובץ CSR) לשליחה ל-CA לטובת הוצאת תעודה מוכרת")

בסיום התהליך נקבל את התעודה בשני חלקים: המפתח הציבורי יופיע בקובץ my_cert.pem והמפתח הפרטי יופיע בקובץ my_cert.key.

הערה:

ניתן להסיר את הסיסמא מהמפתח הפרטי ולשמור אותו ללא סיסמא בקובץ חדש על ידי הרצת הפקודה:

```
openssl rsa -in /tmp/my_cert.key -out /tmp/my_cert_no_pwd.key
```

הצפנה ופענוח של נתונים עם OPENSSL

ניתן באמצעות OPENSSL להצפין נתונים באמצעות סיסמא ולפענח אותם מחדש.

באופן זה ניתן להשתמש למשל במערכת שדורשת סיסמת כניסה באופן אוטומטי תוך שימוש בסקריפט מבלי לחשוף את הסיסמא האמתית של המערכת אלא רק ערבול שלה.

הצפנה:

```
echo <סיסמא אמתית> | \
openssl aes-256-cbc -e -a -salt -k <סיסמת פענוח> \
> hashed-pwd.txt
```

הערות:

- <סיסמא אמתית> סיסמא או כל תוכן אחר שנרצה להצפין.
- aes-256-cbc אלגוריתם ההצפנה.
- -e מתג שמגדיר ל- openssl להצפין את הנתונים שנשלחו אליו.
- -a מתג שמגדיר ל- openssl להצפין את הנתונים בפורמט של base64.
- -salt -k <סיסמת פענוח> הגדרת סיסמא שבאמצעותה נוכל לפענח את ההצפנה.
- בדוגמא זו שלחנו נתונים להצפנה באמצעות פקודת echo אך כמובן שניתן לשלוח נתונים ל- openssl גם באופנים אחרים.

פענוח:

```
cat hashed-pwd.txt | openssl aes-256-cbc -a \
-d -salt -k <סיסמת פענוח>
```

הערות:

- hashed-pwd.txt זוהי סיסמא הפענוח שהגדרנו לעיל בשלב ההצפנה. בדוגמא שמרנו אותה לקובץ אך כמובן שאין זו חובה ונוכל לשמור אותה היכן שנרצה.
- aes-256-cbc אלגוריתם ההצפנה.
- -a מתג שמגדיר ל- openssl לפענח את הנתונים בפורמט של base64.
- -d פענח את הנתונים שמתקבלים.
- -salt -k <סיסמת פענוח> השתמש בסיסמא זו על מנת לפענח את הנתונים המוצפנים.
- הפלט של פקודה זו יהיה הסיסמא האמיתית.

watch - הרצת פקודה באופן מחזורי

הפקודה watch מאפשרת להריץ בלולאה באופן מחזורי פקודה.

האפשרות הזו מאוד שימושית כאשר רוצים לנטר תהליך או תיקייה.

כברירת מחדל הפקודה תרוץ מחדש כל 2 שניות. כמובן שניתן להגדיר זמן אחר (כולל שימוש בשברים לציון חלקי שנייה כגון 0.1 וכדומה).

דוגמאות שימוש:

ניטור תיקייה

הפקודה:

```
watch "ls -l"
```

הערות:

- הפקודה ls -l במקרה הזה תרוץ פעם ב 2 שניות היות שלא צוין אחרת.
- חשוב לזכור לתחום את הפקודה שרוצים להריץ בין גרשיים כפולים כי אחרת פקודות מורכבות עשויות לא לעבוד כראוי.

ניטור מספר הקבצים בתיקייה

הפקודה:

```
watch -d "ls | wc -l" (L קטנה)
```

הערות:

- -d הצג את השינויים בזמן אמת על ידי סימונם בפלט.

ניטור פלט של פקודת mysql

הפקודה:

```
watch -n 30 "mysql -uroot -e \"show processlist;\""
```

הערות:

- -n 30 הרץ את הפקודה כל 30 שניות.
- mysql -e המתג e- מאפשר להריץ שאילתה על ה- DB מבחוץ (ישירות מה-SHELL).
- \"show processlist;\" שאילתה זו מציגה את התהליכים שרצים כרגע בשרת MYSQL. שימו לב שהיא מסתיימת בתו נקודה פסיק - כמו כל שאילתה. היות שהיא גם צריכה להיות תחומה בין גרשיים (היות שהיא מורכבת מיותר ממילה אחת) יש צורך ב- ESCAPE של הגרשיים באמצעות סלאשים הפוכים על מנת ש- Bash לא יפרש אותם בעצמו ואז הפקודה לא תעבוד כראוי.

scp - העתקת קבצים מרוחקת

הפקודה scp מקילה מאוד על העתקת קבצים בין מחשבים ברשת. כל מה שצריך זה שרת SSH באחד מהצדדים וזהו.

ניתן לנצל את הפקודה הזו מכל מיני כיוונים כמו למשל העתקת קבצים ביעילות ובקלות ממערכת לינוקס ללא GUI אל מערכת ווינדוס או העתקת קבצים ממכונה וירטואלית המריצה לינוקס ללא צורך בהתקנה והגדרה של הערכה שמקשרת בין ה- HOST ל- GUEST (כגון vmware tools או virtualbox guest additions).

המבנה של הפקודה זהה למבנה של חיבור רגיל ב- ssh.

מבנה כללי:

העתקת קובץ ממקור מקומי ליעד מרוחק

scp <שילוב של נתיב וקובץ או ביטוי רגולרי> <משתמש>@<כתובת של שרת>:<יעד>

העתקת קובץ ממקור מרוחק ליעד מקומי

scp <משתמש>@<כתובת של שרת>:<שילוב של נתיב וקובץ או ביטוי רגולרי> <יעד>

חשוב לשים לב לתו נקודתיים (:) שמפריד בין שם המחשב לבין הנתיב שנרצה לעבוד אתו.

דוגמאות:

.1

```
scp /etc/*.conf root@172.22.1.1:/tmp
```

דוגמא זו תעתיק ממחשב מקומי את כל הקבצים שמסתיימים במילה conf ונמצאים בתיקייה etc/ אל מחשב מרוחק שכתובתו היא 172.22.1.1 לתוך התיקייה tmp/ שלו.

לאחר הרצת הפקודה נתבקש להכניס את הסיסמא של המשתמש root המרוחק.

.2

```
scp ohadm@192.168.5.5:/etc/skel/.bashrc .
```

דוגמא זו תעתיק ממחשב מרוחק שכתובתו היא 192.168.5.5 את הקובץ etc/skel/.bashrc/ אל התיקייה הנוכחית המקומית (המצוינת באמצעות התו "./").

לאחר הרצת הפקודה נתבקש להכניס את הסיסמא של המשתמש ohadm המרוחק.

כמובן שלמשתמש ohadm בדוגמא זו צריכות להיות הרשאות גישה מתאימות אל הקובץ.

הערה:

אם מכונת הלקוח היא ווינדוס, נוכל להשתמש בתוכנה הגרפית החינמית WinSCP שתאפשר לנו להעתיק קבצים בקלות ובלי להסתבך.

time - מדידת זמן לפקודה

הפקודה time מודדת כמה זמן לוקח לפקודה לרוץ.

דוגמאות שימוש:

תזמון של פקודת העתקה

הפקודה:

```
time cp -R /etc/ .
```

פלט:

```
real    0m5.667s
user    0m0.006s
sys     0m0.755s
```

שמירת תזמון של פקודה לקובץ

הפקודה:

```
{ time ls;} 2>/tmp/time-output.txt
```

הערות:

- { time ls;} הרווח בתחילת הפקודה הוא חובה. תו הנקודה פסיק בסוף הפקודה הוא גם חובה.
- {} time רצה בתת-תהליך. על מנת שנוכל לתפוס את הפלט שלה, צריך לתחום את הפקודה כולה (אותה עצמה ואת הפקודה שהיא מודדת) בין סוגריים מסולסלים שמגדירים ל- Bash להפעיל את הפקודה באותו תהליך.
- 2>/tmp/time-output.txt time שולחת את הפלט שלה לערוץ השגיאות STDERR. לכן על מנת להעביר את הפלט שלה לקובץ, נדרש לנתב את השגיאות (או בשמו הנרדף - 2) אליו.
- על מנת לשמור גם את הפלט של הפקודה עצמה וגם את הפלט של time לקובץ, צריך לנתב את כל הפלט לקובץ על ידי החלפת המספר 2 בתו & (&>file).

strace - הצגת כל הפעולות במערכת שפקודה נתונה מבצעת

הפקודה strace היא פקודה שמאוד חשוב להכיר.

היא מציגה את כל מה שפקודה שמריצים דרכה מבצעת במערכת. כולל קבצי נתונים וקבצי מערכת שהפקודה משתמשת בהם, פניות דרך הרשת, קריאות מערכת וסטאטוסים של הפעולות.

באמצעותה ניתן לדבג פקודות וכלים שלא עובדים כמצופה על ידי ניתוח הפלט של strace לאחר שהופעלה עליהם.

דוגמאות שימוש:

הצגת פלט מהפקודה curl

הפקודה:

```
strace curl google.com
```

פלט (חלקי - הפלט המלא תופס כ- 10 עמודים):

```
execve("/usr/bin/curl", ["curl", "google.com"], [/* 36 vars */]) = 0
brk(0)                                  = 0x1703000
mmap(NULL, 4096, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fcc028c7000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=51521, ...}) = 0
mmap(NULL, 51521, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fcc028ba000
close(3)                                = 0
open("/usr/lib64/libcurl.so.4", O_RDONLY) = 3
```

```
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\20\343 [6\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=350872, ...}) = 0
mmap(0x365b200000, 2444072, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x365b200000
mprotect(0x365b252000, 2097152, PROT_NONE) = 0
mmap(0x365b452000, 12288, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x52000) = 0x365b452000
close(3)                                = 0
open("/lib64/libidn.so.11", O_RDONLY)   = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0/\340J6\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=209120, ...}) = 0
mmap(0x364ae00000, 2301768, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x364ae00000
mprotect(0x364ae32000, 2093056, PROT_NONE) = 0
mmap(0x364b031000, 4096, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x31000) = 0x364b031000
close(3)                                = 0
socket(PF_INET6, SOCK_DGRAM, IPPROTO_IP) = 3
connect(3, {sa_family=AF_INET6, sin6_port=htons(80), inet_pton(AF_INET6,
"2a00:1450:4001:816::200e", &sin6_addr), sin6_flowinfo=0, sin6_scope_id=0}, 28) =
-1 ENETUNREACH (Network is unreachable)
close(3)                                = 0
socket(PF_INET, SOCK_STREAM, IPPROTO_TCP) = 3
setsockopt(3, SOL_SOCKET, SO_KEEPALIVE, [1], 4) = 0
fcntl(3, F_GETFL)                       = 0x2 (flags O_RDWR)
fcntl(3, F_SETFL, O_RDWR|O_NONBLOCK)    = 0
connect(3, {sa_family=AF_INET, sin_port=htons(80),
sin_addr=inet_addr("216.58.210.14")}, 16) = -1 EINPROGRESS (Operation now in
progress)
poll([{fd=3, events=POLLOUT|POLLWRNORM}], 1, 149991) = 1 ([{fd=3,
revents=POLLOUT|POLLWRNORM}])
getsockopt(3, SOL_SOCKET, SO_ERROR, [0], [4]) = 0
sendto(3, "GET / HTTP/1.1\r\nUser-Agent: curl"..., 173, MSG_NOSIGNAL, NULL,
0) = 173
poll([{fd=3, events=POLLIN|POLLPRI|POLLRDNORM|POLLRDBAND}], 1, 0) = 0
(Timeout)
poll([{fd=3, events=POLLIN|POLLPRI|POLLRDNORM|POLLRDBAND}], 1, 0) = 0
(Timeout)
```

```
poll([{fd=3, events=POLLIN|POLLPRI|POLLRDNORM|POLLRDBAND}], 1, 1000)
= 1 ([{fd=3, revents=POLLIN|POLLRDNORM}])
poll([{fd=3, events=POLLIN|POLLPRI|POLLRDNORM|POLLRDBAND}], 1, 0) = 1
([{fd=3, revents=POLLIN|POLLRDNORM}])
recvfrom(3, "HTTP/1.1 302 Found\r\nCache-Contro"..., 16384, 0, NULL, NULL) =
477
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fcc028c6000
write(1, "<HTML><HEAD><meta http-equiv=\"co"..., 79<HTML><HEAD><meta
http-equiv="content-type" content="text/html;charset=utf-8">
) = 79
write(1, "<TITLE>302 Moved</TITLE></HEAD><"..., 38<TITLE>302
Moved</TITLE></HEAD><BODY>
) = 38
write(1, "<H1>302 Moved</H1>\n", 19<H1>302 Moved</H1>
)    = 19
write(1, "The document has moved\n", 23The document has moved
) = 23
write(1, "<A HREF=\"http://www.google.co.il"..., 86<A
HREF="http://www.google.co.il/?gfe_rd=cr&amp;ei=B3N2V5qQLbPb8AfY44KgBA"
>here</A>.
) = 86
write(1, "</BODY></HTML>\r\n", 16</BODY></HTML>
)      = 16
close(3)                          = 0
exit_group(0)                     = ?
+++ exited with 0 +++
```

<u>הערות:</u>

- הפלט של strace הוא גדול ברוב המקרים והוא נשלח לערוץ השגיאות ולכן לא ניתן לעצור אותו עם הפקודה more. כדי לנתח פלט גדול ניתן לשמור אותו לקובץ במקום להציגו.

שמירת פלט של תהליך הפעלת סרוויס לקובץ

<u>הפקודה:</u>

```
strace  -o  /tmp/strace-output.txt  /etc/init.d/httpd start
```

<u>הערות:</u>

- -o /tmp/strace-output.txt שמירת הפלט לקובץ במקום למסך.

הגדרת גודל ספציפי לפלט טקסטואלי ש strace מדפיסה

הפקודה:

```
strace -o /tmp/strace-output.txt -s 512 myCommand
```

הערות:

- -s 512 הצגת 512 תווים ראשונים מכל שדה טקסט. כברירת מחדל, כאשר strace מאתרת פלט של פקודה שהיא הריצה כטקסט, היא מדפיסה רק את 32 התווים הראשונים שלו. על מנת להגדיר גודל גדול יותר כדי שיהיה לנו יותר ברור מה קרה במהלך הריצה של הפקודה שרצינו לנתח, נוכל להשתמש במתג -s של strace ולציין את הגודל הרצוי החדש בתווים.

file - מציאת סוג של קובץ

הפקודה file היא פקודה שימושית ונוחה למדיי. היא מאפשרת לזהות קובץ לפי סוגו.

הפקודה קוראת את הקובץ ומזהה אותו על פי תוכנו.

היא עוזרת מאוד במיוחד בסביבת לינוקס שבה אין מוסכמה לסיומות קבצים ולכן לרוב לא ניתן לדעת מהו סוג הקובץ לפי שמו, אך לאו דווקא. היא שמישה אפילו בסביבת ווינדוס (ניתן להשתמש בה בווינדוס דרך התוכנה CYGWIN) ורצוי מאוד להכיר אותה.

הפקודה:

file <קובץ>

דוגמאות פלט:

```
[root@localhost d]# file myFile
myFile: Bourne-Again shell script text executable

[root@localhost d]# file list.gz
list.gz: gzip compressed data, was "list", from Unix, last modified: Fri Jun
17 04:15:27 2016

[root@localhost lib64]# file libz.so.1
libz.so.1: symbolic link to 'libz.so.1.2.3'

[root@localhost lib64]# file /etc/
/etc/: directory

[root@localhost lib64]# file /sbin/chkconfig
/sbin/chkconfig: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
 dynamically linked (uses shared libs), for GNU/Linux 2.6.18, stripped
```

```
[root@localhost lib64]# file 'which yum'
/usr/bin/yum: a /usr/bin/python script text executable

[root@localhost etc]# file hosts
hosts: ASCII text
```

sort - מיון נתונים

הפקודה sort כשמה כן היא - תפקידה הוא למיין נתונים.

דוגמאות שימוש:

נניח שיש לנו את הקובץ הבא:

```
[root@localhost d]# cat list
4 four
1 one
3 three
2 two
```

נוכל למיין את הרשימה באמצעות הפקודה sort:

מיון רגיל:

```
[root@localhost d]# cat list | sort
1 one
2 two
3 three
4 four
```

מיון הפוך:

```
[root@localhost d]# cat list | sort -r
4 four
3 three
2 two
1 one
```

הערות:

- ניתן למיין רשימת גדלים של קבצים על ידי שימוש במתג -h . הוא יגרום לפקודה לזהות ביטויים כגון M (מגה), G (ג'יגה), T (טרה) וכדומה וימיין את הרשימה בהתאם לפי הגדלים שהאותיות לעיל מציינות.
- הפקודה sort חיונית עבור הפקודה uniq שמוחקת שורות מיותרות ובלעדיה הפקודה uniq לא תעבוד. ראו הסבר אודות הפקודה uniq בסעיף הבא.
- ניתן להשתמש במתג -u של sort במקום לעבוד עם uniq.

uniq - מחיקת שורות כפולות

הפקודה uniq מחזירה פלט שאינו חוזר על עצמו על ידי מחיקת שורות כפולות. על מנת שהיא תוכל לזהות את השורות הכפולות, חייבים להריץ על הנתונים לפי הקריאה לה את הפקודה sort שתמיין אותם קודם. לאחר המיון שורות כפולות יופיעו צמודות אחת לשנייה ורק אז הפקודה uniq תוכל לזהות ולמחוק אותן.

דוגמאות שימוש:

נניח שיש לנו את הקובץ הבא:

```
[root@localhost d]# cat list
2 two
4 four
1 one
3 three
1 one
2 two
3 three
2 two
4 four
```

אם נריץ את הפקודה uniq על הרשימה לעיל ללא מיון מקדים, לא יהיה כל שינוי בפלט:

```
[root@localhost d]# cat list | uniq
2 two
4 four
1 one
3 three
1 one
2 two
3 three
2 two
4 four
```

אבל אם ננסה עם מיון נקבל:

```
[root@localhost d]# cat list | sort | uniq
1 one
2 two
3 three
4 four
```

zcat, zless, zgrep, zmore, zcmp, zdiff - ניהול תוכן של קבצים שנמצאים בתוך קבצים מכווצים ללא צורך לפתוח קודם את הכיווץ

סט הפקודות לעיל מאפשר לנהל קבצים שנמצאים בתוך קבצי gz מכווצים כאילו שהם כבר בחוץ. זה מאוד נוח ומונע סירבול מיותר בעבודה עם קבצים מכווצים.

דוגמאות שימוש:

נניח שיש לנו את 2 קבצי ה- gz הבאים:

```
[root@localhost files]# ls
list1.gz  list2.gz

[root@localhost files]# gzip  -l  list1.gz
        compressed        uncompressed  ratio uncompressed_name
              54                  28  21.4% list1

[root@localhost files]# gzip  -l  list2.gz
        compressed        uncompressed  ratio uncompressed_name
              45                  19  31.6% list2
```

בכל קובץ מכווץ יש קובץ טקסט.

באמצעות הפקודות לעיל נוכל לקרוא את קבצי הטקסט הפנימיים ללא צורך בפתיחת הכיווץ.

למשל:

```
[root@localhost files]# zcat  list1.gz
2 two
4 four
1 one
3 three

[root@localhost files]# zcat  list2.gz
abc

def

ghi

jkl
```

או:

```
[root@localhost files]# zgrep  "one"  list1.gz
1 one
```

וכן הלאה עבור שאר הפקודות.

אגב - לשם השוואה, אם ננסה להריץ פקודה "רגילה" על קובץ מכווץ נקבל משהו כזה:

```
[root@localhost files]# cat  list1.gz
QO————————————————list13R()►►2QH►/-►2T►►K►
2V(►(JM►►►  ►=
```

מעניין למה ....

tcpdump - הצגת נתונים שעוברים ברשת בזמן אמת

הפקודה tcpdump מאפשרת להאזין לכרטיסי הרשת של המערכת בזמן אמת ולהציג או לשמור את הנתונים שעוברים ברשת.

כך ניתן לנתח בעיות תקשורת.

דוגמאות שימוש:

שימוש בסיסי

הפקודה:

```
tcpdump  -i  any
```

הערות:

- -i מתג שמגדיר לאיזה כרטיסי רשת להאזין.
- הרצת הפקודה ללא פרמטרים כלל תציג רק נתונים שנשלחים לכל הרשת כלומר חבילות מידע (Packets, פאקטות) שנשלחות ב- BROADCAST כגון בקשות ARP).

סינון לפי פורט

הפקודה:

```
tcpdump  -i  any  port  443  -n
```

הערות:

- port מילה שמגדירה לפי איזה פורט לסנן. בדוגמא רק מידע שמגיע דרך פורט מספר 443 יוצג.
- -n מתג שמגדיר לפקודה לא להמיר כתובות לשמות וכך למעשה להציג כתובות IP תמיד גם אם יש מיפוי לשם ב- DNS או בקובץ hosts.

סינון לפי שרת

הפקודה:

```
tcpdump  -i  any  host  192.168.1.2  -nn
```

הערות:

- host מילה שמגדירה לפי איזו כתובת לסנן. בדוגמא רק מידע שמגיע דרך הכתובת 192.168.1.2 יוצג.
- nn- מתג שמגדיר לפקודה גם לא להמיר כתובות לשמות וכך למעשה להציג כתובות IP תמיד גם אם יש מיפוי לשם ב- DNS או בקובץ hosts וגם לא המיר מספרי פורטים ידועים לשמותיהם (כגון 443 ל- "https" וכדומה) באמצעות הקובץ /etc/services .

שמירת המידע לקובץ

הפקודה:

```
tcpdump -i any -w /tmp/capture.pcap
```

הערות:

- w- /tmp/capture.pcap שמירת הנתונים לקובץ. הפורמט של הקובץ הוא pcap והוא קריא על ידי התוכנה WIRESHARK החינמית שיש לה ממשק גרפי ואז הרבה יותר נוח לנתח אתה את התעבורה ששמרנו.

פלט מורחב, סינון לפי כרטיס רשת ספציפי, כתובת ספציפית ופורט על דרך השלילה

הפקודה:

```
tcpdump -i eth0 host 10.10.10.1 and port not 22 -vvv
```

הערות:

- vvv- פלט מורחב. קיצור של very very verbose. ניתן להשתמש גם ב- vv- או ב- v- לפחות נתונים בהתאמה.
- and שילוב של 2 תנאים. גם הגבלת הפורט וגם הגבלת הכתובת.
- not הגדרת תנאי הפוך בשלילה.

סינון לפי כתובת מקור, כתובת יעד ופרוטוקול

הפקודה:

```
tcpdump -i any src host 192.168.40.128 and dst 8.8.8.8 and udp
```

הערות:

- src סינון לפי כתובת מקור ספציפית.
- dst סינון לפי כתובת יעד ספציפית.
- udp סינון לפי פרוטוקול UDP (הצגת נתונים שנשלחים ב- UDP בלבד).

הצגת נתונים ב- ASCII - הצגת פאקטות כטקסט - יעיל לניתוח בעיות מול שרתי WEB

הפקודה:

```
tcpdump -i any -A
```

הערות:

- -A הצגת המידע כ- ASCII. יעיל לניתוח מידע שנשלח או מתקבל על ידי שרתי WEB. מתג זה יאפשר את הצגת כל התוכן של הבקשות שנשלחות בפרוטוקול HTTP.

stat - הצגת נתונים מורחבים על קבצים ותיקיות

הפקודה stat מאפשרת לקבל על קבצים ותיקיות מידע מורחב יותר מהמידע שמתקבל דרך הפקודה ls.

ניתן להשתמש בה למשל כאשר רוצים לקבל את ההרשאות של קובץ או תיקייה בתצורה של מספרים (0777) ולא אותיות (rwxr-xr-x) כדי לעבד אותן בקלות או לשימוש בתוך סקריפטים.

הפקודה מאפשרת לקבל פלט ספציפי על פי רשימת שדות וניתן לבחור איזה שדות שרוצים (בדומה לפקודה date שעושה את אותו הדבר עבור תאריכים).

ניתן לראות את רשימת השדות על ידי עיון בפלט של פקודת העזרה שלה (stat --help).

דוגמאות שימוש:

הצגת מידע מורחב על קובץ

הפקודה:

```
stat /etc/sudoers
```

פלט:

```
  File: `/etc/sudoers`
  Size: 4002        Blocks: 8          IO Block: 4096   regular file
Device: 802h/2050d       Inode: 398210      Links: 1
Access: (0440/-r--r-----)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2016-06-30 09:46:43.044020647 -0700
Modify: 2012-03-01 09:18:24.000000000 -0800
Change: 2016-02-18 13:22:13.229999772 -0800
```

הצגת תאריך גישה אחרון ותאריך שינוי אחרון של תיקייה

הפקודה:

```
stat -c '%x %z %n' /etc
```

פלט (מופיע על המסך בשורה אחת):

```
2016-07-01 06:49:30.544006324 -0700 2016-07-01
04:42:56.974007182 -0700 /etc
```

הערות:

- c- פורמט רצוי - לאחר מתג זה בוחרים נתונים שרוצים להציג מתוך רשימה של נתונים. הרשימה המליאה נמצאת בפרטי העזרה (stat --help) של הפקודה. בחירה מרובה של נתונים יש לתחום בין גרשיים בודדים או כפולים כמו בדוגמא.
- %x תאריך גישה אחרון
- %z תאריך שינוי אחרון
- %n שם הקובץ

wc - מניית שורות, תווים ומילים

הפקודה wc מונה שורות, תווים ומילים בקובץ.

היא שימושית למשל כשנרצה לבדוק האם קיבלנו תוצאות כלשהן מפקודה מסוימת ואז במילים אחרות הפלט שלה יכיל יותר משורה אחת.

שימוש לדוגמא:

```
ps -ef | grep -v grep | grep httpd | wc -l
```

בדוגמא זו מתבצעת בדיקה האם סרויס מסוים פעיל או לא על ידי ספירת העותקים שמופיעים בפלט של הפקודה ps.

הערות:

- grep -v grep אם נחפש בפלט של ps את המילה "httpd", במינימום נקבל לעולם לפחות תוצאה אחת - של ה- GREP עצמו שלפני שנייה הרצנו. על מנת לקבל תוצאה נכונה של מספר העותקים של httpd נוכל להחריג את פקודת ה- grep עצמה מרשימת התוצאות של ps (באמצעות grep ...) ואז נקבל תוצאה נכונה.
- wc -l ספירת והצגת מספר השורות בלבד של הפלט הקודם. בדוגמא זו נקבל את הערך 0 אם אין עותקים של httpd שרצים כעת. אחרת נקבל מספר גבוה מ 0.

reset - איפוס ה- SHELL

הפקודה reset מאפסת את המראה של ה- SHELL.

לעתים הוא עשוי להשתבש מכל מיני סיבות. למשל כתוצאה מניסיון להציג כטקסט נתונים בינאריים, מהרצה של פקודה לא תקינה או משימוש בצבעים בפקודות או בקוד שבגלל מבנה לקוי או לא מדויק לא מאפסות את הצבע לאחר הרצה ואז כל ה- SHELL ממשיך להיות צבוע כולל כל הקלט והפלט.

לשם כך בדיוק נועדה פקודה זו. היא לא מפעילה מחדש את המחשב או אפילו את Bash. היא פשוט מאפסת לו את ההגדרות ומחזירה לו את המראה הדיפולטיבי שלו.

אין לפקודה פרמטרים מיוחדים.

הפקודה:

```
reset
```

iconv - המרת קבצים לקידודים שונים

הפקודה מאפשרת לבצע המרות של קידודים שונים לקבצים.

הפקודה:

```
cat myFile.php | iconv -f WINDOWS-1255 -t utf8 > myFile.php.utf8
```

הערות:

- -f הקידוד הנוכחי של הקובץ (FROM).
- -t הקידוד הרצוי (TO).

nl - הצגת מספרי שורות לקבצים

הפקודה מציגה את הקובץ הרצוי עם מספרי שורות לצדו להתמצאות נוחה.

nl זה ראשי תיבות של "Line Numbers" אבל הפוך ...השם ln כבר תפוס על ידי הפקודה שיוצרת לינקים.

הפקודה:

```
nl <קובץ>
```

הערות:

- הפקודה לא סופרת שורות ריקות. יש לשים לב לכך ולא לבנות על המספור שלה כשרוצים למשל להוסיף לקובץ שורה חדשה לאחר שורה מסוימת כי המספור יהיה לא מדויק אם יש בקובץ שורות ריקות.

כללי - כל מיני אחרים

כל הפקודות בחלק זה קיימות ב- 2 המשפחות של מערכות ההפעלה ולכן אין צורך לציין זאת עבור כל פקודה בנפרד.

cal - לוח שנה מבוסס טקסט

ישנו לוח שנה טקסטואלי מאוד חביב שניתן להפעיל ישירות משורת הפקודה. ניתן להשתמש בו למשל כשנרצה לדעת כמה ימים יש בחודש הנוכחי בקלות.

הפקודה:

cal

פלט:

```
      June 2016
Su Mo Tu We Th Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30
```

rev - הפיכת סדר של תווים

הפקודה הופכת את סדר התווים של קלט שנכנס אליה.

הפקודה:

echo "reversing chars …" | rev

פלט:

… srahc gnisrever

shuf - ערבוב נתונים

הפקודה מערבבת נתונים (קיצור של המילה SHUFFLE)

הפקודה:

ls | shuf

בכל פעם שנריץ את הפקודה לעיל, הפלט של הפקודה ls יתערבב מחדש ויציג את התוצאות בסדר אחר.

Bash Scripts

כללי - מספר עקרונות בפיתוח וכתיבת קוד

במישור התיאורטי

1. **REUSE**

- לעולם אל תחזרו פעמיים על אותו קוד. בין אם זה ביחס לסקריפט בודד ובין אם זה ביחס לכל הסקריפטים שיש לכם גם יחד.

2. **PARAMETERS**

- השתמשו תמיד בפרמטרים. בין אם להגדיר ערכים קבועים שעלולים להשתנות בעתיד בין אם להגדיר ערכים שמציינים הגדרות ואפילו עבור שמות של פקודות (היות שהנתיבים משתנים במערכות שונות). לעולם אל תכתבו בתוך הקוד עצמו (בגוף הסקריפט) ערך אבסולוטי אלא רק שמות של פרמטרים שהגדרתם בראש הסקריפט.

3. **VALIDATION**

- היכן שרק ניתן, בדקו את תקינות הנתונים שאתם עובדים איתם בתוך הסקריפט. בין אם זה קלט שמגיע מבחוץ על ידי המשתמש, בין אם זה נתון שנוצר תוך כדי הרצת הסקריפט ובין אם זו הגדרה פנימית כלשהי בתוך הסקריפט. הגדירו גם תהליכים לתיקון עצמי כאשר זה אפשרי. הקפדה על כלל זה תצמצם סיכויים לשגיאות ובעיות בעת הפעלת הסקריפט בסביבות שונות.

4. **DOCUMENTATION**

- לתעד, לתעד ושוב לתעד. תיעוד של הקוד שלכם הוא אולי הדבר החשוב ביותר להקפיד עליו. ככל שהקוד יהיה יותר קריא וברור, יהיה יותר קל לעדכן אותו בעת הצורך ולשתף אותו עם חברי צוות אחרים. התיעוד כולל הסברים בתוך הקוד וגם הגדרות ושמות של משתנים ושל פונקציות בעלי משמעות. ככל שהקוד עצמו יהיה יותר ברור ומובן כן יקטן הצורך בתיעוד מילולי.
- היו עקביים עם מוסכמות פיתוח. למשל: אם החלטתם לקרוא בשם מסוים למשתנה שיש לו תפקיד שחוזר על עצמו, המשיכו עם אותו שם גם בסקריפטים הבאים. אם הגדרתם משתנה בצורה מסוימת שעשויה לחזור על עצמה, הקפידו על אותה צורה גם בפעמים הבאות. כנ"ל לגבי מבנה הסקריפט, ניהול שגיאות וכל חלק אחר בקוד. הקפדה מסוג זה תעזור לכם ליצור תשתית יציבה ואמינה עם התנהגות קבועה.

5. **FLEXIBILITY**

- נסו ליצור סביבה גמישה כמה שיותר שתהיה פתוחה לשינויים ועדכונים עתידיים. חשבו קדימה ובאופן גנרי (כללי, מורחב, מלמעלה) כאשר אתה יוצרים קוד חדש.

במישור המעשי

1. REUSE

- חשבתם על רעיון לסקריפט חדש? נסו לפצל אותו לכמה שיותר תת-סקריפטים כאשר סקריפט ראשי קורא להם, וכל אחד מהם (כולל הסקריפט הראשי) מחזיר בסיומו קוד (0 להצלחה ומספר גבוה מ-0 לכישלון). כל תת-סקריפט יבצע פעולה ספציפית מאוד (לאו דווקא במובן של מספר שורות אלא רעיונית. לדוגמא סקריפט שמעתיק קבצים ממקום למקום יקרא לתת-סקריפט שמעדכן הרשאות ולעוד תת-סקריפט שמגבה את הקבצים המקוריים). באופן זה, סקריפטים מורכבים יבדקו שתת-סקריפטים הסתיימו בהצלחה לפני שימשיכו לרוץ ולא תהיה חזרה על קוד.
- אם מדובר בתהליך קטן שאיננו מצריך סקריפט נפרד, צרו עבורו פונקציה וכך כל הקוד שלכם באופן כללי יהיה מודולארי ותוכלו ליצור סקריפטים חדשים בקלות על ידי פירוקם למודולים רעיוניים שתצטרכו ברוב המקרים לממש רק חלק קטן מהם כי בד"כ הפעולות חוזרות על עצמן ולכן רוב הסיכויים שחלק ניכר כבר כתבתם בסקריפטים קודמים.
- בנו סקריפט כללי שיהווה תבנית לכל סקריפט חדש שתיצרו. הסקריפט יכיל את כל הפונקציות שעד עכשיו כתבתם ובנוסף דוגמאות קוד (כמו למשל: דוגמאות של שימוש בלולאות, של משפטי תנאי, של מניפולציות על משתנים, שימוש במחרוזות, שימוש במערכים וכדומה) וכך יהיה ניתן להתחיל לכתוב סקריפט חדש בקלות וגם יהיה לכם API (רשימה מוכנה של פעולות ממומשות שניתן להשתמש בהן) מוכן מראש במקום להתחיל לחפש מאיפה להתחיל את הסקריפט או איפה יצרנו סקריפט דומה כדי לעבוד אתו בתור תבנית.

2. PARAMETERS

- השתמשו תמיד רק באותיות גדולות, קו תחתון, ומספרים בשמות פרמטרים על מנת להימנע מהתנגשויות בין מילים שמורות ו/או פקודות.

<u>למשל:</u>

```
THIS_IS_MY_NUMBER_6_VAR="6"
```

- בעת השמה של ערכים למשתנים, אל תשתמשו ברווחים.

<u>למשל:</u>

```
MY_VAR="aValue"
```

<u>ולא:</u>

```
MY_VAR  =  "aValue"
```

- הפעילו פקודות וכלים שהסקריפט עושה בהם שימוש דרך משתנה ולא ישירות על מנת שיהיה ניתן לוודא שיש גישה אליהם והסקריפט לא יישבר אם הם חסרים. כמובן שאין צורך בכך עבור פקודות שמובנות בתוך Bash כגון הפקודה cd.

למשל:

```
AWK_BIN=`which awk`

$AWK_BIN '{print \$1}'
```

הערות:

1. שימו לב שבדוגמא יש את התו גרש אחורי (הכפתור שמשמאל למקש 1 במקלדת) ולא גרש רגיל שפירושו 'הרצת פקודה במקום' (ראו הסבר מפורט בהמשך בחלק של טיפים ב- Bash).
2. הפקודה which מחזירה נתיב מלא אל פקודה קיימת וניתן לסמוך על זה שהיא עצמה קיימת במערכת כי היא נחשבת לבסיסית.
3. סימן הדולר הוא עם סלאש הפוך כדי שמי שיעבד אותו יהיה awk ולא Bash.

- בעת בדיקה של פרמטר בתוך משפט IF שמכיל מחרוזת, יש לתחום את שם המשתנה בין גרשיים כפולים על מנת לכפות על Bash להשוות מחרוזת למחרוזת ללא שגיאות הרצה.

למשל:

```
if [ "$MY_VAR" == "myValue" ]; then
        echo doing stuff …
fi
```

בדוגמא ישנה הדפסה בתוך התנאי ולא הערה היות שב- Bash השארת תנאי ריק ללא פקודות כלשהן בתוכו גורמת לשגיאת הרצה.

3. VALIDATION

- בכל סקריפט שמשתמש בסקריפט אחר, הוסיפו קוד (משפט IF) שמוודא שהסקריפט אכן יכול להגיע אל הסקריפט שהוא משתמש בו.
- וודאו תקינות של כל נתון שמגיע מהמשתמש.
- וודאו שפקודות שהסקריפט משתמש בהן אכן קיימות באותה מערכת על ידי שימוש בפקודה which למשל.
- נסו לתקן אוטומטית בעיות שמונעות מהסקריפט לרוץ באופן חלק היכן שניתן. למשל: אם סקריפט ניסה לפנות לשם מחשב שאיננו קיים, הוסיפו אותו דינאמית ל- /etc/hosts כחלק מהקוד של הסקריפט. כנ"ל אם למשל חסר ניתוב אל שרת כלשהו או חסרה חבילה כלשהי וכדומה.

4. DOCUMENTATION

- בראש כל סקריפט הגדירו מקטע הערות כללי שיכיל תיאור כללי של מה שהסקריפט מבצע, איך משתמשים בו, מי יצר אותו, מתי ומספר הגרסה שלו.
- אל תגדירו משתנים חסרי משמעות כגון A, B ו- C ואל תתקמצנו על אורך השם. משתנה עם שם הגיוני יועיל מאוד להבנת הלוגיקה של הסקריפט וזה ממש לא משנה מה האורך שלו.
- כנ"ל לגבי שמות אחרים כגון שמות של פונקציות.
- הוסיפו הסברים בהערות לצד שורות קוד שעלולות להיות מבלבלות או מסובכות יחסית.
- דוגמאות למוסכמות פיתוח:
  - יצירת סקריפטים תתבצע תמיד באופן הבא:
    - יהיו 2 סוגי סקריפטים: סקריפטים בסיסיים שכל אחד מהם מממש תהליך בסיסי וסקריפטים עוטפים שתפקידם להריץ אוסף של סקריפטים בסיסיים ולוודא את תוצאתם.
    - הסקריפט יחזיר קוד שגיאה לאחר שהוא מסתיים.
    - שם הסקריפט יכיל את עיקר פעולתו ובין מילים יהיו מקפים.
  - קריאה לסקריפט מתוך סקריפט אחר תתבצע תמיד באותו אופן:
    - הקריאה תתבצע דרך משתנה
    - המשתנה יקרא בשם קבוע כמו למשל:
      PATH_TO_SAVE_TO_DB_SCRIPT

      (במקרה זה יש תחילית קבועה של הביטוי PATH_TO , סיומת קבועה של המילה SCRIPT ובאמצע מופיע שמו של הסקריפט).
    - ערכו יכיל תמיד את הנתיב המלא אל הסקריפט (ולא רק את שמו).
    - לפני הקריאה תתבצע בדיקה שהסקריפט קיים במערכת הקבצים.
    - לאחר הקריאה תתבצע בדיקה שהסקריפט רץ בהצלחה (לשם כך יש לוודא שהסקריפט הפנימי מסתיים עם קוד שגיאה).
  - תיקייה זמנית: כל הסקריפטים בסביבה יעבדו עם אותה תיקייה זמנית - אותו מיקום ואותו שם.
  - הרצת פקודות: כלי מערכת שאינם מובנים בכל סוגי ההתקנות שלה (שאינם בסיסיים) ועשויים לא להיות קיימים יקראו באמצעות משתנה שיקרא להם ולא באופן ישיר ולפני קריאתם תתבצע בדיקה שהם אכן קיימים במערכת,

פקודות יקראו באותו אופן (echo עם התוכן בין גרשיים כפולים, פקודות בין גרשיים הפוכים (``) ולא בתוך סוגריים עם $ ($(CMD)), ביצוע ESCAPE קבוע של ערכים ותווים ש- Bash משתמש בהם על מנת שהוא לא יפרש אותם כשאנו רוצים שפקודה אחרת תפרש אותם (למשל: אם אנו משתמשים בפקודה awk בתוך סקריפט, נבצע ESCAPE לשמות השדות הרצויים עם סלאש הפוך היות שגם Bash משתמש בסימן $).

- משתנים: משתנה מסוג מערך תמיד יכיל את הסיומת ARR_. משתנה מסוג מחרוזת תמיד יכיל את הסיומת STR_, משתנים עם תפקיד שחוזר על עצמו יכילו תמיד את אותו השם (כגון TEMP_DIR לתיקייה זמנית, GREP_BIN למיקום של הכלי GREP במערכת וכדומה), משתנים יאותחלו תמיד באותו האופן (מחרוזות בין גרשיים בודדים, פלט של פקודה בין גרשיים הפוכים (``) וכדומה).
- פעולות אחרות: פעולות אריתמטיות (פעולות על מספרים) יתבצעו תמיד באותו אופן על ידי סוג אחד של פקודה (למשל באמצעות הפקודה expr ולא על ידי פקודה אחרת כגון bc או let או דרך קריאה נקודתית לשפת תכנות כלשהי בתוך הסקריפט כגון PERL או PYTHON), פעולות כגון חיתוך קלט יתבצעו תמיד עם אותו סט פקודות (למשל באמצעות הפקודה awk ולא cut).

5. FLEXIBILITY

- גמישות ניהולית - אם למשל יש לכם תיקייה קבועה בסביבה שאתם מנהלים שבה אתם שומרים גיבויים, אל תניחו שהיא תישאר קבועה לתמיד. הפכו אותה לגמישה על ידי כתיבת סקריפט שמחזיר את מיקומה ונהלו את הסביבה שלכם עם הסקריפט ולא ישירות מול התיקייה הזו. באופן זה אם בעתיד תוסיפו לסביבה שלכם בעתיד רכיבים קצת שונים, תוכלו פשוט לעדכן את הסקריפט בהתאם ולהוסיף לו את המיקומים החדשים וכך שאר תשתית הניהול שלכם תמשיך לעבוד כרגיל גם מול הרכיב החדש.
- צרו אוסף של סקריפטים ניהוליים גנריים שעובדים בכל הסביבה שלכם. בכל עדכון הפיצו אותם לכל הסביבה מחדש וכך תשתית הניהול תישאר עקבית וגנרית.

מבנה כללי של סקריפט

1. השורה שמגדירה למערכת איך להריץ את הסקריפט (ידועה גם בשם sha-bang) על ידי צירוף של התו סולמית (#, "sha" באנגלית) עם התו סימן קריאה (!, "bang" באנגלית). זה גם מושג שעוזר לזכור מה לכתוב קודם.

   השורה במקרה של סקריפט ב- Bash הינה:

#!/bin/bash

2. הגדרות של ערכים קבועים שהמשתמש מגדיר לפי צרכיו (**אופציונאלי**, בד"כ יותר בשימוש על ידי סקריפטים שרצים אוטומטית על ידי המערכת)

למשל:

NUMBER_OF_BACKUPS_TO_SAVE=100

3. הגדרות של ערכים קבועים שבשימוש פנימי על ידי הסקריפט.

למשל:

PATH_TO_PERMISSIONS_UPDATE_SCRIPT=/root/scripts/perms-update.sh

GIT_BIN_FILE=`which git`

4. בדיקת תקינות המשתנים שהוגדרו בחלקים הקודמים.

למשל:

```
if ! [ "$NUMBER_OF_BACKUPS_TO_SAVE " -eq
"$NUMBER_OF_BACKUPS_TO_SAVE " ] 2>/dev/null
 then
    echo ERROR: The 2nd argument is NOT a valid number!
    exit 1
fi
```

ישנו טריק ב-Bash שבו אם משווים פרמטר לעצמו באמצעות האופרטור "-eq", בפועל Bash ברקע ממיר למספר אלגברי את שני המשתנים (שדה מסוג מספר ממש ולא מספר שנמצא בשדה מסוג מחרוזת) וכך יוצא שתנאי כזה בפועל בודק האם משתנה מסוים הוא באמת מספר או לא. אם יתבצע ניסיון להכניס טקסט כלשהו למשתנה הנבדק כאן בתנאי, התנאי לא יעבור בהצלחה והסקריפט יסתיים על ידי הפקודה exit.

יחד עם זאת, כפי שניתן לראות, על מנת להימנע מפלט לא אסתטי ולא רצוי שיוצג כשגיאת ריצה על ידי Bash כאשר יכנס למשתנה בדוגמא ערך שאיננו מספר יש לדאוג לשנות קצת את התנאי כך ששגיאה תיזרק ל- NULL ואת המילה then של התנאי יש לרשום בשורה חדשה וללא התו נקודה-פסיק לפניה (התו נדרש כאשר המילה מופיעה באותה שורה עם התנאי).

לדוגמאות נוספות ראו להלן בסעיף 5 - "גוף הסקריפט".

5. עיקר הסקריפט / גוף הסקריפט

החלק העיקרי בסקריפט יכול להיות מורכב מאחד או יותר מהרכיבים הבאים:

* תנאים

- משפטי IF (בקרה ווידוא קלט)

- משפטי CASE (תפריטים, מתגים, אינטראקטיביות)

* לולאות

* מערכים

* מחרוזות

* פונקציות

חלק זה מורכב בעיקר מדוגמאות מעשיות לכל סוג.

<u>תנאים:</u>

<u>משפטי IF</u>

מבנה כללי

```
if [ תנאי ]; then
        Do something
else
        Do something else
fi
```

תנאי הפוך - בשלילה (באמצעות התו !):

```
if ! [ תנאי ]; then
        Do something
else
        Do something else
fi
```

תזכורת חשובה!

חובה להשאיר רווחים בין הסוגריים שתוחמים את התנאי לבין הביטוי שבתוכו אחרת תהיה שגיאת הרצה וה IF לא יעבוד (בפועל הביטוי "]" הוא פקודה בפני עצמה הקיימת במערכת כקובץ בינארי בדומה לפקודות רגילות אחרות (*) ולכן אסור שהוא יהיה צמוד לביטויים אחרים כלשהם. הביטוי הסוגר של התנאי, "]", הוא פרמטר שהפקודה [מקבלת ולכן גם הוא צריך לעמוד בפני עצמו).

(*) אתם מוזמנים לחפש אותו במחשב (ואל תשכחו שהביטוי [בד"כ משמש כביטוי רגולרי אז חפשו אותו עם תו ה- ESCAPE סלאש הפוך כך: "]\" על מנת להתייחס אליו כטקסט פשוט בפקודת החיפוש שלכם).

לא לשכוח את המילה fi בסוף משפט ה- if הסוגרת אותו (fi זה if הפוך).

<u>דוגמאות:</u>

1. תנאי שבודק האם קובץ קיים ואיננו ריק

```
if [ -s "$SITE_DIR"/"$DRUPAL_SETTINGS_FILE" ]; then
        echo "a file was found valid!"
fi
```

הערה חשובה:

התנאי s- בודק קבצים באופן כללי. גם תיקייה נכללת בקטגוריה זו. לכן צריך לשים לב לכך ואם תיקייה היא קלט לא רצוי, יש לשלב תנאי נוסף שמוודא שמדובר בקובץ שאיננו תיקייה. הדוגמא הבאה מדגימה בדיקה של תיקייה.

2. תנאי שבודק האם תיקייה קיימת

```
if [  -d  "$SAVE_IN"  ]; then
        echo  "this dir exists."
else
        echo  "this dir does not exist."
fi
```

3. תנאי שבודק האם הפקודה האחרונה הסתיימה בהצלחה

```
if [  "$?" == 0  ]; then
        echo  "last command ended successfully"
fi
```

4. תנאי שבודק האם נכנסו 2 פרמטרים לסקריפט על ידי המשתמש (בדיקה האם המשתנה הראשון או השני שצריכים להגיע מבחוץ על ידי המשתמש ריקים או לא)

```
if  !  [  "$1" == ""  -o  "$2" == ""  ]; then
        echo "we got 2 parameters from the user"
fi
```

o- : פירושו "או"

! : NOT, הופך את התנאי

5. תנאי שבודק האם משתנה כלשהו מכיל ערך כלשהו

```
if [  "$HTTP_ONLY" == "found"  ]; then
        echo "This variable has this value"
fi
```

6. תנאי שבודק האם לינק סימבולי קיים

```
if  !  [  -h /opt/v  ]; then
        echo "There is a link named  '/opt/v'  "
fi
```

7. תנאי שבודק האם משתנה שהגיע מהמשתמש הוא מספר או לא

```
if [  "$1"  -eq  "$1"  ]  2>/dev/null
then
        echo  "The number gotten from the user is a number"
fi
```

כפי שכבר הוזכר לעיל, ישנו טריק ב- Bash שבו אם משווים פרמטר לעצמו באמצעות האופרטור "-eq", בפועל Bash ברקע ממיר למספר אלגברי את שני המשתנים (שדה מסוג מספר ממש ולא מספר שנמצא בשדה מסוג מחרוזת) וכך יוצא שתנאי כזה בפועל בודק האם משתנה מסוים הוא באמת מספר או לא.

-eq : אופרטור השוואה אלגברי

<u>משפטי CASE</u>

- משפט CASE פותח במילה "case" ומסתיים במילה "esac" שהינה המילה "case" הפוך (כמו ב- IF).
- '*' בתוך התנאי פירושה "כל השאר" (DEFAULT).
- פעמיים נקודה פסיק זה ה- BREAK בין מקרה למקרה.

מבנה כללי:

```
case $PARAMETER in

        option1)
                do something
        ;;
        option2)
                do something
        ;;
        optionN)
                do something
        ;;
        *)
                do something for other options  (default case)
        ;;
esac
```

<u>דוגמאות:</u>

1. משפט CASE שמציג תפריט פשוט למשתמש שמאפשר לו להריץ בקלות פקודות סטנדרטיות.

זו דוגמא לסקריפט אינטראקטיבי שמאפשר למשתמש לבחור פעולה ביתר קלות.

```
echo Please select a command:

echo 1\) df  -h
echo 2\) du  -hs  \*
echo 3\) ls  -l
```

```
echo  -e  "\nYour choice:"

read CHOICE

case $CHOICE in

        1) df  -h
        ;;
        2) du  -hs  *
        ;;
        3) ls  -l
        ;;
        q) echo Bye Bye!
          exit 0
        ;;
        *) echo "Invalid option. Please try again."
        ;;
esac
```

2. מתגים - ניתוח קלט מהמשתמש

זו דוגמא קצת יותר מורכבת שמציגה את היכולת להשתמש במשפט CASE לטובת קבלת רשימת ארגומנטים מורכבת יחסית מהמשתמש אבל די בקלות. ☺

בדוגמא משפט ה- CASE נועד לבודד את הנתונים מהמתגים ולהבין מהמשתמש איך להשתמש בפקודת סריקת הרשת NMAP.

המשתמש יכול להשתמש באופציה h- שזהה ל- host-- כדי לציין שם מחשב ספציפי לסריקה או לבחור טווח של פורטים לסריקה בפרוטוקול TCP או UDP והמתגים בהתאמה.

הקוד הנוסף מסביב נועד להשלים את התמונה. הוא מכניס את הערכים מהמשתמש למערך, בודק את תקינותם ולאחר מכן מייצר ומריץ פקודת NMAP מתאימה לפי בקשת המשתמש.

```
if [ "$#" -gt 0 ]; then
        INPUT=$@

        TOKENS=(${INPUT//\ / })
        LEN=${#TOKENS[@]}
        COUNTER=0
```

```
while [ $LEN -gt 0 ]
do
        if [ "$DEBUG" == "yes" ]; then
                echo ${TOKENS[$COUNTER]}
        fi

        case "${TOKENS[$COUNTER]}" in
          -h|--host)
                HOST=${TOKENS[$COUNTER+1]}

                echo $HOST | $GREP_BIN -E "^(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-
                9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-
                9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$" > /dev/null

                # if the input is not a valid v4 ip address
                if ! [ "$?" == "0" ]; then
                        echo "ERROR! Wrong ip value!"
                        echo "Aborting ..."
                        exit $WRONG_PARAMETERS_ERR
                fi

                LEN=0
          ;;
          -t|--tcp)
                RANGE=${TOKENS[$COUNTER+1]}

                RANGE_ARR=(${RANGE//-/ })

                MIN_PORT_RANGE=${RANGE_ARR[0]}
                MAX_PORT_RANGE=${RANGE_ARR[1]}

                if [ "$DEBUG" == "yes" ]; then
                        echo "TCP INPUT:"
                        echo "RANGE: "$RANGE
                        echo "MIN_PORT_RANGE: "$MIN_PORT_RANGE
                        echo "MAX_PORT_RANGE: "$MAX_PORT_RANGE
                fi
```

```
        # checking if a num equals itself in bash actually checks if a value
        # is an algebric number or not

        if ! [ "$MIN_PORT_RANGE" == "" -a "$MAX_PORT_RANGE" == "" ];
        then
                if [ "$MAX_PORT_RANGE" -eq "$MAX_PORT_RANGE" -a
                "$MIN_PORT_RANGE" -eq "$MIN_PORT_RANGE" ]; then
                        if [ "$MIN_PORT_RANGE" -le "$MAX_PORT_RANGE" -a
                        "$MIN_PORT_RANGE" -gt 0 -a "$MAX_PORT_RANGE" -
                        gt 0 ]; then
                                if [ "$UDP" == "" ]; then
                                        TCP=" -p T:$MIN_PORT_RANGE-
                                        $MAX_PORT_RANGE"
                                else
                                        echo "The 'udp' option was alreay chosen.
                                        Ignoring 'tcp' request."
                                fi

                                LEN=0
                        else
                                echo "ERROR! Wrong port values!"
                                echo "Aborting ..."
                                exit $WRONG_PORTS_VALUES_ERR
                        fi
                else
                        echo "ERROR! Wrong port values!"
                        echo "Aborting ..."
                        exit $WRONG_PORTS_VALUES_ERR
                fi
        else
                echo "ERROR! Wrong port values!"
                echo "Aborting ..."
                exit $WRONG_PORTS_VALUES_ERR
        fi
;;
-u|--udp)
        RANGE=${TOKENS[$COUNTER+1]}
```

```
RANGE_ARR=(${RANGE//-/ })

MIN_PORT_RANGE=${RANGE_ARR[0]}
MAX_PORT_RANGE=${RANGE_ARR[1]}

if [ "$DEBUG" == "yes" ]; then
        echo "UDP INPUT:"
        echo "RANGE: "$RANGE
        echo "MIN_PORT_RANGE: "$MIN_PORT_RANGE
        echo "MAX_PORT_RANGE: "$MAX_PORT_RANGE
fi

# checking if a num equals itself in bash actually checks if a value
# is an algebric number or not

if ! [ "$MIN_PORT_RANGE" == "" -a "$MAX_PORT_RANGE" == "" ];
then
        if [ "$MAX_PORT_RANGE" -eq "$MAX_PORT_RANGE" -a
        "$MIN_PORT_RANGE" -eq "$MIN_PORT_RANGE" ]; then
                if [ "$MIN_PORT_RANGE" -le "$MAX_PORT_RANGE" -a
                "$MIN_PORT_RANGE" -gt 0 -a "$MAX_PORT_RANGE" -
                gt 0 ]; then
                        if [ "$TCP" == "" ]; then
                        UDP=" -sU -p U:$MIN_PORT_RANGE-
                        $MAX_PORT_RANGE"
                else
                        echo "The 'tcp' option was alreay chosen. Ignoring
                        'udp' request."
                fi

                LEN=0
        else
                echo "ERROR! Wrong port values!"
                echo "Aborting ..."
                exit $WRONG_PORTS_VALUES_ERR
        fi
else
```

```
                        echo "ERROR! Wrong port values!"
                        echo "Aborting ..."
                        exit $WRONG_PORTS_VALUES_ERR
                fi
        else
                echo "ERROR! Wrong port values!"
                echo "Aborting ..."
                exit $WRONG_PORTS_VALUES_ERR
        fi
    ;;
    *) #default
        echo "ERROR! Wrong parameter insterted."
        echo "Usage: "$0$USAGE
        exit $WRONG_PARAMETERS
    ;;
    esac

        COUNTER=$(expr $COUNTER + 1)
        LEN=$(expr $LEN - 1)
    done
fi

if ! [ "$HOST" == "" ]; then
        nmap $HOST
else
    if ! [ "$TCP" == "" ]; then
        nmap $TCP
    else
        if ! [ "$UDP" == "" ]; then
            nmap $UDP
        fi
    fi
fi
```

לולאות:

קיימים מספר סוגים של לולאות והן עוזרות מאוד (בייחוד בשילוב עם מערכים - ראו להלן) ליצור דינאמיות לקוד.

דוגמאות:

1. ריצה על רשימת הקבצים שנמצאים בתיקייה הנוכחית

```
for i in $(ls); do
        echo $i
done
```

2. ריצה על כל התיקיות שמכילות קבצים בעלי סיומת 'cnf' החל מהתיקייה /etc והעתקתן לתיקיית גיבוי אם היא לא קיימת שם כבר

```
for i in $(find /etc -name "*.cnf" -exec dirname {} \;); do
        if ! [ -d /backup/"$i" ]; then
                echo "Copying '$i' to '/backup' ..."
                cp -R $i /backup
        else
                echo "Folder '/backup/$i' was already copied."
        fi
done
```

3. ריצה על תוכן של קובץ שורה אחרי שורה וביצוע פעולה על כל שורה

```
cat /tmp/myFile | while read line; do
   echo $line
done
```

4. ריצה על רשימת קבצים שנמצאת בתוך קובץ טקסט

```
for i in $(cat  "$LIST_OF_FILES_for_REMOVAL"); do
        rm $i
done
```

5. ריצה על רשימת ערכים שמוגדרת בתוך משתנה ומופרדת באמצעות רווחים

```
SITES_DIRS="/sitesDir1  /sitesDir2 /var/www/html"

for i in $(echo  $SITES_DIRS); do
        echo $i
done
```

פלט:

```
/sitesDir1
/sitesDir2
/var/www/html
```

6. ריצה עם מספר רץ - דוגמא 1

```
for i in `seq 1 5`; do
        echo $i;
 done
```

<u>פלט:</u>

```
1
2
3
4
5
```

<u>הערה:</u>

ניתן לספק לפקודה seq טווח רצוי גם באמצעות ערכים של משתנים (למשל: seq $x $y) זאת בניגוד לשיטה השנייה בהמשך (הגדרת טווח בין סוגריים מסולסלים) שבה זה לא אפשרי.

7. ריצה עם מספר רץ - דוגמא 2 (שיטה מהירה יותר)

```
for i in {1..10}; do
        echo $i;
done
```

<u>פלט:</u>

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

8. ריצה עם מספר רץ - דוגמא 3 (דילוג של יותר מאחד בפעם - בדוגמא דילוגים של 2 בפעם)

```
for i in {1..10..2}; do
        echo $i;
done
```

פלט:

```
1
3
5
7
9
```

9. ריצה עם מספר רץ - דוגמא 4 (שימוש במספר כדי ליצור חלק מפקודה)

```
for i in {1..254}; do
        ssh 192.168.5.$i 1 >> /root/ssh.txt 2>&1
done
```

10. ריצה עם מספר רץ - דוגמא 5 (סגנון שפת C - לשים לב שיש בפקודה **2 זוגות של סוגריים**)

```
for ((i=1; i<=25; i++)); do
        mkdir dir$i;
done
```

11. ריצה על רשימת ערכים ספציפית (ערכים אקראיים - הערכים לא תחומים כלל בשום סוג של סוגריים)

```
for i in word a b 6# f ggg 6 77; do
        echo $i;
done
```

פלט:

```
word
a
b
6#
f
ggg
6
77
```

12. ריצה על תוכן של קובץ שורה אחרי שורה.

נניח שזה הקובץ:

```
$ cat list.txt
one
two
three
four
```

```
while read line
do
        echo "$line" '!'
done <list.txt
```

<u>פלט:</u>

```
one !
two !
three !
four !
```

13. ריצה על טקסט לפי מחלק (DELIMITER) שנבחר על ידי המשתמש (בניגוד למחלק ברירת המחדל שהינו תו שורה חדשה). בכל מחזור של הלולאה המשתנה יכיל מקטע ערכים לפי המחלק שהוגדר בלולאה

```
echo "this is one line + this is a second line + this is a third line +
this is a forth line + this is a fifth line +" | while read -d "+" i; do
        echo $i;
done
```

בדוגמא זו ניתן לראות שהמחלק החדש הוא התו פלוס (+).

<u>פלט:</u>

```
this is one line
this is a second line
this is a third line
this is a forth line
this is a fifth line
```

<u>מערכים:</u>

מערכים בשילוב עם לולאות מספקים דרך קלה, פשוטה, מהירה, נוחה ועוצמתית לייצור קוד דינאמי שמבצע פעולות מסוגים שונים.

<u>דוגמאות:</u>

<u>אתחול מערך</u>

```
1. array1=()
2. array2=(a b c d e f g h)
3. array3=(`ls`)
4. array4=(one two three)
5. array5=$(find `get-backup-folder.sh` -maxdepth 1 -name "*.gz" -mtime +120)
```

```
6. MY_STRING=192.168.111.1
TOKENS_ARRAY=(${MY_STRING//\./ })
```

כפי שניתן לראות, ניתן לאתחל מערך במספר דרכים כאשר אחת מהן היא הפלט של פקודה כלשהי. אופציה זו מאוד נוחה לשימוש.

אופציה נוחה נוספת הינה האופציה האחרונה ברשימה שממירה מחרוזת למערך לפי מחלק רצוי או במילים אחרות מחלקת מחרוזת לאסימונים (TOKENS) לפי מחלק (DELIMITER) ומכניסה אותם למערך. בדוגמא המחלק הוא התו נקודה אך ניתן להחליפו בתו רצוי אחר כגון רווח ואז הביטוי שמייצר את המערך יהיה: (${MY_STRING//\ / }).

<u>הצגת מספר האיברים במערך</u>

```
${#array[@]}
```

<u>או</u>

```
${#array[*]}
```

<u>גישה לאיבר במערך</u>

```
${array[0]}


${array[3]}
```

האיבר הראשון במערך נגיש באינדקס 0 וכן הלאה ...

<u>הוספת איברים למערך</u>

```
array+=('item1')
array+=('item2')
array+=(item3)
array+=(`command`)
```

ניתן להוסיף איברים למערך באמצעות האופרטור =+ כולל שרשור של תוצאה של פקודה.

<u>מחיקת איברים ממערך</u>

נניח שיש לנו את המערך הבא:

```
array=(a1 b2 c3 d4)
```

שורה זו מוחקת מהמערך את האיבר b2:

```
array=(${array[@]/b2})
```

<u>הדפסת מערך בפקודה אחת</u>

```
echo ${array[@]}
```

<u>או</u>

```
echo ${array[*]}
```

שמירת כל האינדקסים של מערך כערכים במערך חדש

```
indexes=(${!array[@]})
```

הדפסת חלק ממערך (SUBSET)

הדפסת האיבר האחרון במערך

```
echo ${array[@]: -1}
```

הדפסת חמשת האיברים האחרונים במערך

```
echo ${array[@]: -5}
```

הדפסת כל האיברים במערך למעט 2 האיברים הראשונים

```
echo ${array[@]: 2}
```

שכפול מערך

```
copy_array=${array[@]}
```

בדיקה האם איבר מסוים קיים במערך או לא

נניח שיש לנו את המערך הבא:

```
array=(a b c d e f g h)
```

שלב א' - נריץ GREP עם שם האיבר שנרצה למצוא

לאחר הדפסת המערך כולו:

```
echo "${array [@]}" | grep d   >   /dev/null
```

דרך נוספת (על ידי האופרטור <<<):

```
grep d <<< "${array [@]}"    >   /dev/null
```

שלב ב' - נשתמש בקוד החזרה של Bash (שנשמר במשתנה '$?') על מנת לקבוע האם האיבר נמצא במערך או לא:

```
if [ "$?" == "0" ]; then
      echo item exists!
else
      echo item does not exist.
fi
```

ריצה על מערך

דרך א

```
SERVER_IPS=(`ip addr | grep inet | \
awk '{print \$2}' | awk -F/ '{print \$1}'`)
```

```
for i in "${SERVER_IPS[@]}"; do
    echo $i
done
```

הפקודה הראשונה יוצרת מערך המכיל רשימה של כתובות ה- IP של המערכת.

הפקודה השנייה היא לולאת for המדפיסה את המערך איבר אחד בפעם.

כמובן שבפועל נבצע על כל איבר פעולות קצת יותר מורכבות שהרי להדפסה יש לנו פקודה פשוטה נפרדת (ראו לעיל "הדפסת מערך בפקודה אחת").

<u>דרך ב</u>

```
MY_ARR=(`ls`)

COUNTER=${#MY_ARR[@]}

i=0

while ! [ "$i" == "$COUNTER" ]
do
    ELEMENT="${MY_ARR[$i]}"

    echo $ELEMENT

    i=$(expr $i + 1)
done
```

כאן הלולאה שרצה על המערך (שמאותחל להיות רשימת הקבצים והתיקיות בתיקייה הנוכחית) הינה לולאת WHILE ולכן היא מצריכה זוג משתנים נוסף: האחד שישמש כמונה של האיברים במערך והשני שיכיל את מספר האיברים במערך על מנת שהלולאה תעצור לאחר המעבר על כולם.

המונה עולה באמצעות שימוש בפקודה expr (לדוגמאות נוספות אודות פעולות מתמטיות ב- Bash ראו להלן בסעיף על טיפים ב- Bash).

<u>מחרוזות:</u>

להלן דוגמאות לעבודה עם מחרוזות.

כל הדוגמאות מתייחסות למחרוזת הבאה:

```
STR='this is my string'
```

הדוגמאות מתייחסות להדפסה למסך אך כמובן שניתן לבצע על תתי-המחרוזות פעולות נוספות כגון שמירתן למשתנה, הרצת פקודה עליהן וכדומה.

דוגמאות:

הדפס את מספר התווים במחרוזת

```
echo ${#STR}
```

פלט:

```
17
```

הדפס רק את 2 התווים האחרונים

```
echo "${STR:${#STR}-2}"
```

פלט:

```
ng
```

הדפס את כל התווים למעט 2 התווים האחרונים

```
echo "${STR:0:${#STR}-2}"
```

פלט:

```
this is my stri
```

הדפס רק את 2 התווים הראשונים

```
echo "${STR:0:2}"
```

פלט:

```
th
```

הדפס את כל התווים למעט 2 התווים הראשונים

```
echo "${STR:2}"
```

פלט:

```
is is my string
```

ספירת מספר המילים במחרוזת

```
ARR=($STR)

echo "${#ARR[@]}"
```

דוגמא זו מדפיסה את מספר המילים במחרוזת לעיל על ידי שימוש במערך זמני.

החלפת תווים במחרוזת

ניתן להחליף תווים בתוך מחרוזת בקלות ובאופן מיידי.

מבנה כללי:

```
${var_name_to_work_on/str_to_find/str_to_replace_with}
```

למשל, אם נפעיל את ההחלפה על המחרוזת לעיל עם הביטוי הבא:

```
STR=${STR/my/his}
```

נקבל:

```
echo $STR
this is his string
```

מחיקת התו האחרון אם הוא סלאש הפוך (באמצעות האופרטור %)

```
PATH='/boot/grub/'
PATH=${PATH%/}
```

דוגמא זו תמחק את תו הסלאש האחרון מהמחרוזת "/boot/grub/" ותשמור מחדש את המחרוזת ללא הסלאש במשתנה PATH.

פונקציות:

כמו בכל שפה, פונקציות הינן יכולת חשובה ונוחה לעשות סדר בקוד. זה לא מזיק ואף רצוי לנצל אותה בכל הזדמנות על מנת שלא יהיה לנו קוד כפול.

יחד עם זאת, כפי שצוין כבר לעיל, ניתן להסתכל גם על סקריפט כפונקציה ולחלק את הפעולות של תהליך שנרצה להגדיר למספר סקריפטים קטנים שכל אחד מבצע פעולה בסיסית ומחזיר ערך ל- Bash. את הסקריפטים הקטנים נפעיל יחדיו מתוך סקריפט "עוטף" וכך ניצור תהליך שלם באופן מסודר. באופן זה כל סקריפט בסיסי יתפקד כמו פונקציה וכך גם לא נחזור על קוד וגם נוכל להפעיל תהליך ביתר קלות על ידי הפעלה פשוטה של הסקריפט הבסיסי. יתרון נוסף של שיטה זו הוא היכולת להפעיל סקריפטים בסיסיים בפני עצמם ישירות מתוך Bash כשנרצה בכך.

במילים אחרות, ניתן לבחור בין 2 האסטרטגיות אך הכי נכון לשלב ביניהן - כשסקריפט בסיסי חוזר על קוד, להשתמש בפונקציות בתוכו אך את התהליך הבסיסי שהוא מממש להשאיר כסקריפט ממש ולא ליצור פונקציה גם עבור התהליך כולו גם אם הוא בסיסי.

דוגמאות:

```
echoToStdErr()
{
	# input: text
	# output: nothing
	# action: sends text to stderr

	echo "$@" 1>&2; # stderr
}
```

פונקציה שמדפיסה ל- STDERR את הקלט שנכנס אליה.

קריאה לפונקציה:

```
echoToStdErr  "ERROR! Invalid path or file name given as results file."
```

הביטוי @$ אומר "כל הקלט שנכנס לפונקציה", בדיוק כמו הפירוש שלו ברמת הסקריפט כולו. לכן בהתאמה גם ארגומנטים בודדים שיכנסו לפונקציה יקבלו שמות בדיוק כמו הארגומנטים שנכנסים לסקריפט מבחוץ. אם המשפט האחרון לא כל כך מובן אז אולי הדוגמא הבאה תוכל לפשט אותו קצת יותר:

```
myHelloWorldFunction()
{
   echo $1
   echo $2
}


myHelloWorldFunction  "Hello"  "World"
```

פלט:

```
Hello
World
```

אז כפי שניתן לראות מדוגמא זו, כמו שארגומנטים שנכנסים לסקריפט מבחוץ דרך ה-SHELL מקבלים את השמות $1, $2 וכן הלאה בהתאמה, כך נקרא גם שמם בתוך פונקציות אך מדובר במשתנים נפרדים.

6. הקריאה לסקריפט: ניתן לקרוא לסקריפט עם ארגומנטים נוספים או בלי. לרוב יותר נוח להגדיר ארגומנטים על מנת להפוך את הסקריפט ליותר גמיש ומשוכלל. יחד עם זאת ישנם מקרים שזה דווקא פחות נוח לעבוד עם ארגומנטים כגון בהפעלת סקריפטים אוטומטית על ידי המערכת עצמה לצורך תחזוקה שוטפת למשל.

קריאה עם ארגומנטים לדוגמא:

generate-a-random-number.sh 8 5

קריאה לסקריפט עם 2 ארגומנטים: במקרה הזה הוא יחולל 5 מספרים אקראיים באורך של 8 תווים.

הארגומנטים בתוך הסקריפט מיוצגים על ידי הביטויים $1, $2, $3 עבור הנתון הראשון, השני והשלישי בהתאמה והלאה כאשר $0 הוא ביטוי המייצג את שם הקובץ שכתבנו בו את הקוד (שם הסקריפט בעצם). בדוגמא של הקריאה לעיל המשתנים $1 ו- $2 קיבלו ערכים מהמשתמש.

טיפים הקשורים ל-Bash ולכתיבת סקריפטים

1. הדפסת פלט למסך:
   - 1.1 הדפסת פלט מסקריפטים וההתקן /dev/null
   - 1.2 הדפסת שורה חדשה (\n)
   - 1.3 הדפסת התו !
   - 1.4 הדפסת התו ′
   - 1.5 הדפסת תוכן של משתנה
   - 1.6 הדפסת הודעות שגיאה
   - 1.7 שימוש בתו * בסקריפט כמו שהוא
2. פקודה בתוך פקודה (גרש הפוך לעומת $())
3. מתמטיקה ב- Bash
4. צבעים ב- Bash
5. קבלת קלט מהמשתמש
6. משתנים מיוחדים
7. הרצת מספר פקודות במקביל בשורה אחת
8. שימוש בתאריכים ושעות
9. מעבר בין תיקיות עם popd ו- pushd
10. ALIASים ב- Bash
11. --color=always או --color=auto
12. התחמקות מבעיות:
    - 12.1 התו ″ לעומת ′
    - 12.2 תיחום משתנים עם {}
    - 12.3 שימוש בפקודה which
    - 12.4 הרצת פקודה מורכבת בתוך סקריפט
13. יכולות נוספות שיש ל- Bash:
    - 13.1 התיקייה הקודמת (התו -)
    - 13.2 שרשור פקודות עם התו PIPE (|)
    - 13.3 האופרטור *
    - 13.4 האופרטור ?
    - 13.5 הפעלת פקודה על רשימה והאופרטור {}
    - 13.6 הפקודה eval
    - 13.7 האופרטור <<<
    - 13.8 יצירת קובץ תוך כדי הרצת פקודה
    - 13.9 יצירת קובץ עם שורות מרובות
    - 13.10 התו סלאש הפוך (\) בפני עצמו

1. הדפסת פלט למסך

יש 2 פקודות להדפסת פלט למסך. האחת היא echo והשנייה היא printf.

מטעמי נוחות וגם היות שהיא יותר מוכרת, נשתמש בעיקר ב- echo.

1.1 הדפסת פלט מסקריפטים וההתקן /dev/null

באופן כללי, מומלץ מאוד להשקיע כמה שיותר בהדפסת פלט למסך לכל אורך הסקריפט כי באופן זה יהיה יותר קל להבין מה לא עובד או באיזה שלב תהליך ההרצה נתקע (בנוסף עיינו בסעיף 15 בהמשך שמסביר איך לדבג סקריפטים).

דרך נוספת היא, להגדיר פרמטר נוסף לסקריפט שמתקבל על ידי המשתמש. כמו

debug-- או verbose-- למשל ואם המשתמש יקרא לסקריפט אתו, יודפס פלט יותר מפורט לאורך הרצת הסקריפט.

פקודות רבות בלינוקס עובדות באופן דומה. לעתים יש אפילו רמות של פירוט ואז מתג כגון – vvv שקיים בפקודות כגון SSH פירושו "very very verbose" ...

יחד עם זאת, לפעמים נרצה שלא להדפיס פלט של סקריפט בסיסי כאשר אנו קוראים לו מסקריפט אב כי אנו רוצים פלט יחסי לסקריפט האב ולא לסקריפט הבן שהאב קורא לו. בנוסף, יש פקודות שנרצה להריץ מבלי להציג או להשתמש בפלט שלהן באופן ישיר (אלא רק בעקיפין על ידי קריאת סטאטוס הפקודה מ- Bash - ראו הסבר להלן). במקרה כזה כל מה שצריך

לעשות זה פשוט לשלוח את הפלט לאחר קריאה לסקריפט הבן או של כל פקודה אחרת שלא נרצה להציג את הפלט שלה להתקן המיוחד /dev/null כך:

```
PATH_TO_MY_SCRIPT ='/path/to/my/script.sh'
$PATH_TO_MY_SCRIPT  >  /dev/null
```

<u>או</u>

```
GIT_COMMAND=`which git` 2>/dev/null
$GIT_COMMAND  pull origin master  >  /dev/null
```

<u>או</u>

```
grep "something"  file  >  /dev/null
if [ "$?" == 0 ]; then echo "found"; fi
```

ההתקן המיוחד /dev/null הוא מעין סל אשפה. מה שמגיע אליו לא מגיע לשום מקום אחר והולך לאיבוד.

בצורה זו, נוכל לשלוט בפלט ואז מתי שנרצה יהיה לנו פלט מפורט כי הגדרנו לעצמינו לעבוד כך כמוסכמה אך מתי שלא נרצה אותו, נוכל לשלוח אותו להתקן הזה והוא לא יוצג בשום מקום וכך נהנה משני העולמות גם יחד.

1.2 הדפסת שורה חדשה (\n)

יש 2 דרכים להדפסת שורה חדשה:

1. הדפסת שורה ריקה סתם לטובת עיצוב פלט יותר ברור

<u>פקודה:</u>

פשוט echo ללא פרמטרים

```
echo
```

<u>או</u>

```
echo  ""
```

<u>או</u>

```
echo  -e  "\n"
```

2. הדפסת שורות ריקות לפני ו/או אחרי טקסט

<u>פקודה:</u>

```
echo  -e  "\nHello World"'!'"\n"
```

<u>פלט:</u>

```
שורה ריקה למעלה
Hello World!
שורה ריקה למטה
```

הערות:

- -e המתג מאפשר ל- echo להתייחס לתווים מיוחדים.
- לסימן קריאה יש משמעות מיוחדת ב- Bash (הרצת פקודה מהמאגר שנשמר בהיסטוריה) ולכן על מנת להדפיסו כמו שהוא חייבים לתחום אותו בין תווים של גרש בודד ('!'). לכן במקרה זה כדי להגיע לתוצאה כפי שהיא מופיעה בפלט נדרש לתחום אותו כנזכר ולאחר מכן להוסיף עוד תו מיוחד של \n בין גרשיים כפולים לטובת הדפסת שורה חדשה נוספת בסוף וזה מה שמרכיב את אוסף הגרשיים לעיל.

1.3 הדפסת התו !

לתו סימן קריאה (!) יש משמעות מיוחדות כאשר משתמשים בו ב- SHELL. הוא מאפשר להריץ פקודות ממאגר היסטוריית הפקודות של Bash.

לכן התנהגותו משתנה כשמשתמשים בו בסקריפט או מחוצה לו.

היות שאין קשר ישיר בין סקריפטים לבין מאגר ההיסטוריה, ניתן להדפיס את התו ! בתוך סקריפט כרגיל עם פקודת echo רגילה:

```
echo "!"
```

אך, על מנת להדפיס את התו הזה מחוץ לסקריפט צריך להשתמש בדרכים עקיפות בגלל תפקידו המיוחד. במקרה זה נוכל להשתמש באחד מהביטויים הבאים:

```
echo '!'        (בין גרשיים בודדים)
echo !          (ללא גרשיים כלל)
echo \!         (עם סלאש הפוך וללא גרשיים כלל)
```

הפקודה לעיל, echo "!" , כאמור לא תעבוד מחוץ לסקריפט ותחזיר את השגיאה:

```
bash: !: event not found
```

חשוב לזכור זאת ולהבחין בין ההתנהגויות השונות כדי לדעת מתי להשתמש בדרכים העקיפות ומתי ניתן להשתמש בדרך הסטנדרטית. אם נרצה למשל להכניס את התו ! כחלק מארגומנט לסקריפט מתוך Bash (כחלק ממחרוזת שמגדירה סיסמא למשל), נצטרך להשתמש בדרך עקיפה (כגון עם סלאש הפוך לפני התו - !\).

1.4 הדפסת התו '

אמרנו שצריך להקפיד על מוסכמות פיתוח קבועות וזה אכן חשוב ונכון ואמרנו שעל מנת להתחמק מבעיות מיותרות, כדאי ונכון להדפיס מחרוזות על ידי שימוש בתו גרש בודד (כך: 'ביטוי' echo).

כללים אלה יוצרים בעיה כאשר נרצה להדפיס את התו גרש בודד עצמו כפי שהוא.

כמובן שהביטוי:

```
echo '''
```

לא יעבוד היות שלגרשיים יש תפקיד מיוחד (תיחום מספר מילים לביטוי אחד) ובמקרה זה חסר הסוגר של סט הגרשיים השני שנפתח.

זה מובן, אך באופן מפתיע גם הביטוי הבא שלכאורה אמור לעבוד אינו עובד:

```
echo  '\"
```

ולכן עלינו להגדיר כמקרה חריג את ההדפסה של התו ולהשתמש בדרך אחרת להדפיסו. הדרכים להדפיסו הם:

1. בין גרשיים כפולים:

```
echo  " ' "
```

כל ביטוי מורכב יותר בשילוב אתו גם יעבוד ללא בעיה כי תחמנו את כל הביטוי בין גרשיים כפולים. כמו למשל:

```
echo  " 123 \" ' abc "
```

2. הדפסתו ללא גרשיים כלל אך עם התו סלאש הפוך:

```
echo  \'
```

במקרה זה ביטוי מורכב יותר שמכיל אותו (ועונה על כללי הפיתוח שהגדרנו) יראה כך:

```
echo  '123'  \'  ' abc '
```

ניתן לראות בדוגמא שהגרש הבודד שנרצה להדפיס כטקסט עומד בפני עצמו מחוץ לשאר הביטוי והוא מחלק את הביטוי כולו לתתי ביטויים התחומים בין גרשיים בודדים.

1.5 הדפסת תוכן של משתנה

אם נרצה להדפיס תוכן של משתנה, הדרך הכי ברורה לבצע זאת היא על ידי הבחנה ברורה בין המלל של ההודעה עצמה לבין ערכו של המשתנה.

היות שחשוב מאוד לפתח בצורה קבועה, גם הדפסה נכללת ברשימה ולכן כדאי שנמצא דרך נוחה וקבועה גם לפעולה זו.

כדי להבחין בין המלל לבין המשתנה נוכל להשתמש במבנה קבוע שבו המלל נמצא בין סוגריים כפולים דווקא שמאפשרים התייחסות לתווים מיוחדים (בניגוד לגרש בודד שמתייחס לתווים כפי שהם) והמשתנה נמצא בין גרשיים בודדים שנמצאים בתוך הגרשיים הכפולים זאת על מנת לדעת בדיוק איפה מתחיל ואיפה נגמר הערך של המשתנה. בנוסף, שימוש בגרשיים בודדים בתוך הכפולים **שוללת** את ההתנהגות הרגילה של גרשיים בודדים וכך נקבל את ערכו של המשתנה ולא את שמו בניגוד למצב רגיל של שימוש בגרשיים בודדים.

הנה דוגמא לשימוש במבנה הזה:

קטע קוד (להרצה **בתוך** סקריפט):

```
DIR_NAME='/opt/myProgram'

echo  "ERROR!  The  dir  '$DIR_NAME'  is an invalid dir!"
```

פלט:

ERROR! The dir '/opt/myProgram' is an invalid dir!

תזכורת!

קוד זה יעבוד רק בתוך סקריפט עקב התו הבעייתי ! שיש לו תפקיד מיוחד כאשר משתמשים ישירות בתוך ה- SHELL (קורא לפקודות מההיסטוריה).

הערות:

- שימו לב שוב לכך שגרשיים בודדים **בתוך** כפולים מתנהגים אחרת בהשוואה לשימוש בהם בפני עצמם.

פקודה להשוואה למשל:

```
echo 'The name of my dir is: $DIR_NAME '
```

פלט:

```
The name of my dir is: $DIR_NAME
```

ניתן לראות שהגרשיים הבודדים כאשר הם בשימוש בפני עצמם ידפיסו במקרה זה את שם המשתנה ולא את ערכו.

- התו סלאש הפוך (\) או בשמו הנוסף תו ה- ESCAPE לא יודפס כפי שהוא מתוך משתנה גם כאשר התו נמצא בין גרשיים בודדים (שזה תפקידם - להתייחס לתווים כפי שהם). על מנת להדפיס סלאש הפוך מתוך משתנה יש לאתחל את המשתנה בביטוי '\\' תוך שימוש בסלאש פעמיים (ESCAPE של תו ה- ESCAPE עצמו...) ולא פעם אחת.

1.6 הדפסת הודעות שגיאה

יש 2 נקודות שחשוב לשים אליהם לב בהקשר של הדפסת הודעות שגיאה:

1. COPY - PASTE

זו נקודה מאוד חשובה כי היא יוצרת מצבים מתסכלים.

קוד נוטה לחזור על עצמו הרבה ולכן הרבה ממנו מועתק. עובדה זו עלולה לגרום לכך שקוד מועתק של הדפסת הודעת שגיאה לא יתעדכן עבור המקרה הנכון מחוסר תשומת לב ואז במקרה של שגיאה אמיתית תודפס ההודעה הלא נכונה שתגרום לבלבול ואי יכולת למצוא את המקור לשגיאה.

למשל:

נניח שהוגדרו המשתנים הבאים:

```
PATH_TO_PERMS_SET _SCRIPT =/root/scripts/update-perms.sh
PATH_TO_REM _TXT_FILES_SCRIPT=/root/scripts/rem-txt-files.sh
```

כעת אנחנו רוצים לוודא ששני הסקריפטים נגישים וקיימים. על מנת לבצע את הבדיקה עבור הסקריפט הראשון נוכל לכתוב למשל קוד כזה:

קוד מקורי:

```
if ! [  -s  "$PATH_TO_PERMS_SET _SCRIPT"  ]; then
  echo  "ERROR!  Internal problem!"
  echo  "Cannot find the permissions set script! Aborting ..."
  echo  "Make sure the permissions set script exists and accessible."

  exit 1
fi
```

על מנת לבדוק את הנגישות של הסקריפט השני נוכל להעתיק את מקטע הקוד הקודם ורק לעדכן בו את המשתנה שמכיל את הנתיב אל הסקריפט השני ואת הודעת השגיאה בהתאמה.

אך אם נשכח לעדכן את ההודעה, נקבל:

קוד מועתק ולא מעודכן:

```
if ! [  -s  "$ PATH_TO_REM _TXT_FILES_SCRIPT"  ]; then
  echo  "ERROR!  Internal problem!"
  echo  "Cannot find the permissions set script! Aborting ..."
  echo  "Make sure the permissions set script exists and accessible."

  exit 1
fi
```

אם תהיה בעיה בגישה לסקריפט השני, יהיה קשה להבין זאת כי נקבל שגיאה עבור הסקריפט הראשון.

לכן צריך לזכור לשים לב לעניין הזה כדי להימנע מתסכולים מיותרים.

2. מבנה קבוע

גם בהודעות שגיאה חשוב להקפיד על מבנה קבוע.

רצוי להגדיר מבנה כללי קבוע לשגיאות ולעשות הבחנה בין סוגי שגיאות גם ברמת חומרת השגיאה ואפילו ברמת המלל על מנת ליצור אחידות מקסימלית.

בנוסף, היכן שניתן, כדאי להדפיס שגיאות אינפורמטיביות שמציגות למשתמש את תכני הארגומנטים שהוא ניסה להכניס לסקריפט במקום שגיאה כללית וכך יהיה יותר קל להבחין בשגיאה.

לדוגמא:

אם משתמש צריך להכניס לסקריפט כארגומנט ראשון נתיב אל תיקייה והוא הכניס נתיב לא קיים או נתיב אל קובץ במקום אל תיקייה, במקום לכתוב את השגיאה הכללית הזו:

```
echo  "ERROR!  The parameter inserted is an invalid path!"
```

נשתמש בשגיאה הספציפית הזו:

```
echo  "ERROR!  The parameter  '$1'   is an invalid path!"
```

דוגמאות נוספות לשגיאות במבנה קבוע:

שגיאות קריטיות:

```
echo "ERROR!  The file  '$1'  is not a valid file."

echo  "ERROR!  Internal problem! Cannot find the backup script!
Aborting …"

echo  "ERROR!  Wrong Input."

echo  "ERROR!  Invalid site dir entered."

echo  "ERROR!  Something went wrong while trying to create an sql
 db file."

echo "ERROR!  Something went wrong with the backup process.
Aborting …"
```

שגיאות אזהרה:

```
echo "WARNING:  Cannot find GIT binary file. This will prevent
some of the functionality of the script."

echo " WARNING:  Something went wrong with the auto checkout.
 Ignoring  ..."
```

1.7 שימוש בתו * בסקריפט כמו שהוא

כידוע, לתו * יש תפקיד מיוחד ופירושו "הכל". לכן, אם ננסה להשתמש בו כמו שהוא מבלי לתחום אותו בין גרשיים (בודדים או כפולים) נקבל ביטוי שהינו תוצאה של חישוב (כגון רשימת הקבצים והתיקיות שבמיקום הנוכחי) ולא אותו כשלעצמו.

אם נרצה למשל, לשלוח אותו כפי שהוא לפקודה אחרת כדי שהיא תהיה זו שתחשב אותו במקום Bash, אנו צריכים להגדיר הגדרה מיוחדת בסקריפט שתאפשר לנו להשתמש בתו * כטקסט ולא כביטוי בעל משמעות מיוחדת.

הגדרה זו מתבצעת על ידי עדכון פרמטר פנימי של Bash באמצעות הפקודה set.

לפני שנרצה להשתמש בתו * כטקסט בתוך הסקריפט, נכתוב את השורה:

```
set -o noglob
```

פירוש הפקודה הוא "שנה (set) את האופציה (o) 'גלובאליות' (noglob) לכבויה (-)".

מספיק פעם אחת בתוך הסקריפט (ולכן רצוי כמה שיותר בהתחלה על מנת שנזכור ששינינו את ההגדרה בסקריפט זה).

ההגדרה תשפיע רק על הקוד בסקריפט הספציפי שבו היא תופיע והיא לא תשמר באופן גלובאלי (ניתן להחזיר את המצב לקדמותו במפורש על ידי הפקודה ההפוכה set +o noglob).

2. פקודה בתוך פקודה (גרש הפוך לעומת $( )

אחד מהיתרונות הגדולים של Bash בהשוואה לשורת הפקודה של ווינדוס הוא שניתן להריץ פקודה בתוך פקודה ולהעביר מיידית פלט של פקודה אחת לתוך פקודה אחרת.

ניתן לבצע את ההרצה הפנימית ב 2 דרכים:

1. באמצעות שימוש בתו גרש **הפוך** (במקלדת הוא נמצא משמאל למקש 1 בקצה השמאלי יחד עם התו ~). לא להתבלבל עם הגרש הרגיל - גרש הפוך נראה כך: ` בעוד שגרש רגיל נראה כך: '.

   <u>דוגמא:</u>

   ```
   echo "Hello `whoami`" '!'
   ```

   <u>פלט:</u>

   ```
   Hello ohadm !
   ```

   כפי שניתן לראות בדוגמא, הפקודה whoami בוצעה לפני ה- echo בזכות השימוש בגרשיים ההפוכים וכך הודפס למסך שם המשתמש הנוכחי ולא המילה 'whoami' עצמה.

2. באמצעות שימוש בביטוי שמורכב משילוב בין הסימן $ לבין סוגריים עגולים.

   <u>דוגמא:</u>

   ```
   echo "Hello $(whoami)" '!'
   ```

   הפלט כמובן זהה לפקודה לעיל.

   תכונה זו שימושית מאוד ובעלת כוח רב כפי שניתן לשער ולכן חשוב מאוד להכיר אותה.

3. מתמטיקה ב-Bash

יש דרכים רבות לבצע פעולות אריתמטיות ב- Bash.

לצורך הפשטות, גם כי לא צריך יותר מדרך אחת וגם על מנת להיות עקביים בקוד שלנו - נתמקד רק באחת מהן.

מדובר בפקודה בשם expr.

החלק החשוב ביותר שיש לזכור כאשר משתמשים בפקודה expr הוא שחובה להפריד עם רווחים בין הפקודה, הנתונים שנכנסים אליה והפעולה האריתמטית שבה משתמשים.

כלומר: expr [רווח] פעולה [רווח] ביטוי [רווח]

<u>תזכורת!</u>

לא לשכוח שאנחנו ב- Bash ולכן עלינו להשתמש בתו ה- ESCAPE עבור תווים שנרצה להשתמש בהם בתוך פקודה פנימית כגון expr כדי ש- Bash לא "יאכל" אותם ויפרש אותם בעצמו. תו שכזה הוא למשל התו '*' שפירושו "כפל" מתמטית ואז כדי להשתמש בו בתוך הפקודה expr נצטרך לבצע לו ESCAPE (ראו להלן).

<u>דוגמאות שימוש:</u>

נניח שהגדרנו את המשתנים הבאים:

```
NUM1=21
NUM2=7
```

אז נוכל לבצע את הפעולות הבאות:

<u>חיבור:</u>

```
ADDITION=`expr $NUM1 + $NUM2`

echo $ADDITION
```

<u>פלט:</u> 28

<u>חיסור:</u>

```
SUBSTRACTION=`expr $NUM1 - $NUM2`

echo $SUBSTRACTION
```

<u>פלט:</u> 14

<u>כפל:</u>

```
PRODUCT=`expr $NUM1 \* $NUM2`

echo $PRODUCT
```

<u>פלט:</u> 147

חילוק שלם:

```
QUOTIENT1=`expr $NUM1 / $NUM2`
QUOTIENT2=`expr $NUM2 / $NUM1`

echo $QUOTIENT1 $QUOTIENT2
```

פלט: 3 0

שארית:

נשנה את הערכים על מנת לקבל ערך גדול מאפס כשארית:

```
NUM1=9
NUM2=4

REMINDER=`expr $NUM1 % $NUM2`

echo $REMINDER
```

פלט: 1

4. צבעים ב-Bash

לפעמים נרצה לייצר פלט צבעוני. לדוגמא, אם אנו מחשבים סטאטוס של תוצאה שיש לה מספר מצבים, נוכל להשתמש בצבעים כדי להציג בצורה ברורה יותר את הסטאטוסים השונים. נוכל למשל לסמן סטאטוסים חיוביים בירוק וסטאטוסים שליליים באדום. דוגמא נוספת היא הצגת חלקים שונים של פעולות זהות עבור קבצים בעלי תפקידים זהים בצבעים שונים. ניתן לחשוב על מקרים נוספים בהם יהיה נוח לעבוד עם צבעים.

היות שמדובר בהגדרה ברמת מערכת, 2 פקודות ההדפסה - גם echo וגם printf - ידפיסו בצבע לאחר הגדרת הצבעים.

הצבעים מוגדרים בשתי דרכים: באמצעות פקודת מערכת בשם tput שיכולה לשנות את הגדרות הטרמינל או באמצעות הגדרת מערכת מיוחדת שמגדירה את הצבעים.

ברגע שמגדירים צבע מסוים, הוא נשאר מוגדר עד שמגדירים צבע מחדש. זה אומר שלאחר ששינינו הגדרה של צבע, ***הטרמינל כולו*** יקבל את אותו הצבע ויוצג בו מעתה. לכן, יש לאפס את הצבע לשחור מיד לאחר הדפסת הפלט שרצינו להדפיס בצבע הנוכחי.

דוגמאות שימוש:

אתחול הצבעים:

```
COLOR_YELLOW_ON_BLACK='\e[1;33;40m'
COLOR_TURQUOISE=$(tput setaf 6)
COLOR_PURPLE=$(tput setaf 5)
COLOR_BLUE=$(tput setaf 4)
```

```
COLOR_ORANGE=$(tput setaf 3)
COLOR_GREEN=$(tput setaf 2)
COLOR_RED=$(tput setaf 1)
COLOR_BLACK=$(tput sgr0)
```

הדפסה בצבעים:

```
OK_MSG='OK'
ERR_MSG='ERROR'
WARNING_MSG='WARNING'
```

הדפסה בירוק:

```
echo -e "Status: ${COLOR_GREEN} $OK_MSG ${COLOR_BLACK}"
```

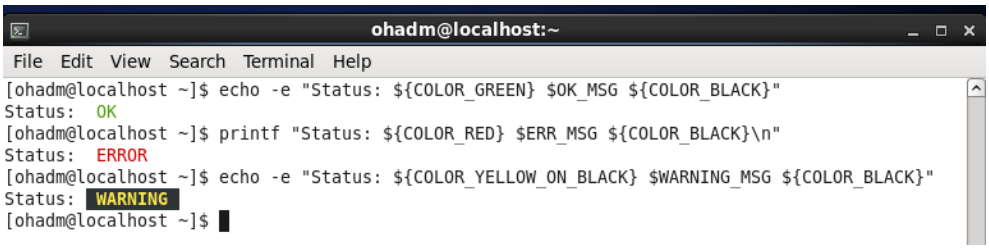
הדפסה באדום:

```
printf "Status: ${COLOR_RED} $ERR_MSG ${COLOR_BLACK}\n"
```

הדפסה בצהוב על גבי שחור:

```
echo -e "Status: ${COLOR_YELLOW_ON_BLACK} $WARNING_MSG
${COLOR_BLACK}\n"
```

פלט:



הערה:

לאפשרויות נוספות של צבעים, ראו נספח בסוף הספר.

5. קבלת קלט מהמשתמש

על מנת לקבל קלט מהמשתמש משתמשים בפקודה read.

השימוש שלה הינו פשוט למדי.

דוגמא:

```
read -p "Please select a key: " CHOICE

echo "Your choice was: $CHOICE"
```

וזה הכל ...

כפי שניתן לראות הבחירה של המשתמש נשמרת בדוגמא זו במשתנה בשם CHOICE.

6. משתנים מיוחדים

Bash מכיר ומנהל מספר משתנים מיוחדים שניתן להשתמש בהם.

המשתנים המיוחדים השימושיים ביותר הם חמישה:

1. $0 - שם הקובץ (שם הסקריפט ...)

<u>דוגמת שימוש:</u>

```
if [ "$1" == "" ]; then
        echo "Error! Paramer  missing."
        echo "Usage: $0 <options>"
fi
```

בדוגמא זו, אם המשתמש יריץ סקריפט שמכיל קוד זה ללא פרמטרים הסקריפט ידפיס למסך הודעת שגיאה בצירוף הדרך הנכונה לקרוא לו תוך שימוש בשמו על ידי המשתנה $0.

2. $? - קוד השגיאה האחרונה.

<u>דוגמת שימוש:</u>

```
echo $STATUS  |  grep "ok" > /dev/null

if [ "$?" == 0 ]; then
        echo "All ok!"
else
        echo "Problem …"
fi
```

בדוגמא זו מתבצעת בדיקה האם קיים הטקסט "ok" בתוך המשתנה STATUS. הבדיקה מתבצעת בפועל על ידי הרצת הפקודה grep ללא הדפסת פלט למסך ולאחר מכן בדיקת מצב המשתנה המיוחד $?. אם הוא שווה 0 זה אומר שהפקודה הקודמת בוצעה בהצלחה או במקרה הזה ש- grep מצאה את מה שרצינו לחפש.

כאשר משתמשים בשיטה זו חשוב לזכור לבדוק את המשתנה מיד לאחר הרצת הפקודה שרוצים לוודא שעברה בהצלחה (ולא להוסיף בטעות שום פקודה בין הפקודה לבדיקה כגון הדפסה למסך או שמירת משתנה זמני וכדומה כי אז היא תהיה זו שתיבדק) ולשלוח את הפלט שלה עצמה ל-NULL על מנת שלא יודפס פלט לא קשור למסך בעת הרצת הסקריפט.

3. $@ - רשימת כל הארגומנטים שנכנסו לסקריפט.

<u>דוגמת שימוש:</u>

```
$PATH_TO_AN_INNER_SCRIPT $@
```

בדוגמא זו הסקריפט הנוכחי קורא לסקריפט פנימי עם כל הפרמטרים שנכנסו אל הסקריפט הראשי על ידי שימוש במשתנה @$. באופן זה ניתן לשלוח את כל המשתנים שנכנסו מהמשתמש בקלות לעיבוד נוסף גם אם איננו יודעים בכמה משתנים מדובר.

4. #$ - מספר הארגומנטים הכולל שנכנס לסקריפט.

<u>דוגמת שימוש:</u>

```
if [ "$#"  -gt  0 ]; then
        echo  "We have arguments …"
fi
```

בדוגמא זו אנו בודקים האם המשתמש קרא לסקריפט עם ארגומנטים או לא. נשתמש בתנאי כזה למשל בסקריפט שמפעיל משהו מסוים כאשר הוא רץ ללא ארגומנטים כלל ומשהו אחר אם מכניסים לו נתונים נוספים (ואז נצטרך להבחין בין 2 המקרים).

5. $IFS ־ התו או התווים שנחשבים כהמחלקים של טקסט (Internal Field Seperator).

כדיפולט, Bash מגדירה את התווים המפרידים בין מילים בטקסט (DELIMITERS) להיות התווים שורה חדשה, TAB ורווח. אם נרוץ בלולאה על טקסט, ההפרדה בין הנתון הנוכחי לבין הנתון הבא תתבצע לאחר חלוקת הטקסט לפי רשימת התווים האלה.

הגדרה זו נשמרת במשתנה מערכת מיוחד בשם IFS.

אם נרצה, נוכל לשנות את ערכו וכך להגדיר רשימה או תו בודד אחר שישמש כמחלק טקסט.

<u>דוגמת שימוש:</u>

נגדיר את המשתנה הבא:

```
LIST='a 1 r 5 7 8 rfff defdsf 323'
```

אם ננסה להדפיס את תוכנו בלולאת for, נקבל שורות פלט לפי חילוק של התו רווח כי זו ההתנהגות הדיפולטיבית על פי ההגדרה של המשתנה IFS

הלולאה:

```
for i in $(echo $LIST); do
        echo $i
done
```

הפלט:

```
a
1
r
5
```

7

8

rfff

defdsf

323

אך לעומת זאת, אם נשנה את ערך המשתנה IFS לתו אחר, הטקסט יתחלק מחדש לפי התו החדש והתוצאות של הלולאה בהתאם.

למשל עבור הגדרת התו "מינוס" (-) כך:

IFS='-'

לאחר הרצה שנית של הלולאה נקבל:

a 1 r 5 7 8 rfff defdsf 323

וזאת היות שאין בטקסט מחלק שהינו התו מינוס ולכן הטקסט לא התחלק כלל וקיבלנו את כל התוכן של המשתנה בבת אחת.

<u>הערה:</u>

ערכו של המשתנה IFS מתאפס בחזרה לערכו הדיפולטיבי לאחר פתיחת טרמינל חדש.

לרשימה המלאה של המשתנים המיוחדים ראו נספח בסוף הספר.

7. הרצת מספר פקודות במקביל בשורה אחת

הרצת מספר פקודות במקביל יכולה להיות אופציה מאוד נוחה וחשובה במקרים מסוימים.

מקרים כאלה כוללים למשל הרצת פקודה מורכבת מה- SHELL ישירות ולא דרך סקריפט על מנת לבודקה, הרצת מספר פקודות מרחוק (במיוחד אם האחת סותרת את השנייה כגון שחרור ובקשת כתובת IP משרת DHCP) והרצת פקודות שתלויות אחת בשנייה.

יש 2 דרכים להרצת כמה פקודות במקביל:

א. באמצעות התו נקודה פסיק (;).

ב. באמצעות <u>**צמד**</u> תווים של "וגם" (Ampersand - &&).

ההבדל בין 2 הדרכים הוא שבדרך השנייה הפקודה הבאה <u>לא</u> תרוץ אם הפקודה שקדמה לה לא הסתיימה בהצלחה.

מבנה כללי:

א.

```
cmd1 ; cmd2 ; cmd3  … ;  cmdN
```

ב.

```
cmd1  &&  cmd2  &&  cmd3  …  &&  cmdN
```

דוגמאות:

1. הדפסה למסך (דרך א)

```
echo Hello ; echo World
```

פלט:

```
Hello
World
```

2. הדפסה למסך (דרך ב)

```
echo Hello  &&  echo World
```

פלט:

```
Hello
World
```

3. זוכרים את משפטי ה- if? אמרנו שם שהסוגריים הם פקודה בפני עצמה.

אז כעת ניתן להבין מדוע יש שם נקודה פסיק:

שימוש מקבילי:

```
if [ -s "a_file" ]; then
        echo "File exists!"
fi
```

שימוש **לא** מקבילי:

```
if [ -s "a_file" ]
then
        echo "File exists!"
fi
```

4. for בשורה אחת (דרך א):

ניתן להריץ לולאות בשורה אחת ישירות משורת הפקודה ולא דרך סקריפט. כך קל יותר לדבג בעיות.

למשל:

```
for i in $(ls); do echo $i; done
```

5. if בשורה אחת (דרך א):

אותו דבר נכון גם עבור משפטי if:

```
if [ -d "/etc" ]; then echo "Dir exists!"; fi
```

6. מקביליות עם בדיקה (דרך ב):

ניסיון להעתיק קובץ והדפסת הודעת הצלחה למסך. במקרה של כישלון לא יודפס כלום היות שהפקודה השנייה לא תרוץ כלל.

```
cp a_file another_file 2>/dev/null && echo File Copied!
```

8. שימוש בתאריכים ושעות

על מנת שתהיה אחידות בקוד, רצוי מאוד להתאפס על פורמט אחיד של תאריך ושעה ובו להשתמש בקביעות.

בצורה זו גם לא נצטרך לפשפש באופציות השונות של יצירת תאריך ושעה בכל פעם מחדש. פשוט נעתיק את הפורמט שיצרנו לסקריפט הבא מהקודם או מתבנית וזהו.

ניתן גם להגדיר משתנה סביבה גלובאלי שיוגדר ברמת מערכת של פורמט תאריך ואותו להפיץ בכל הסביבה ואז תהיה נגישות תמידית לפורמט הרצוי.

<u>לדוגמא:</u>

```
DATE='date +%Y-%m-%d_%H-%M-%S'
```

הפורמט שהוגדר לעיל נראה כך:

```
2016-06-24_08-59-12
```

רצוי לבנות את הפורמט כמו בדוגמא, כאשר השנה ראשונה ולאחר מכן החודש כדי שיהיה יותר נוח לחפש קובץ לפי תאריך כאשר ישנם מספר קבצים יחד עם שמות שכוללים תאריך ושעה.

על מנת להפוך את ההגדרה למשתנה סביבה כל מה שצריך לעשות זה להשתמש בפקודה export לאחר הגדרת המשתנה.

כך:

```
DATE='date +%Y-%m-%d_%H-%M-%S'
export DATE
```

כמובן שנרצה את המשתנה כפקודה ולא כנתון של התאריך הנוכחי. לכן יש לשים לב לקרוא למשתנה תוך <u>הפעלה</u> של תוכנו באמצעות גרש הפוך (או באמצעות $ וסוגריים כפי שהוזכר לעיל) ולא סתם לקרוא לו.

למשל:

```
mv a_file a_file-`$DATE`
```

לפרטים נוספים ואפשרויות נוספות ניתן לעיין בפרטי הפקודה date.

שימוש בפורמט זמן מותאם אישית בפקודה ls:

יש פקודות שניתן להגדיר עבורן פורמט זמן מותאם אישית. אפשרות זו עוזרת כשנרצה להשתמש בפלט מיוחד של הפקודה בסקריפט למשל.

לדוגמא:

```
ls  -l  --time-style="+%Y-%m-%d"
```

9. מעבר בין תיקיות עם popd ו-pushd

כאשר כותבים סקריפטים שמבצעים פעולות על קבצים ותיקיות במערכת הקבצים, נדרש לשוטט בין מספר תיקיות שונות במיקומים שונים. אם נרצה לחזור בקלות למיקום שממנו הגענו, נוכל להשתמש בזוג הפקודות popd ו- pushd.

זוג הפקודות הזה בפועל מנהל מחסנית פשוטה המכילה רשימה של תיקיות שהכנסנו אליה. ברגע שנשלוף שם מהרשימה, המערכת אוטומטית תעביר אותנו למיקום של התיקייה שנשלפה ותמחק אותה מהמחסנית.

השימוש בתכונה זו הוא פשוט למדי ומאוד נוח.

הפקודה pushd מקבלת כפרמטר נתיב במערכת הקבצים ומכניסה אותו למחסנית. אם הנתיב שונה מהנתיב שבו אנו נמצאים כרגע, הפקודה גם תעביר אותנו אליו לאחר הכנסתו למחסנית.

הפקודה popd מעבירה אותנו לנתיב הראשון ששמור במחסנית ומוחקת אותו ממנה.

תרחיש קלאסי לשימוש בתכונה זו בסקריפט:

1. נגיע למיקום רצוי במערכת הקבצים שנרצה לחזור אליו שנית במהלך הסקריפט.
2. נבצע סט פעולות ראשונות באותו מיקום.
3. נשמור את המיקום הנוכחי שלנו במחסנית על ידי הפקודה:

```
pushd  .  >  /dev/null
```

4. נעבור למיקום חדש ונבצע בו פעולות.
5. נחזור למיקום ששמרנו על ידי הפקודה:

```
popd  > /dev/null
```

6. נבצע את שאר הפעולות שנרצה במיקום המקורי.

שימו לב שלא נרצה שהסקריפט ידפיס למסך את תוכן המחסנית הנוכחי לאחר כל אחת מן הפקודות האלה כי זה יהיה פלט מיותר ומבלבל וזה מה שיקרה כברירת מחדל אם לא נשלח למיקום אחר את הפלט של 2 הפקודות. לכן בדוגמא לעיל הפלט נשלח להתקן המערכת /dev/null וזה יגרום לאי הדפסתו.

10. ALIASים ב- Bash

מספר נקודות חשובות לגבי ALIASים:

1. לנצל כמה שניתן

היכולת לעבוד עם ALIASים יכולה לשפר משמעותית את העבודה. חבל לפספס אותה.

ניתן ורצוי ליצור ALIASים לכל הפקודות היומיומיות כולל גישה מהירה לסקריפטים ופקודות מורכבות.

ALIASים מאפשרים בנוסף לגישה מהירה גם התחמקות אלגנטית מטעויות הקלדה וגם הגדרת פלט של פקודות כך שיופיע באופן יותר קריא ושימושי.

<u>דוגמאות:</u>

```
alias  cd='pushd  . >/dev/null ; cd'
alias  cd..='cd  ..'
alias  dc='cd'
alias  sl='ls'
alias  du='du  -hs'
alias  free='free  -m'
alias  grep='grep  --color=always'
alias  df='df  -h'
alias  vab='/root/scripts/verify_and_backup.sh'
alias  u='/root/scripts/update_a_site_permissions.sh'
alias  tc='apachectl configtest'
alias  dirs='/bin/ls -l | grep  ^d'
alias  ports='netstat  -tulanp'
alias  a='alias'
alias  cls='clear'
alias  f='find  .  -iname'
alias  g='grep  -i'
alias  m='more'
alias  gl='git  log  --pretty=format:"%h %ad | %s%d [%an]"  --graph  --date=short  -p'
```

2. דילוג על ALIAS

לפעמים נרצה להפעיל את הפקודה המקורית ולא ALIAS שלה.

למשל: כברירת מחדל פקודות מסוכנות כגון rm מוגדרות ב- CentOS עם ALIAS ששואל את המשתמש האם הוא בטוח (זה לא המצב ב- Ubuntu). לפעמים נרצה למחוק מבלי להתעכב על אישור ביצוע הפעולה.

אם מוגדר ALIAS על השם המדויק של הפקודה, נוכל לקרוא לה בגרסתה המקורית ב-2 דרכים:

א. על ידי ESCAPE של ה-ALIAS וקריאה לפקודה עם תו ה-ESCAPE סלאש הפוך (\):

```
\ls
```

ב. על ידי קריאה לקובץ הבינארי של הפקודה עם נתיב מלא:

```
/bin/ls
```

3. הפעלה מתוך סקריפט

כברירת מחדל, ALIASים מוגדרים במערכת ש<u>לא</u> לעבוד מתוך סקריפטים.

המשמעות היא שכאשר נפעיל פקודות שכן מוגדר עבורן ALIAS מתוך סקריפט, המערכת לא תפעיל את ה- ALIAS אלא את הפקודה בתצורה המקורית שלה.

על מנת לאפשר את הפעלתם בכל זאת, ניתן לשנות הגדרת מערכת של Bash.

יש להוסיף את הפקודה הבאה בתחילת הסקריפט:

```
shopt -s expand_aliases
```

לאחר מכן נוכל לקרוא ל-ALIASים ישירות מתוך הסקריפט אך הם עדיין לא יעבדו מתוך משתנה אלא רק באופן ישיר על ידי קריאתם בשמם ממש.

4. הסרת ALIAS

על מנת להסיר ALIAS משתמשים בפקודה unalias:

```
unalias <שם>
```

11. --color=always או --color=auto

חלק מהפקודות מאפשרות שימוש בצבעים לנוחות קריאה. דוגמא אחת היא הפקודה grep.

אין סיבה לא להשתמש בצבע היות שאופציה זו מקילה מאוד במציאת חלק של פלט מתוך פלט גדול בטרמינל. יחד עם זאת, חשוב לשים לב לכך שצבעים מוגדרים באמצעות תווי טקסט רגילים שמציינים למערכת מה לצבוע. העברת פלט של פקודה שעובדת כרגע בצבע לתוך קובץ או לתוך פקודה כגון xargs, תשבש אותו ותוסיף בצדדי הטקסט גם תווים נוספים שהינם התווים שמגדירים את הצבעים. התוצאה תהיה קובץ משובש או פקודה שלא תעבוד ותחזיר שגיאה מוזרה ולא הגיונית כי המבנה שלה נכון ותקין ורק תווי הצבע שאפילו לא רואים אותם משבשים אותה. לכן, יש לזכור שלא להשתמש בצבעים במקרים אלו. נוכל לעקוף את השימוש בהם בין השאר על ידי שימוש בפקודה בגרסתה הקלאסית הנגישה באמצעות תו ה ESCAPE.

בכל אופן, הגדרת הצבע מתבצעת על ידי שימוש במתג בשם --color.

למתג זה 3 אפשרויות: never, always או auto.

never היא האופציה הכי מובנת וגם הכי לא מומלצת. השאלה היא מה ההבדל בין auto ל- always.

ובכן, ההבדל הוא כזה:

auto - הצג פלט בצבעים אך אם הפלט שורשר לפקודה נוספת, אל תציג יותר צבע.

always - תמיד הצג צבעים. גם כאשר פלט מפקודה שתומכת בצבע נשלח לפקודה אחרת או ליותר.

המשמעות היא, שאם נגדיר את auto, ברגע שנשרשר עם PIPE (|) פקודות, נאבד את הצבע לאחר השרשור. אך אם נעבוד עם always, תמיד יהיה לנו פלט בצבע והוא ישמר גם אם נשרשר כמה פקודות לאחר הפקודה שהחזירה פלט צבעוני.

המסקנה מכל זה היא שהכי כדאי להשתמש באופציה always היות שזה מאוד נוח להשתמש בצבעים כשעובדים עם פקודות SHELL וחלון פקודות שחור.

יש רק עוד משהו קטן שצריך לשים אליו לב - השימוש באופציה always עשוי לגרור אחריו תופעת לוואי. אם שורת הפלט האחרונה של הפקודה הצבעונית הייתה גם היא בצבע, היות שההגדרה תמיד תעביר הלאה את הצבע לאן שלא יהיה והצבע לא הספיק להתאפס כי לא היו תווים בפלט שלא היו צריכים להיצבע אחרי החלק הצבוע, זה עלול לשבש לנו את הצבע של ה- SHELL כולו ולשנות גם לו עצמו במקרים מסוימים את הצבע ואז כל ה- SHELL וגם כל מה שנקליד מאותה שנייה יהיה בצבע. למזלנו, נוכל בכל זאת להשתמש ללא חשש באופציה always המומלצת היות שבעיה זו נפתרת בקלות באמצעות פקודה בשם **reset** (שהוזכרה כבר לעיל) שפשוט מאפסת את ה- SHELL להגדרות ברירת המחדל שלו וכך למעשה נוכל לחזור לצבעים הרגילים שלנו בכל פעם שיצבעו לנו חלקים שלא רצינו שיצבעו.

לפרטים נוספים אודות צבעים ב-Bash, עיינו בסעיף 2 לעיל.

12. התחמקות מבעיות

12.1 התו " לעומת התו '

ישנו הבדל מהותי בין שני התווים לעיל. השילוב ביניהם מאפשר יצירה של ביטויים מורכבים של שורה אחת ללא בעיות הרצה.

<u>ההבדל:</u>

ביטוי בין גרשיים כפולים <u>כולל</u> חישוב של ביטויים אם הם קיימים.

בעוד ש-

ביטוי בין גרשיים בודדים <u>אינו כולל</u> חישוב של ביטויים אם הם קיימים והתווים בו מוצגים כולם כליטרלים (כמו שהם).

לכן כשנרצה ביטוי טקסטואלי נטו, נשתמש בגרשיים בודדים אך על מנת לחשב ביטויים כגון ערכי משתנים או פקודות נשתמש בגרשיים כפולים.

דוגמא:

נגדיר משתנה:

```
MY_VARIABLE='hello bash'
```

את תוכנו אנו מאתחלים תוך שימוש בגרשיים בודדים בהתאם למוסכמה כללית שהגדרנו לעיל. המוסכמה אומרת שמחרוזות יש לאתחל תמיד עם גרשיים בודדים על מנת להימנע מבעיות.

שימו לב שאין כאן סתירה בין סעיף זה למוסכמה שהוגדרה כי היא מתייחסת **לתוכן** של משתנה ולא לאופן הקריאה למשתנה (או במילים אחרות הסעיף מתייחס לשמו של המשתנה ממש ולא לתוכן שלו).

גרשיים כפולים:

פקודה:

```
echo  "$MY_VARIABLE"
```

פלט:

```
hello bash
```

פקודה:

```
echo  "`who`"
```

פלט:

```
ohadm    tty1        2016-06-16 08:57 (:0)
ohadm    pts/0       2016-06-16 09:00 (:0.0)
```

גרשיים בודדים:

פקודה:

```
echo  '$MY_VARIABLE'
```

פלט:

```
$MY_VARIABLE
```

פקודה:

```
echo  '`who`'
```

פלט:

```
`who`
```

12.2 תיחום משתנים עם {}

לפעמים צריך להצמיד חלק מפקודה למשתנה.

על מנת ש-Bash לא יפרש את הביטוי המחובר כביטוי אחד ויצור ממנו משתנה אחד עם שם ארוך, ניתן לתחום בין סוגריים מסולסלים את שם המשתנה וכך Bash ידע להפריד בין הביטויים.

דוגמא:

נגדיר את המשתנה:

```
APPEND_AFTER_LINE=2
```

לפני השימוש בסוגריים:

```
sed -i "$APPEND_AFTER_LINEa \
this line will be appended after line 2" myFile
```

בדוגמא ישנו ניסיון להוסיף שורה חדשה (APPEND) אחרי שורה 2 לקובץ בשם myFile. היות שעל פי מבנה הפקודה של sed, אנו חייבים להצמיד את מספר השורה הרצוי אל הפקודה a, אם לא נתחם את המשתנה בסוגריים מסולסלים לא תהיה לנו הפרדה בין המשתנה לפקודה ונקבל תוצאה בלתי רצויה כי Bash ינסה להשתמש במשתנה בשם "$APPEND_AFTER_LINEa" שאיננו מאותחל ולכן ערכו יהיה "כלום" (הוא יהיה ריק).

תזכורת!

הסלאש ההפוך הבודד שמופיע כאן בסוף השורה הראשונה תפקידו לומר ל- Bash "תמתין לקלט נוסף כי הפקודה עדיין לא הסתיימה". באופן זה ניתן להעתיק ולהדביק לתוך הטרמינל פקודות ארוכות מרובות שורות - כך הן נשארות קריאות.

פלט:

```
sed: -e expression #1, char 8: extra characters after command
```

כפי שניתן לראות אי התיחום גרם להרצת פקודת sed שגויה שגרמה להצגת שגיאה על המסך.

לאחר השימוש בסוגריים:

לעומת זאת אם כן נדאג לתחום בסוגריים את שם המשתנה אז תהיה הפרדה והפקודה תהיה תקינה ונכונה.

```
sed -i "${APPEND_AFTER_LINE}a \
this line will be appended after line 2" myFile
```

במקרה זה אין פלט כי יש שימוש במתג i– של sed שפירושו in-place או במילים אחרות "בצע את ההוספה ישר לתוך הקובץ" אך גם לא תופיע שגיאה על המסך וזו עדות לכך שהפקודה בוצעה בהצלחה. אם יש בקובץ 2 שורות ומעלה ונסתכל בו, נוכל לראות שם את השורה שהוספנו.

חשוב לשים לב לעניין הזה ולהשתדל לתחום תמיד את המשתנים בסוגריים על מנת להימנע מראש ממצבים כאלה.

12.3 שימוש בפקודה which

הפקודה which מחזירה את המיקום של פקודה נתונה במערכת הקבצים. היא בסיסית ולכן מותקנת בכל מקרה במערכות לינוקס סטנדרטיות רבות. המשמעות היא שניתן לסמוך עליה ולהשתמש בה בסקריפטים על מנת לגשת לפקודות מערכת במקום להגדיר נתיבים אל פקודות באופן ידני בסקריפט שיצטרכו עדכון אם בעתיד נרצה להפעיל אותו במערכות אחרות.

באופן זה, הסקריפט לא יהיה תלוי במיקומים של מערכת כזו או אחרת ויגיע אל הפקודות תמיד.

<u>דוגמא:</u>

```
CP_BIN=`which cp`
echo "$CP_BIN"
```

פלט:

```
/bin/cp
```

<u>הערה:</u>

הפקודה which מחזירה גם aliasים ולכן יש לשים לכך לב כשמשתמשים בה כדי שלא יכנס למשתנה בטעות ALIAS או פלט מעורבב ואז לא נוכל להריץ את הפקודה דרך המשתנה.

<u>דוגמא:</u>

```
which ls
```

לפקודה ls למשל יש ALIAS ולכן הרצה שלה תחזיר 2 שורות פלט ולא רק אחת. הראשונה היא ה- ALIAS והשנייה היא המיקום של הפקודה.

פלט:

```
alias ls='ls --color=auto'
/bin/ls
```

אם נכניס את הפלט הזה כפי שהוא למשתנה, נקבל תוצאה לא רצויה (כל הפלט לעיל יכנס למשתנה כשורה אחת) ולא נוכל להפעיל דרכו את הפקודה.

על מנת לשמור לתוך את המשתנה רק את הנתיב אל הפקודה, נוכל להשתמש ב- grep כדי להתעלם מכל ה- ALIASים כך:

```
LS_BIN=`which ls | grep  -v  alias`
```

יחד עם זאת, יש לזכור שבעיה זו קיימת רק כאשר משתמשים בפקודה which ישירות בתוך ה- SHELL. אם נריץ אותה בתוך סקריפט אין את הבעיה הזו היות שבתוך סקריפטים כברירת מחדל אין שימוש ב- ALIASים.

הערה 2:

להשתמש ב- which זה רעיון טוב וזה אפילו מומלץ אבל צריך לזכור לטפל במקרה שבו הפקודה שחיפשנו על ידי which לא נמצאה.

במקרה כזה יש לנו 2 תופעות להתמודד איתן.

ראשית, תודפס שגיאה למסך דרך ערוץ השגיאות שלא נרצה להציג למשתמש.

ושנית, אם הפקודה לא נמצאה, עלינו להחליט האם להמשיך עם הסקריפט או לצאת כי בלי הפקודה אולי אין טעם.

התמודדות עם תופעה מספר 1

על מנת שלא תודפס שגיאה, פשוט ננתב שגיאות להתקן המערכת /dev/null ואז הן לא ישלחו למסך.

כך:

```
CP_BIN=`which cp 2>/dev/null`
```

בצורה זו אם הפקודה לא נמצאה במערכת לא יודפס כלום למסך למרות שחזרה שגיאה והמשתנה יהיה ריק.

התמודדות עם תופעה מספר 2

אמרנו שאם לא נמצאה הפקודה, המשתנה יהיה ריק אז נוכל פשוט לבדוק אם הוא ריק או לא ואם הוא ריק, נצא מהסקריפט.

הקוד יראה כך:

```
if  [ "$CP_BIN" == ""  ]; then
        echo "ERROR!  The  command  was not found! "
        echo "Aborting …"
        exit 1
fi
```

12.4 הרצת פקודה מורכבת בתוך סקריפט

לעתים קורה, שכאשר הפקודה מורכבת מדיי היות שנעשה בה שימוש במספר פקודות שונות יחדיו כשלכל אחת מהן יש דרישות משלה לגבי האופן שבו יש להריצה, היא לא תעבוד כמו שצריך באופן ישיר ותעשה בעיות.

על מנת להתחמק מהבעיה ולדאוג שהיא בכל זאת תרוץ ללא בעיות ותחזיר לנו את מה שביקשנו, ניתן לשלוח לחלון Bash נפרד את הפקודה במקום להריצה כפי שהיא בדומה למה שעשינו לעיל כאשר הגדרנו לפקודה find לדוגמא שורה מורכבת להרצה (שלחנו אותה לחלון נפרד של Bash).

ישנם 2 מקרים כלליים להרצת פקודה מורכבת:

1. ללא שימוש במשתנה נוסף.

2. תוך שימוש במשתנה נוסף שמכיל חלק מהפלט של פקודות קודמות.

1. ללא שימוש במשתנה נוסף

כפי שהוזכר לעיל, כאשר הפקודה מורכבת הדרך הנכונה ביותר שתמיד עובדת היא לשלוח אותה לחלון נפרד של Bash. לאחר מכן ניתן להחליט האם לשמור את הפלט למשתנה או להדפיסו.

בפועל, על מנת ששיטה זו תעבוד יש להקפיד על הכלל הבא:

לשמור את הפקודה המורכבת במשתנה בין גרשיים **בודדים** על מנת שכל התווים ישמרו כפי שהם כולל תווים מיוחדים (להסבר נוסף אודות גרשיים בודדים ראו לעיל בסעיף 12.1 - התו " לעומת ').

למשל:

```
CMD='cat  /etc/hosts  |  grep  "192.168"'
```

לאחר מכן נוכל לשלוח את הפקודה ל- Bash כך:

```
echo  $CMD | bash
```

ואם נרצה לשמור את הפלט למשתנה נוכל לעשות זאת עם גרש הפוך כך:

```
OUTPUT=`echo  $CMD | bash`
```

2. תוך שימוש במשתנה נוסף שמכיל חלק מהפלט של פקודות קודמות.

אם נרצה להשתמש בפלט שכבר שמרנו במשתנה אחר אנו בבעיה קטנה כי אמרנו לעיל שצריך לשמור את הפקודה בין גרשיים בודדים וכך נקבל את שם המשתנה ולא את תוכנו. לכן צריך לזכור שאם אנחנו רוצים לקחת גם נתון ממשתנה לתוך הפקודה כחלק מהבנייה שלה, עלינו **להפריד** בין המשתנה (או משתנים) לבין שאר הפקודה ולפתוח בכל פעם זוג גרשיים בודדים בנפרד עבור החלק בפקודה שעדיין נחשב כמחרוזת פשוטה ולא כביטוי שיש לו משמעות מיוחדת כגון משתנה.

למשל:

נניח שנרצה להשתמש בפקודה מהדוגמא הקודמת אך במקום לקרוא ישירות לקובץ, נקרא לו דרך משתנה כך:

```
HOSTS_FILE=/etc/hosts

CMD='cat  '$HOSTS_FILE'  |  grep 192.168'
```

שימו לב להפרדה בין 2 החלקים. החלק הראשון בפקודה עד המשתנה (עד התו $ לא כולל) הוא מבחינתנו מחרוזת בשלב זה ולכן הוא יהיה תחום בין זוג גרשיים ראשון. לאחר מכן אנו רוצים לקבל את **התוכן** של המשתנה שלנו ולכן נשאיר אותו ללא כל גרשיים ולבסוף ניצור עוד חלק תחום בין זוג גרשיים שניים עבור החלק השני של הביטוי שגם הוא כרגע מחרוזת רגילה. כך נקבל במקביל גם את הפקודה המורכבת כמחרוזת פשוטה וגם את התוכן של המשתנה שלנו שיכנס גם הוא למשתנה CMD.

כעת נוכל לשלוח את הפקודה כרגיל לחלון Bash נפרד בדיוק כמו מקודם:

```
echo  $CMD | bash
```

הקפדה על דרך ההרצה הזו תפתור מאליה וללא סיבוכים מיותרים בעיות רבות של קונפליקטים בין סוגי גרשיים ו/או תחביר של פקודות שונות כאשר משתמשים בכמה מהן במקביל (וזה קורה לא מעט כאשר מנצלים כמו שצריך את היכולות של המערכת).

13. יכולות נוספות שיש ל- Bash

13.1 התיקייה הקודמת (התו -)

ניתן להגיע אל התיקייה הקודמת באמצעות שימוש בתו מינוס (-).

Bash שומר את המיקום של התיקייה האחרונה שהיינו בה.

על מנת להגיע אליה ניתן להשתמש בפקודה:

```
cd  -
```

13.2 שרשור פקודות עם התו PIPE (|)

זו יכולת בסיסית ושכיחה ביותר בלינוקס אבל חייבים לציין אותה מפאת חשיבותה.

ניתן לשרשר פלט מפקודה אחת לפקודה אחרת על מנת לעבוד על תוצאה של פקודה. כך למעשה נוכל לסנן פלט ולהשיג רק את החלק הרצוי שלו על ידי שרשור של מספר פקודות שונות באמצעות התו PIPE (|).

למשל:

```
ls  |  wc  -l
```

זוהי פקודה פשוטה שמשרשרת את הפלט של ls ל- wc באמצעות התו PIPE (|) על מנת למנות ולהציג את מספר הקבצים והתיקיות במיקום הנוכחי.

13.3 האופרטור *

באופן כללי האופרטור * פירושו "כל האפשרויות".

ב- Bash עצמו ולא בתוך פקודה ספציפית כלשהיא, האופרטור * מתורגם לרשימת כל הקבצים והתיקיות ביחס למיקום נוכחי או מיקום אחר שצוין.

לכן ניתן להשתמש בו לביצוע פעולה בודדת על מספר קבצים או תיקיות במכה אחת.

מה שחשוב לזכור כאן הוא שמי שמפרש את האופרטור הוא Bash ולא פקודה כזו או אחרת ולכן ניתן לשלב אותו בכל פקודה שהיא כדי להתייחס לרשימה (אלא אם כן נעשה בו שימוש בתוך פקודה ממש כאמור).

למשל:

```
ls *

cat /etc/pam.d/*

mv /tmp/*.txt /dev/shm
```

13.4 האופרטור ?

האופרטור ? פירושו תו אחד כלשהו בהכרח.

ניתן להשתמש בו ליצור ביטויים שמגדירים משפחה ספציפית של שמות קבצים או תיקיות. כמובן שניתן לשלב אותו עם האופרטור * לעיל.

בנוסף ניתן להשתמש בו על מנת להגביל קבוצת שמות עם שם מינימלי ואז Bash יבצע סינון מתאים ויוריד מהרשימה את השמות שלא עומדים בשם המינימלי שהוגדר.

גם כאן, מי שמפרש את האופרטור הוא Bash עצמו.

למשל:

שמות שמתחילים ב- b ומכילים בסך הכל 5 תווים:

```
ls b????
```

כל הקבצים שיש לשמם סיומת בת 3 תווים:

```
cat /etc/sysconfig/*.???
```

13.5 הפעלת פקודה על רשימה והאופרטור {}

כידוע, הרבה פקודות תומכות בהפעלה בודדת על רשימה. אם נרצה לדוגמא להעתיק יותר מקובץ אחד, נוכל לעשות זאת בפקודת cp בודדת כגון:

```
cp file1 file2 file3 /path/to/copy/to
```

אבל, מה יקרה אם נרצה להעתיק רשימת קבצים בעלי שם חלקי משותף?

במקרה כזה נוכל להשתמש באופרטור {}.

האופרטור {} מאפשר להריץ פקודה על רשימה שיש בה חלק משותף מבלי לציין את החלק המשותף יותר מפעם אחת.

כלומר, את הפקודה לעיל נוכל לכתוב גם כך:

```
cp  file{1,2,3}  /path/to/copy/to
```

ניתן לראות שהחלק המשותף הוא המילה file והחלק הלא משותף שהוא המספור נמצא בין סוגריים מסולסלים.

במילים אחרות, בדוגמא זו החלק המשותף הוא **שם של קובץ**.

דוגמא נוספת:

```
mkdir   /tmp/{dir1,dir2,dir3}
```

בדוגמא זו, החלק המשותף הוא **נתיב במערכת הקבצים**.

פקודה זו תיצור במיקום tmp/ 3 תיקיות בשמות dir1, dir2 ו- dir3.

את הדוגמא השנייה ניתן לשכלל קצת יותר ולשלב את 2 החלקים המשותפים שלה (גם השם "dir" וגם המיקום tmp/) וליצור מחדש את הפקודה כך:

```
mkdir  /tmp/dir{1,2,3}
```

כך יוצא שבדוגמא זו, החלק המשותף הוא גם **שם של קובץ** וגם **נתיב במערכת הקבצים**.

13.6 הפקודה eval

הפקודה eval מאפשרת להריץ פקודה שנמצאת בתוך משתנה.

<u>למשל:</u>

הרצת הפקודות:

```
export  MY_CMD="echo  The  output  of  \'ls to /etc/cron.d\':  ;  ls"

eval  $MY_CMD
```

תגרום להצגת הפלט:

```
The  output  of  'ls to /etc/cron.d':
0hourly  raid-check  sysstat
```

<u>הערה:</u>

אין חובה להגדיר את המשתנה עם הפקודה export והפקודות לעיל יעבדו גם בלעדיה.

הפקודה export גורמת לשמירת המשתנה וערכו ברשימת המשתנים של המשתמש (משתני סביבה) הנגישה דרך הפקודה env.

13.7 האופרטור >>>

האופרטור >>> מאפשר לשלוח ערך של מחרוזת ישירות לפקודה.

למשל:

דוגמא 1

פקודה:

```
sh <<< "hostname"
```

פלט:

```
localhost.localdomain
```

דוגמא 2

הפקודה:

```
grep "search for this string in var" <<< $VAR
```

הינה זהה לחלוטין לפקודה:

```
echo $VAR | grep "search for this string in var"
```

13.8 יצירת קובץ תוך כדי הרצת פקודה

ניתן ליצור קבצים ב- Bash תוך כדי הרצת פקודה.

יצירת הקובץ מתבצעת באמצעות הביטוי (שם_קובץ)> כלומר שם הקובץ החדש בין סוגריים והסימן גדול מ- (>) בצמוד לסוגריים השמאליים.

למשל:

הדוגמא הבאה יוצרת 2 רשימות ממוינות מ- 2 סוגים של נתונים (קובץ ופלט של פקודה) על ידי יצירת 2 קבצים שנשלחים מיד לפקודת ההשוואה comm שמקבלת 2 קבצים ממוינים כקלט.

הפקודה תחזיר רשימה של שורות שקיימות ב- 2 הקבצים.

הפקודה:

```
comm -1 -2 <(cat list1.txt | sort) <(ls /tmp | sort)
```

הערות:

- <(cat list1.txt | sort) יצירת קובץ ממוין ראשון מקובץ טקסט.
- <(ls /tmp | sort) יצירת קובץ ממוין שני מהפלט של הפקודה ls.
- -1 מתג שמגדיר לפקודה comm להתעלם משורות שקיימות רק בקובץ הראשון.
- -2 מתג שמגדיר לפקודה comm להתעלם משורות שקיימות רק בקובץ השני.

דוגמא נוספת אך הפוכה - הצגת רשימה של שורות שלא קיימות באף אחד משני קבצים שנוצרים לפני הקריאה לפקודה מ 2 סוגים שונים של נתונים:

```
diff <(cat list1.txt) <(ls /var | grep "cnf")
```

13.9 יצירת קובץ עם שורות מרובות

ניתן ליצור קובץ עם יותר משורה אחת דרך ה- SHELL על ידי הגדרת ביטוי שמסמן את ההתחלה והסוף של תוכן הקובץ.

למשל:

```
cat >> mutlilined-input.txt << EOF
line one
line two
line three
EOF
```

בדוגמא זו, הביטוי המסמן הוא "EOF", אך ניתן להשתמש גם בשם אחר. כפי שניתן לראות הוא נמצא גם בהתחלה וגם בסוף של הפקודה cat. הפקודה גורמת ליצירת קובץ בשם mutlilined-input.txt שמכיל 3 שורות שנכנסו אליו דרך ה- SHELL עצמו.
ניתן להשמיט מהפקודה את החלק של המעבר דרך קובץ (את הביטוי multilined-input.txt >>) כדי לשלוח את הפלט מרובה השורות במקום אל קובץ אל התקן הפלט הדיפוליבי, הרי הוא המסך, אשר ידפיסו בשלמותו. זו דרך נוספת להדפיס מספר שורות בפקודת הדפסה בודדת בין בסקריפט ובין ישירות דרך הטרמינל.

13.10 התו סלאש הפוך (\) בפני עצמו

התו סלאש הפוך (\) - כאשר משתמשים בו בפני עצמו - מסמן ל- Bash שיש עדיין המשך לפקודה והיא טרם נגמרה. אם נקליד אותו ונלחץ על Enter, Bash לא יריץ עדיין את הפקודה שהקלדנו ויסמן לנו שהוא ממתין לקלט נוסף בשורה חדשה. באמצעותו ניתן לפשט פקודות ארוכות ומורכבות לבעלות מבנה קריא יותר.

כך למשל ניתן להפוך פקודה מבלבלת ומסורבלת כגון זו:

```
./configure  --prefix=/opt/httpd  --with-included-apr  --enable-ssl
--with-ssl=/opt/openssl-1.0.1e  --enable-ssl-staticlib-deps  --enable-
mods-static=ssl
```

לפקודה המובנת והמסודרת הזו:

```
./configure \
   --prefix=/opt/httpd \
   --with-included-apr \
   --enable-ssl \
   --with-ssl=/opt/openssl-1.0.1e \
   --enable-ssl-staticlib-deps \
   --enable-mods-static=ssl
```

חשוב לזכור שלא לכתוב שום תו כולל רווח לאחר הסלאש. רק שורה חדשה יכולה להופיע אחריו על מנת שהוא יתורגם כראוי.

13.11 הדרך הנכונה לשלוח תוכן של קובץ לפקודה

השיטה הרווחת להכניס קובץ לפקודה היא להשתמש בפקודה הידועה cat אך למעשה זו לא הדרך הנכונה לעשות זאת.

הפקודה cat נועדה לשרשר (המילה cat היא קיצור של המילה concatenate שפירושה בעברית "שרשור") מספר קבצים יחד ולא לשלוח קובץ לפקודה אחרת.

השימוש השגוי הזה בפקודה cat מוגדר כ- UUOC - קיצור של Useless Use Of Cat.

הדרך הנכונה להעברת תוכן של קובץ לפקודה הינה באמצעות REDIRECT (התו <).

<u>שימוש שגוי (UUOC):</u>

```
cat  myFile | grep "myWord"
```

<u>שימוש נכון:</u>

```
<myFile   grep "myWord"
```

שימו לב שאין PIPE כלל בדוגמא של השימוש התקין. ה- REDIRECT כבר גורם ל- Bash לשלוח את התוכן של הקובץ לפקודה.

<u>הערה חשובה:</u>

למרות כל האמור לעיל, בסביבות רגישות רצוי דווקא <u>שלא</u> להשתמש באופציה של redirect היות שניתן די בקלות להתבלבל בכיוון של תו ה- redirect ואז אנו עלולים בטעות לאפס קבצי מערכת והגדרות מה שלא יקרה אף פעם עם השיטה של cat. גם אם היא לא נכונה כל כך רעיונית, היא הרבה יותר בטוחה לשימוש.

13.12 לינקים סימבוליים כקיצור לפקודה פנימית של פקודה

ניתן להשתמש בלינקים סימבוליים כקיצור דרך לפקודה פנימית שנמצאת תחת פקודה ראשית.

שמו של הלינק חייב להיות זהה לשם של הפקודה הפנימית על מנת שהרצתו תגרום להרצתה.

המערכת בעצמה עושה שימוש כזה בסט הכלים שמנהלים מחיצות מסוג LVM.

lvm היא הפקודה במקרה זה ויש לינקים לכל מיני תת-פקודות שנמצאות בתוך הפקודה.

<u>להלן מספר דוגמאות של לינקים כאלה:</u>

```
lrwxrwxrwx. 1 root root      3 Feb 18 16:59 vgconvert -> lvm
lrwxrwxrwx. 1 root root      3 Feb 18 16:59 vgcreate -> lvm
lrwxrwxrwx. 1 root root      3 Feb 18 16:59 vgdisplay -> lvm
lrwxrwxrwx. 1 root root      3 Feb 18 16:59 vgexport -> lvm
lrwxrwxrwx. 1 root root      3 Feb 18 16:59 vgextend -> lvm
lrwxrwxrwx. 1 root root      3 Feb 18 16:59 vgimport -> lvm
lrwxrwxrwx. 1 root root      3 Feb 18 16:59 vgmerge -> lvm
lrwxrwxrwx. 1 root root      3 Feb 18 16:59 vgmknodes -> lvm
```

```
lrwxrwxrwx. 1 root root        3 Feb 18 16:59 vgreduce -> lvm
lrwxrwxrwx. 1 root root        3 Feb 18 16:59 vgremove -> lvm
lrwxrwxrwx. 1 root root        3 Feb 18 16:59 vgrename -> lvm
```

כדוגמא נוספת, נוכל לקחת את הפקודה openssl וליצור לינק סימבולי בשם "version" ואז הרצתו תגרום להרצת הפקודה "openssl version" שתציג לנו ישירות את הגרסה של openssl.

יצירת הלינק, הצגתו עם ls והרצתו:

```
ln -s `which openssl` version

lrwxrwxrwx 1 root root 16 21:57 2 אוג version -> /usr/bin/openssl

./version
```

פלט:

```
OpenSSL 1.0.2g-fips 1 Mar 2016
```

13.13 sh -c : הרצת פקודות מרובות או פקודות שלא ניתן להריץ באמצעות sudo

ישנם מקרים בהם נצטרך להריץ SHELL בתוך SHELL.

דוגמא אחת לכך היא הרצת מספר פקודות במקביל בתוך פקודה אחרת (ולא ישירות דרך ה-SHELL שאז כמובן אין בעיה) כגון בתוך הפקודה xargs.

דוגמא שנייה היא צורך בשימוש בפקודה עם הרשאות root כאשר בהכרח נרצה להריצה ממשתמש שאינו root ואז אנו חייבים את sudo אך הפקודה לא עובדת עם sudo. דוגמא לפקודה כזו היא הפקודה echo.

במקרים אלו, נוכל לקרוא ל- shell נוסף בתוך ה- shell הקיים עם המתג c- שמאפשר לשלוח ל- shell מחרוזת של פקודות שתתבצענה וה shell ייסגר מיד לאחר מכן בניגוד למצב הדיפולטיבי של shell שהינו אינטראקטיבי וממתין בקביעות בלולאה לקלט מהמשתמש.

דוגמאות:

1. פקודות מרובות בתוך xargs (הסבר מפורט על הפקודה xargs ניתן למצוא לעיל בחלק "כלי מערכת"):

```
ls -t *.gz | xargs -I {} sh -c 'echo {}: ; tar -tvf {} | \
grep "a file name" '
```

2. פקודת echo דרך sudoer להוספת שורה בקובץ /etc/hosts :

```
sudo sh -c "echo 127.0.0.1 myServer >> /etc/hosts"
```

13.14 האופרטור &

האופרטור & מאפשר להריץ פקודה ברקע (ב- background). כדי להשתמש בו פשוט מוסיפים לפקודה הרגילה שמריצים את התו & בסופה. האופרטור יגרום לשליחת הפקודה שתעבוד ברקע ונקבל מחדש את ה- SHELL מוכן לקבלת פקודות נוספות.

באופן זה נוכל להמשיך לעבוד כרגיל תוך כדי הרצת פקודה אחת או יותר שלוקחות זמן מבלי צורך להמתין עד תומן.

דוגמא נוספת לשימוש באופרטור היא היכולת להריץ פקודה שמפעילה תוכנה גרפית מה-SHELL מבלי לתקוע אותו במצב המתנה עד שהיא תיסגר (זו ההתנהגות הדיפולטיבית).

על מנת לחזור אל הפקודה שרצה ברקע או במילים אחרות להחזיר אותה אל החזית, משתמשים בפקודה fg (קיצור של המילה foreground - חזית). ניתן לתת לה פרמטר אם מדובר במספר פקודות שרצות כעת ברקע (הן ממוספרות החל מ 1).

<u>הערה:</u>

לעתים יתכן שהפעלת האופרטור על תוכנה גרפית לא תחזיר את ה-SHELL מיד ובמקום יופיעו נתונים שמגיעים מהתוכנה ללא הצגת ה- PROMT של ה-SHELL מחדש אך בפועל זה רק נראה כך. זה קורה כי התוכנה שולחת נתונים אודות הריצה הנוכחית שלה לפלט הסטנדרטי שהינו המסך ובמקרה זה זהו ה- SHELL. אם נלחץ על Enter במקרה כזה נקבל בחזרה את ה-SHELL כרגיל (לאחר שליחת התוכנה אל הרקע באמצעות האופרטור &).

13.15 קבלת קובץ או תיקייה כפרמטר לסקריפט

בלא מעט מקרים, נרצה להכניס לסקריפט קובץ או תיקייה כפרמטר על מנת לעבוד עליהם / בתוכם.

הקובץ או התיקייה יכולים להגיע במספר אופנים קבועים ולכן נוכל להגדיר קטע קוד עבור הכנתם לעבודה ולחזור עליו בכל פעם בסקריפטים שונים. באופן זה גם לא נצטרך להמציא לוגיקה עבור תהליך זה מחדש, גם נוכל לממש בקלות תהליך זה על ידי העתקה פשוטה של הקוד מהסקריפט הקודם, גם הקוד שלנו בכל הסקריפטים יהיה אחיד וגם נדע שהוא עובד באופן בטוח כי השתמשנו בו כבר בעבר.

נבחין בין 2 מקרים:

1. קבלת תיקייה

תיקייה יכולה להגיע מהמשתמש באופנים הבאים:

א. באמצעות האופרטור נקודה (.) שפירושו "התיקייה הנוכחית" .

ב. באמצעות נתיב מלא אל התיקייה החל מתיקיית השורש.

ג. באמצעות נתיב חלקי אל התיקייה החל מהמיקום הנוכחי.

לכן, על מנת לממש את כל המקרים השונים -

בשלב הראשון נבדוק באיזה אופן קיבלנו את התיקייה. נמיר את הביטוי "." לנתיב מלא אל התיקייה הנוכחית וגם תיקייה יחסית נשלים לנתיב מלא באמצעות הפקודה pwd.

בשלב השני נשמור למשתנה את הנתיב המלא אליה. שאר הסקריפט ימומש תוך התייחסות אל התיקייה דרך המשתנה.

במקרים ב' ו- ג' נוודא שמדובר בתיקייה שבאמת קיימת במערכת הקבצים.

לבסוף נבדוק האם נכנס ערך למשתנה שמייצג את התיקייה ואם הוא נשאר ריק פירושו של דבר שלא נכנס ערך תקין עבור תיקייה לסקריפט ונצטרך לצאת ממנו באמצע.

מקרה ב' נראה לכאורה מיותר היות שברור שנתיב יחסי יעבוד גם ללא המרתו לנתיב מלא אך עלינו תמיד לזכור שאנו מסתכלים על מקרים כלליים ורוחביים על מנת להשיג ג'נריות של הקוד (בניית תהליכים כך שיעבדו בכל מקום ובכל מקרה) ולכן יש צורך בהמרת נתיבים יחסיים למלאים כי עם נתיב יחסי רק הסקריפט הנוכחי יעבוד כמו שצריך אך אם נרצה למשל לקרוא לסקריפט אחר בתוך הסקריפט הנוכחי הנתיב היחסי כבר לא יעבוד אם הסקריפט הפנימי לא נמצא בדיוק בנתיב של הסקריפט הנוכחי ולכן חשוב להמירו תמיד לנתיב מלא.

הקוד הבא מציג דוגמא למימוש של התהליך:

```
if [ "$1" == "." ]; then
        DIR_PATH=`pwd`
else
        if [ -d `pwd`/"$1" ]; then
                DIR_PATH=`pwd`/"$1"
        else
                if [ -d "$1" ]; then
                        DIR_PATH=$1
                fi
        fi
fi

if [ "$DIR_PATH" == "" ]; then
        echo "ERROR! The given parameter '$1' is not a valid directory."
        echo "Aborting" …
        exit 1
fi
```

2. קבלת קובץ

קובץ יכול להגיע מהמשתמש באופנים הבאים:

א. באמצעות נתיב מלא אל הקובץ החל מתיקיית השורש.

ב. באמצעות נתיב חלקי אל הקובץ החל מהמיקום הנוכחי.

התהליך דומה מאוד לתיקייה למעט מקרה אחד שאין בקובץ ובדיקה אחת שאין בתיקייה. המקרה שאין בקובץ הוא האופרטור נקודה (.) והבדיקה שאין בתיקייה היא וידוא שהקובץ איננו תיקייה (היות שתיקייה היא סוג של קובץ אך קובץ איננו סוג של תיקייה).

לכן במקרה זה הקוד יראה למשל כך:

```
if ! [ -s "$1" ]; then
        if [ -s `pwd`/"$1" ]; then
                FILE_PATH=`pwd`/$1
        else
                FILE_PATH=""
        fi
else
        FILE_PATH=$1
fi

if [ -d "$FILE_PATH " ]; then
        echo "ERROR! The given parameter '$1' is a directory."
        echo "Aborting" …
        exit 1
fi

if [ "$FILE_PATH" == "" ]; then
        echo "ERROR! The given parameter '$1' is not a valid file."
        echo "Aborting" …
        exit 1
fi
```

14. קיצורי מקלדת שימושיים בטרמינל

14.1 Shift + Pg Up / Shift + Pg Dn - עיון במסכי הפלט הקודמים

הטרמינל שומר היסטוריה של מספר מסכי פלט אחורנית גם כשלא משתמשים בו מתוך הממשק הגרפי ששם יש אזור גלילה כמו בכל חלון גרפי אחר.

כך נוכל לבחון לאחור פלט שכבר נמחק מהמסך עקב הרצה של פקודה עם פלט רב וכדומה.

ניתן לעיין בהיסטוריה של הפלט שנכתב לתוך הטרמינל על ידי שימוש בצמד המקשים Shift + Page Up (להחזיק את SHIFT ולחזור על ההקשה על PAGE UP לפי הצורך על מנת להגיע אל מסכים נוספים) כדי לעיין בפלט קודם (כיוון מעלה) או על ידי שימוש בצמד המקשים Shift + Page Down (כיוון מטה, להחזיק את Shift ולחזור על ההקשה על Page Down לפי הצורך) כדי לחזור חזרה לפלט העדכני ביותר.

14.2 Ctrl + k - מחיקת התווים מהמיקום הנוכחי ועד סוף השורה

לחיצה על הצירוף Ctrl + k בתוך הטרמינל תגרום למחיקת כל התווים מהמיקום הנוכחי של הסמן ועד סוף השורה.

14.3 Ctrl + d - מחיקת התו הנוכחי

לחיצה על הצירוף Ctrl + d בתוך הטרמינל תגרום למחיקת התו הנוכחי (התו שהסמן נמצא עליו).

14.4 Ctrl + w - מחיקת המילה שלפני הסמן

לחיצה על הצירוף Ctrl + w בתוך הטרמינל תגרום למחיקת כל המילה שלפני הסמן (עד התו רווח הקרוב).

14.5 Ctrl + l - מחיקת כל המסך

לחיצה על הצירוף Ctrl + l (L קטנה) בתוך הטרמינל תגרום למחיקת כל המסך כולו (כמו הפקודה clear).

14.6 Ctrl + a - קפיצה לתחילת השורה

לחיצה על הצירוף Ctrl + a בתוך הטרמינל תגרום לסמן לקפוץ לתחילת השורה.

14.7 Ctrl + e - קפיצה לסוף השורה

לחיצה על הצירוף Ctrl + e בתוך הטרמינל תגרום לסמן לקפוץ לסוף השורה.

14.8 Ctrl + r - שליפת פקודה מהירה מההיסטוריה

ניתן לשלוף במהירות פקודה מתוך היסטוריית הפקודות על ידי שימוש בצירוף המקשים Ctrl + r. לוחצים על הצירוף, מתחילים להקליד חלק מפקודה שרוצים למצוא ואם היא קיימת היא תופיע בשלמותה ונוכל ללחוץ על Enter כדי להריץ אותה כפי שהיא, או על אחד מחיצי הצדדים (מקש חץ ימין או מקש חץ שמאל) על מנת לערוך אותה לפני הרצתה או לחזור על הצירוף Ctrl + r כדי להציג את הפקודה הבאה שמכילה את הביטוי שחיפשנו עד שנמצא את הפקודה הרצויה.

14.9 Ctrl + c - מניעת שמירת הפקודה בהיסטוריה

מלבד הפקודה הידועה של צמד המקשים Ctrl + c שמאפשר סגירה של תכניות והפסקתן באמצע, ניתן להשתמש בו גם למניעת שמירת הפקודה הנוכחית בהיסטוריה.

כברירת מחדל, כל הקלדה של תו בודד גורמת לשמירת הפקודה הנוכחית בהיסטוריה גם אם עדיין לא הרצנו אותה בפועל.

לכן אם נרצה רק לבדוק משהו באופן זמני מבלי לשמור שינויים על ידי הקלדת תווים בטרמינל מבלי להריץ בפועל פקודה כלשהי (כגון לוודא מיקום של נתיב במערכת הקבצים או שפקודה מסוימת קיימת), לאחר הבדיקה, נוכל ללחוץ על צמד המקשים Ctrl + c ואז הביטוי שבדקנו לא יישמר בהיסטוריית הפקודות.

14.10 Backspace - מחיקת פקודות מסוכנות מההיסטוריה

אם נאלצנו להריץ פקודה מסוכנת (כגון " \rm –r * " - בהקשר הזה מומלץ מאוד מאוד שלא להריץ פקודות כאלה. תמיד יש דרכים עקיפות), נוכל להשתמש בתכונת שמירת ההיסטוריה של Bash שהוזכרה בסעיף הקודם שבה כל תו שמוקלד נשמר מיידית גם אם לא הרצנו בפועל את הפקודה שהוקלדה רק באופן הפוך. נוכל לנצל תכונה זו למחיקת הפקודה מההיסטוריה דרך הטרמינל על ידי מציאתה ומחיקתה תו אחר תו <u>תוך השהיית הסמן בין מחיקה למחיקה</u>. באופן זה הפקודה המסוכנת תוחלף בפקודה החדשה הריקה שיצרנו ואז לא נסתכן בהרצת הפקודה המסוכנת שנית בטעות.

15. DEBUGGING

על מנת לדבג את הקוד, כל מה שצריך לעשות זה להוסיף את המתג x- לשורה הראשונה בסקריפט:

#!/bin/bash **-x**

לאחר הוספת המתג, מעתה כל הפעלה של הסקריפט תגרום להצגת הפקודות לפני ביצוען וכך ניתן לעלות על השורה שעושה בעיות.

<u>דוגמא:</u>

נניח שיש לנו את הסקריפט הפשוט הבא ששומר עותק של קובץ נתון שמתקבל כפרמטר:

```
#!/bin/bash

DATE_PREFIX=`date +%Y-%m-%d_%H-%M-%S-`

SAVE_IN=~/Desktop

if [ "$SAVE_IN" == "" ]; then
        echo "ERROR! Could not find the path to the backup folder!"
        echo "Please check the script settings. Aborting ..."
        exit 1
```

```
fi

if [ "$1" == "" ]; then
        echo  "ERROR! Wrong input."
        echo  "Usage: $0 <file_to_backup>"
        EXIT_CODE=1
else
        if ! [ -s  "$1" ]; then
                echo  "ERROR! the input '$1' is an invalid non-empty file."
                echo  "Aborting ..."
                EXIT_CODE=1
        else
                FILE_NAME=`basename $1`

                cp  $1  $SAVE_IN/$DATE_PREFIX$FILE_NAME

                ls  -l  $SAVE_IN/$DATE_PREFIX$FILE_NAME

                EXIT_CODE=0
        fi
fi

exit  $EXIT_CODE
```

ככה נראית קריאה ללא פרמטר לפני הוספת המתג:

```
 [ohadm@localhost Desktop]$ ./backup-a-file.sh
ERROR! Wrong input.
Usage: ./backup-a-file.sh  <file_to_backup>
```

וככה היא נראית לאחר ההוספה:

```
 [ohadm@localhost Desktop]$ ./backup-a-file.sh
++ date +%Y-%m-%d_%H-%M-%S-
+ DATE_PREFIX=2016-06-24_00-46-40-
+ SAVE_IN=/home/ohadm/Desktop
+ '[' /home/ohadm/Desktop == '' ']'
+ '[' '' == '' ']'
+ echo 'ERROR! Wrong input.'
ERROR! Wrong input.
+ echo 'Usage: ./backup-a-file.sh <file_to_backup>'
Usage: ./backup-a-file.sh <file_to_backup>
+ EXIT_CODE=1
+ exit 1
```

סקריפטים לדוגמא

סקריפט שנועד לרוץ בשרת שמריץ שרת WEB מסוג HTTPD ותפקידו להחזיר רשימת אתרים שהשרת מארח לפי חיתוכים שונים

הסקריפט מציג דוגמת שימוש בתיעוד (כללי בראש הסקריפט ופרטני בגופו), בפונקציות, במערכים ועוד. בנוסף הסקריפט מהווה דוגמא לסקריפט בסיסי שסקריפטים רבים יכולים לעשות בו שימוש.

שם הסקריפט:

get-sites-list.sh

תוכן הסקריפט:

```
#!/bin/bash

##############################################################
# General
##############################################################
#
# SCRIPT NAME get-sites-list.sh
# SCRIPT VERSION 1.0
#
# CREATION DATE 15-5-2016
# AUTHOR ohadm
#
# Description:
#
# This script gets a list of web sites by given properties based on vhost files this server serves.
#
#
##############################################################

# set the location of the vhost files here
VHOSTS_DIR="/opt/v"

if ! [ -d "$VHOSTS_DIR" ]; then
    echo "ERROR! '$VHOSTS_DIR' is not a valid dir."
    echo "Aborting ..."
    exit 1
fi

printAnArrLineByLine()
{
    # print output array

    OUTPUT=($@)
    COUNTER=1

    for i in "${OUTPUT[@]}"; do
        echo $i
```

```
    done
}

printAnArrWithSpaces()
{
    # print output array

    OUTPUT=($@)
    COUNTER=1

    for i in "${OUTPUT[@]}"; do
        printf "$i "

        EVEN=`expr $COUNTER % 2`

        if [ "$EVEN" == 0 ]; then
            printf "\n"
        fi

        COUNTER=$(expr $COUNTER + 1)
    done

}

printUsageInfoAndExit()
{
    echo -e "ERROR! Wrong input.\n"
    echo -e "Usage: $0 <site_protocol> <data_type>\n"
    echo -e "'site_protocol' options: \n"
    echo -e "1) all (http and https)"
    echo -e "2) https_only (if a site is also available via http it won't be listed)"
    echo -e "3) https (includes sites that support both protocols)"
    echo -e "4) http (only http sites)"

    echo -e "\n'data_type' options: \n"
    echo -e "1) dirs (site directories)"
    echo -e "2) urls (site urls)"
    echo -e "3) main-urls (only main urls)"
    echo -e "4) all (urls and dirs - not always applicable)"

    echo -e "\nExamples:\n"

    echo -e "Example 1: $0 all urls (e.g. for 'curl' testing)"
    echo -e "Example 2: $0 https_only dirs (e.g. for updating secure cookie header)"
    echo -e "Example 3: $0 https urls (e.g. for checking ssl settings with 'openssl')"
    echo -e "Example 4: $0 https main-urls (e.g. for checking certificates status with 'openssl')"
    echo -e "Example 5: $0 all all (e.g. for getting a table of sites info)"

    exit 1
}

printASiteDirectoryTakenFromAVhostFile()
{
    if [ -s "$1" ]; then
        if ! [ -d "$1" ]; then
```

```
            SITE_DIR=`cat $1 | grep -i "directory" | grep -v "<\/" | awk '{print \$2}' | tr -d ">" | tr -d
"\""`

            SITE_DIR=${SITE_DIR%/}

            echo $SITE_DIR
        fi
    fi
}

printSiteUrlsTakenFromAVhostFile()
{
    if [ -s "$1" ]; then
        if ! [ -d "$1" ]; then
            URLS_ARR=()

            echo $1 | grep "ssl" > /dev/null

            if [ "$?" == "0" ]; then
                PREFIX="https://"
            else
                PREFIX="http://"
            fi

            MAIN_URL=`cat $1 | grep -i "ServerName" | awk '{print \$2}'`
            URLS_ARR+=($PREFIX$MAIN_URL)

            ALIASES=`cat $1 | grep -i "ServerAlias" | grep -v \# | awk '{\$1=""; print \$0}'`

            TOKENS_ARR=(${ALIASES//\\n/ })

            for i in "${TOKENS_ARR[@]}"; do
                URLS_ARR+=($PREFIX$i)
            done
        fi
    fi

    printAnArrLineByLine ${URLS_ARR[@]}
}

printATableOfUrlsAndDirsTakenFromAVhostFile()
{
    if [ -s "$1" ]; then
        if ! [ -d "$1" ]; then
            TABLE_ARR=()

            SITE_DIR=`printASiteDirectoryTakenFromAVhostFile $1`

            SITE_URLS_ARR=`printSiteUrlsTakenFromAVhostFile $1`

            # Bug Fix: the array gotten from the previous func
            # comes back as one line instead of an actual array
            # because adding '\n' to a variable doesn't work
            # hence re-tokenizing by space is needed here:

            TOKENS_ARR=(${SITE_URLS_ARR//\ / })
```

```
            for i in "${TOKENS_ARR[@]}"; do
                LINE=$i\ $SITE_DIR

                TABLE_ARR+=($i\ $SITE_DIR)
            done
        fi
    fi

    printAnArrWithSpaces ${TABLE_ARR[@]}
}

if [ "$1" == "" -o "$2" == "" ]; then
    printUsageInfoAndExit
else
    case $2 in
        "dirs")
            CMD="grep -i -v redirect | xargs -I {} cat {} | grep -i directory | grep -v \"<\/\" | awk
'{print \$2}' | awk -F\> '{print \$1}'"

            PREFIX="no"
        ;;
        "urls")
            CMD="grep -i -v redirect | xargs -I {} cat {} | grep -v \# | grep -i
\"ServerName\|ServerAlias\" | awk '{print \$2}'"

            PREFIX="yes"
        ;;
        "main-urls")
            CMD="grep -i -v redirect | xargs -I {} cat {} | grep -v \# | grep -i \"ServerName\" | awk
'{print \$2}'"

            PREFIX="yes"
        ;;
        "all")
            if ! [ "$1" == "all" ]; then
                printUsageInfoAndExit
            else
                TABLE="yes"
            fi
        ;;
        *) printUsageInfoAndExit
        ;;
    esac

    COUNTER=0

    case "$1" in

        "all"|"https_only")
            if [ "$TABLE" == "yes" ]; then
                for i in $(ls $VHOSTS_DIR/*.conf | grep -iv redirect); do
                    printATableOfUrlsAndDirsTakenFromAVhostFile $i
                done
            else
```

```
                LIST_OF_ITEMS=`echo "ls $VHOSTS_DIR/*.conf | grep -v "ssl" | $CMD" | sh
2>/dev/null`
                if ! [ "$LIST_OF_ITEMS" == "" ]; then
                    for i in $(echo "$LIST_OF_ITEMS"); do

                        if [ "$1" == "https_only" ]; then
                            DELETE_ARR[$COUNTER]=$i
                        else
                            if [ "$PREFIX" == "yes" ]; then
                                SITES[$COUNTER]=http://$i
                            else
                                # trim the char '"' if it exists
                                CURRENT_SITE=`echo $i | tr -d "\""`


                                # trim last slash if it exists
                                CURRENT_SITE=${CURRENT_SITE%/}


                                SITES[$COUNTER]=$CURRENT_SITE
                            fi
                        fi

                        COUNTER=$(expr $COUNTER + 1)
                    done
                fi
            fi

            # the '&' tells bash not to exit the case and continue running the statements in it.
            ;&
        "https")

            # if the request is for 'all' sites then a prefix is needed but if the request is for https sites
only then for 'openssl' to work properly we can't use a prefix so this condition removes it

            if [ "$1" == "https" ]; then
                PREFIX="no"
            fi

            LIST_OF_ITEMS=`echo "ls $VHOSTS_DIR/ssl*.conf | $CMD" | sh 2>/dev/null`

            if ! [ "$LIST_OF_ITEMS" == "" ]; then
                for i in $(echo "$LIST_OF_ITEMS"); do

                    if [ "$PREFIX" == "yes" ]; then
                        SITES[$COUNTER]=https://$i
                    else
                        # trim the char '"' if it exists
                        CURRENT_SITE=`echo $i | tr -d "\""`


                        # trim last slash if it exists
                        CURRENT_SITE=${CURRENT_SITE%/}


                        SITES[$COUNTER]=$CURRENT_SITE
```

```
                fi

                COUNTER=$(expr $COUNTER + 1)
            done
        fi

        # remove duplicate items
        NO_DUPS_ARR=(`echo ${SITES[@]} | tr " " "\n" | sort | uniq`)

        if [ "$1" == "https_only" ]; then
            for i in "${DELETE_ARR[@]}"; do
                OUTPUT_ARR=(${NO_DUPS_ARR[@]/$i})
            done
        else
            OUTPUT_ARR=${NO_DUPS_ARR[@]}
        fi
    ;;
    *) printUsageInfoAndExit
    ;;
esac

printAnArrLineByLine ${OUTPUT_ARR[@]}
fi
```

סקריפט שמדפיס תאריך פקיעה של תעודות SSL

סקריפט שעושה שימוש בסקריפט לעיל ועבור כל אתר HTTPS שנמצא על השרת הוא מדפיס את תאריך פקיעת התעודה שלו. סקריפט זה הוא בן של הסקריפט לעיל אך הוא גם אב של סקריפט אחר שמשתמש בו - הסקריפט הבא. זו בין השאר דוגמא להיררכיה של סקריפטים.

שם הסקריפט:

get-certs-dates.sh

תוכן הסקריפט:

```
#!/bin/bash

PATH_TO_GET_SITES_LIST_SCRIPT=/root/scripts/get-sites-list.sh

if ! [ -s "$PATH_TO_GET_SITES_LIST_SCRIPT" ]; then
  echo "ERROR! Internal problem!"
  echo "Cannot find the get-sites-list script! Aborting ..."
  echo "Make sure the script exists and accessible or edit this script."
  exit 1
fi

NUM_OF_SSL_SITES=`$PATH_TO_GET_SITES_LIST_SCRIPT https main-urls | wc -l`

if [ "$NUM_OF_SSL_SITES" -gt 0 ]; then
    URLS_AND_DATES=`$PATH_TO_GET_SITES_LIST_SCRIPT https main-urls | xargs -I {} sh
-c "printf '{}:  ' && openssl s_client -connect {}:443 -servername {} 2>/dev/null | openssl x509 -
noout -dates | grep -i notafter && printf '#'"`

    echo $URLS_AND_DATES | while IFS= read -r -d $'#' i; do
```

```
        URL_AND_DATE=$i

        URL=`echo $URL_AND_DATE | awk -F= '{print \$1}' | awk -F: '{print \$1}'`

        DATE=`echo $URL_AND_DATE | awk -F= '{print \$2}'`


        DIFF=$(expr `echo $DATE | xargs -I {} date -d {} '+%s'` - `date '+%s'`)
        DAYS=`expr $DIFF / 60 / 60 / 24`

        if ! [ "$1" == "" ]; then
            printf "$URL=$DAYS\n"
        else
            printf "${URL_AND_DATE}(that's $DAYS day(s) to go)\n"
        fi
    done
else
    if [ "$1" == "" ]; then
        echo "NO SSL sites were found on this server."
    fi
fi
```

סקריפט ששולח מייל לרשימת נמענים לאחר שנשארו מספר ימים עד פקיעת תעודת SSL

את מספר הימים הרצוי מגדירים בסקריפט. כאמור הסקריפט עושה שימוש בסקריפט הקודם כמו גם בסקריפט בסיסי נוסף שמאפשר לשלוח מייל - זהו הסקריפט הבא.

שם הסקריפט:

notify-on-about-to-expire-certs.sh

תוכן הסקריפט:

```
#!/bin/bash

NUM_OF_DAYS_TO_NOTIFY_ON=14

MAIL_RECIEVERS_CSV_FILE=/root/scripts/staff.csv

PATH_TO_GET_CERTS_DATES_SCRIPT=/root/scripts/get-certs-dates.sh
PATH_TO_SEND_MAIL_SCRIPT=/root/scripts/send-mail.sh

if ! [ -s "$PATH_TO_GET_CERTS_DATES_SCRIPT" ]; then
  echo "ERROR! Internal problem!"
  echo "Cannot find the get-certs-dates script! Aborting ..."
  echo "Make sure the script exists and accessible or edit this script."
  exit 1
fi

if ! [ -s "$PATH_TO_SEND_MAIL_SCRIPT" ]; then
  echo "ERROR! Internal problem!"
  echo "Cannot find the send-mail script! Aborting ..."
  echo "Make sure the script exists and accessible or edit this script."
  exit 1
```

```
fi

NUM_OF_DATES=`$PATH_TO_GET_CERTS_DATES_SCRIPT raw | wc -l`

if [ "$NUM_OF_DATES" -gt 0 ]; then
    URLS_AND_DATES=`$PATH_TO_GET_CERTS_DATES_SCRIPT raw`

    for i in $(echo $URLS_AND_DATES); do
        URL_AND_DATE=$i

        URL=`echo $URL_AND_DATE | awk -F= '{print \$1}'`

        DAYS=`echo $URL_AND_DATE | awk -F= '{print \$2}'`

        if [ "$DAYS" -eq "$DAYS" ] 2>/dev/null
        then
            if [ "$DAYS" -le "$NUM_OF_DAYS_TO_NOTIFY_ON" ]; then
                echo "Sending mail for about site '$URL' ..."

                $PATH_TO_SEND_MAIL_SCRIPT $MAIL_RECIEVERS_CSV_FILE
"WARNING: An SSL Certificate is about to expire!" "The SSL for site '$URL' will expire in
'$DAYS' days! " > /dev/null
            else
                echo "Date OK for site '$URL'."
            fi
        fi
    done
else
    echo "NO SSL sites were found on this server."
fi
```

סקריפט ששולח מייל

הסקריפט הנוסף שהסקריפט לעיל עושה בו שימוש. זהו סקריפט שמשתמש בסקריפט אחר על מנת לשלוח מייל. במקרה זה מדובר בסקריפט בפייתון ולכן נמצא מחוץ לגבולות הספר.

שם הסקריפט:

send-mail.sh

תוכן הסקריפט:

```
#!/bin/bash

SMTP_SERVER_IP_ADDRESS=192.168.1.25

PYTHON_BIN_PATH=/usr/bin/python

PATH_TO_PYTHON_SEND_MAIL_WITH_ATTACHMENT_SCRIPT=/root/scripts/sendmail-
with-attachment.py

if ! [ -s "$PYTHON_BIN_PATH" ]; then
    echo "Internal Error! Check parameters! Exiting ..."
    exit 1
fi
```

```
if ! [ -s "$PATH_TO_PYTHON_SEND_MAIL_WITH_ATTACHMENT_SCRIPT" ]; then
    echo "Internal Error! Check parameters! Exiting ..."
    exit 1
fi

if [ "$1" == "" -o "$2" == "" -o "$3" == "" ]; then
    echo ERROR! Required parameters missing.
    echo Usage: $0 \<recievers_list_csv_file\> \<subject\> \<body\>
else
    if ! [ -s "$1" ]; then
        echo ERROR! \'$1\' is not a valid file!
        echo Aborting ...
        exit 1
    fi

    $PYTHON_BIN_PATH
    $PATH_TO_PYTHON_SEND_MAIL_WITH_ATTACHMENT_SCRIPT
    "$SMTP_SERVER_IP_ADDRESS" "$1" <(`echo dummy`) "$2" "$3"
fi
```

סקריפט שמחולל סיסמא אקראית לפי אורך שהוגדר לו

שם הסקריפט:

generate-a-random-password.sh

תוכן הסקריפט:

```
#!/bin/bash

if [ "$1" == "" ]; then
    echo ERROR: Wrong Input.
    echo Usage: $0 \<password_length\>
else
    RAND_CMD="cat /dev/urandom | tr -dc 'a-zA-Z0-9' | fold -w $1 | head -1"

    PASSWORD=`echo $RAND_CMD | sh`

    echo $PASSWORD | grep "[0-9]" | grep "[A-Z]" | grep "[a-z]" > /dev/nul

    while ! [ "$?" == "0" ]
    do
        PASSWORD=`echo $RAND_CMD | sh`

        echo $PASSWORD | grep "[0-9]" | grep "[A-Z]" | grep "[a-z]" > /dev/nul
    done

    echo $PASSWORD
fi
```

סקריפט שמוסיף כתובת DNS לקובץ HOSTS

שם הסקריפט:

add-a-site-to-hosts.sh

תוכן הסקריפט:

```
#!/bin/bash

if [ "$1" == "" ]; then
    echo "ERROR! Wrong input."
    echo "Usage: $0 <site_dir_or_url_without_protocol>"
    exit 1
else
    SERVER_IPS=(`ip addr | grep inet | grep -v host | awk '{print \$2}' | awk -F/ '{print \$1}' | grep -v
":"`)

    IP_ADDRESS=${SERVER_IPS[0]}

    if [ -d "$1" ]; then
        HOST_TO_ADD=`basename $1`
    else
        HOST_TO_ADD=$1
    fi

    cat /etc/hosts | grep $HOST_TO_ADD > /dev/null

    if [ "$?" == 0 ]; then
        echo $HOST_TO_ADD

        exit 0
    else
        if ! [ "$IP_ADDRESS $HOST_TO_ADD" == "" -a "$HOST_TO_ADD" == "" ]; then
            # we only add the first item of the array
            echo $IP_ADDRESS $HOST_TO_ADD >> /etc/hosts

            echo $HOST_TO_ADD

            exit 0
        else
            echo "Error!"
            exit 1
        fi
    fi
fi
```

סקריפט אינטראקטיבי שבודק תקינות של אתרי אינטרנט

הסקריפט עושה שימוש בפקודה curl והוא גם משתמש בסקריפט הקודם (אם הוא זמין) במקרה שבו שם ה- DNS של האתר הנבדק לא מוגדר בקובץ HOSTS.

שם הסקריפט:

curl-interactive-test.sh

תוכן הסקריפט:

```
#!/bin/bash

VHOSTS_DIR=/opt/v

CURL_BIN=`which curl`

PATH_TO_ADD_A_SITE_TO_HOSTS_SCRIPT=/root/scripts/add-a-site-to-hosts.sh

if [ "$CURL_BIN" == "" ]; then
	echo "ERROR! 'curl' was not found on the server!"
	echo "Aborting ..."
	exit 1
fi

if ! [ -d "$VHOSTS_DIR" ]; then
	echo "ERROR! '$VHOSTS_DIR' is not a valid dir."
	echo "Aborting ..."
	exit 1
fi


if ! [ -s "$PATH_TO_ADD_A_SITE_TO_HOSTS_SCRIPT" ]; then
	NO_ADD_TO_HOSTS_SCRIPT=yes
fi


COUNTER=1

IFS_BACKUP=$IFS

IFS=$'\n'

ls $VHOSTS_DIR/ssl*.conf 1>&2 2>/dev/null

if [ "$?" == 0 ]; then
	for i in $(cat $VHOSTS_DIR/ssl*.conf | grep -v \# | grep "ServerName\|ServerAlias" | awk '{print
$2}'); do
		URLS_ARR[$COUNTER]=https://$i

		COUNTER=$(expr $COUNTER + 1)
	done
fi

for i in $(ls $VHOSTS_DIR/*.conf | grep -v redirect | xargs -I {} cat {} | grep -v \# | grep
"ServerName\|ServerAlias" | awk '{print $2}'); do
	URLS_ARR[$COUNTER]=http://$i

	COUNTER=$(expr $COUNTER + 1)
done


if ! [ "$1" == "" ]; then
	if [ "$1" -eq "$1" ] 2>/dev/null
	then
```

```
        echo "Running the commands '$CURL_BIN -k ${URLS_ARR[$1]}' and '$CURL_BIN -I -k
${URLS_ARR[$1]}' ..."

        echo "$CURL_BIN -k ${URLS_ARR[$1]}" | sh
        echo "$CURL_BIN -I -k ${URLS_ARR[$1]}" | sh
    else
        echo "ERROR! Invalid input."
        echo "Aborting ..."
        exit 1
    fi
else

    COUNTER=1

    echo -e "\nPlease pick a site to test with curl:\n"

    for i in "${URLS_ARR[@]}"; do
        echo $COUNTER\) ${URLS_ARR[$COUNTER]}

        COUNTER=$(expr $COUNTER + 1)
    done

    echo q\) quit

    echo -e "\nYour Choice: "

    read NUM

    if [ "$NUM" == "q" ]; then
        echo "Exiting on user's request. Goodbye."
    else
        if [ "$NUM" -eq "$NUM" ] 2>/dev/null
        then
            echo "Running the commands '$CURL_BIN -k ${URLS_ARR[$NUM]}' and
'$CURL_BIN -I -k ${URLS_ARR[$NUM]}' ..."

            #echo "$CURL_BIN ${URLS_ARR[$NUM]}" | sh

            $CURL_BIN -I -k ${URLS_ARR[$NUM]} &>/dev/null

            if ! [ "$?" == "0" ]; then
                if ! [ "$NO_ADD_TO_HOSTS_SCRIPT" == "yes" ]; then
                    SITE_TO_ADD=${URLS_ARR[$NUM]}


                    TOKENS_ARR=(${SITE_TO_ADD//\\// })

                    TOKENS_ARR=(`echo ${TOKENS_ARR[@]} | tr " " "\n"`)

                    $PATH_TO_ADD_A_SITE_TO_HOSTS_SCRIPT ${TOKENS_ARR[1]}
                fi
            fi

            echo "$CURL_BIN -k ${URLS_ARR[$NUM]}" | sh
            echo "$CURL_BIN -I -k ${URLS_ARR[$NUM]}" | sh
        else
```

```
            echo "ERROR! Invalid Input."
        fi
    fi
fi

IFS=$IFS_BACKUP
```

סקריפט שמגבה קובץ נתון

הסקריפט כבר הוזכר לעיל.

הסקריפט מדגים גם החזרת קוד שגיאה ל- Bash. סקריפט שקורא לו יוכל לדעת באמצעות קוד שגיאה זה האם להמשיך בתהליך או לא.

על מנת לדעת היכן לשמור את קובץ הגיבוי, הסקריפט קורא לסקריפט אחר שמחזיר את הנתיב שבו יש לשמור את הקובץ. זו דוגמא לגנריות שמאפשרת להריץ בסביבות מסוגים שונים את אותו הסקריפט, ובכל סביבה קובץ הגיבוי יישמר במקום אחר על פי המימוש של הסקריפט שמחזיר נתיבים לפי מקרים. חשוב רק להקפיד על כך שהסקריפט שמחזיר את הנתיבים לפי המערכות השונות ימצא בכל אחת מהן ויחזיר נתיבים יחסיים נכונים ואז הסקריפטים הקוראים לו הם גנריים מבחינתנו כי הקוד שלהם לעולם נשאר זהה ומכיל שורה קבועה שקוראת לסקריפט בסיסי כגון זה.

<u>שם הסקריפט:</u>

backup-a-file.sh

<u>תוכן הסקריפט:</u>

```
#!/bin/bash

DATE_PREFIX=`date +%Y-%m-%d_%H-%M-%S-`

SAVE_IN=`/root/scripts/get-backup-folder.sh`

if [ "$SAVE_IN" == "" ]; then
    echo "ERROR! Could not find the path to the backup folder!"
    echo "Please check the script settings. Aborting ..."
    exit 1
fi

if [ "$1" == "" ]; then
    echo "ERROR! Wrong input."
    echo "Usage: $0 <file_to_backup>"

    EXIT_CODE=1
else
    if ! [ -s "$1" ]; then
        if [ -s `pwd`/"$1" ]; then
            FILE_PATH=`pwd`/$1
        else
            FILE_PATH=""
        fi
    else
        FILE_PATH=$1
```

```
    fi


    if ! [ "$FILE_PATH" == "" ]; then

        FILE_NAME=`basename $1`

        cp $1 $SAVE_IN/$DATE_PREFIX$FILE_NAME

        ls -l $SAVE_IN/$DATE_PREFIX$FILE_NAME

        EXIT_CODE=0
    else
        echo "ERROR! the input '$1' is an invalid non-empty file."
        echo "Aborting ..."

        EXIT_CODE=1
    fi
fi

exit $EXIT_CODE
```

סקריפט שמעדכן קובץ הגדרות של אתר דרופל 7

הסקריפט עושה שימוש בסקריפט הגיבוי שקדם לו.

שם הסקריפט:

update-drupal-7-settings-file.sh

תוכן הסקריפט:

```
#!/bin/bash

SETTINGS_FILE_NAME=settings.php

DATABASE_SECTION_VERIFY_LINE='$databases = array ('

SETTINGS_SET_DATABASE_NAME_PARAM_NAME='database'
SETTINGS_SET_DATABASE_SERVER_ADDRESS_PARAM_NAME='host'
SETTINGS_SET_DATABASE_USER_NAME_PARAM_NAME='username'
SETTINGS_SET_DATABASE_PASSWORD_PARAM_NAME='password'

PATH_TO_BACKUP_A_FILE_SCRIPT='/root/scripts/backup-a-file.sh'

if ! [ -s "$PATH_TO_BACKUP_A_FILE_SCRIPT" ]; then
    echo "ERROR! Internal problem!"
    echo "Cannot find the backup-a-file script! Aborting ..."
    echo "Make sure the backup-a-file script exists and accessible or edit this script."
    echo "Aborting ..."
    exit 1
fi

if [ "$1" == "" -o "$2" == "" -o "$3" == "" -o "$4" == "" -o "$5" == "" ]; then
    echo "ERROR! Wrong Input."
```

```
    echo "Usage: $0 <path_to_a_drupal_site_dir> <new_sql_server_ip> <new_sql_db_name>
<new_sql_username> <new_sql_password> [optional: type anything for this parameter to skip
original file backup]"

else
    if [ "$1" == "." ]; then
        SITE_DIR=`pwd`
    else
        SITE_DIR=$1
    fi

    if ! [ -d "$SITE_DIR" ]; then
        echo "ERROR! given dir is an invalid dir!"
        echo "Aborting ..."
        exit 1
    else
        cd $SITE_DIR
        #pwd
    fi

    if ! [ -s sites/default/"$SETTINGS_FILE_NAME" ]; then
        echo "INFO: the dir '$SITE_DIR' is not a valid drupal site."
        echo "Could not find a valid 'settings.php' file as
'$SITE_DIR/sites/default/$SETTINGS_FILE_NAME'"
        echo "Aborting"
        exit 1
    else
        SETTINGS_FILE_PATH=$SITE_DIR/sites/default/$SETTINGS_FILE_NAME

        cat $SETTINGS_FILE_PATH | grep "$DATABASE_SECTION_VERIFY_LINE" >
/dev/null

        if ! [ "$?" == 0 ]; then
            echo "ERROR! the settings.php file supplied seems invalid."
            echo "Aborting ..."
            exit 1
        fi

        if [ "$6" == "" ]; then
            echo "Backing up original file ..."

            $PATH_TO_BACKUP_A_FILE_SCRIPT $SETTINGS_FILE_PATH > /dev/null

            if ! ["$?" == 0]; then
                echo "ERROR!  Something went wrong with the backup process.
Aborting …"

exit 1
            fi
        else
            echo "NOTE: As requested, skipping backup."
        fi

        echo "Updating settings file with new data ..."
```

```
        sed --follow-symlinks -i "/\*/b;
s/\(\s*'$SETTINGS_SET_DATABASE_SERVER_ADDRESS_PARAM_NAME'\s*=>\s*\)'.*'/\1'$2'/
" $SETTINGS_FILE_PATH

        sed --follow-symlinks -i "/\*/b;
s/\(\s*'$SETTINGS_SET_DATABASE_NAME_PARAM_NAME'\s*=>\s*\)'.*'/\1'$3'/"
$SETTINGS_FILE_PATH

        sed --follow-symlinks -i "/\*/b;
s/\(\s*'$SETTINGS_SET_DATABASE_USER_NAME_PARAM_NAME'\s*=>\s*\)'.*'/\1'$4'/"
$SETTINGS_FILE_PATH

        sed --follow-symlinks -i "/\*/b;
s/\(\s*'$SETTINGS_SET_DATABASE_PASSWORD_PARAM_NAME'\s*=>\s*\)'.*'/\1'$5'/"
$SETTINGS_FILE_PATH


        echo "After the change:"

        cat $SETTINGS_FILE_PATH | grep -v "*" | grep -v "#" | grep
"'$SETTINGS_SET_DATABASE_SERVER_ADDRESS_PARAM_NAME'"

        cat $SETTINGS_FILE_PATH | grep -v "*" | grep -v "#" | grep
"'$SETTINGS_SET_DATABASE_NAME_PARAM_NAME'"

        cat $SETTINGS_FILE_PATH | grep -v "*" | grep -v "#" | grep
"'$SETTINGS_SET_DATABASE_USER_NAME_PARAM_NAME'"

        cat $SETTINGS_FILE_PATH | grep -v "*" | grep -v "#" | grep
"'$SETTINGS_SET_DATABASE_PASSWORD_PARAM_NAME'"

    fi
fi
```

סקריפט שמריץ בתיקייה שמכילה אתרי WEB סקריפט אחר שמתקבל כפרמטר

סקריפט זה משתמש אף הוא בסקריפט הבסיסי get-sites-list.sh שהוזכר לעיל.

הסקריפט מדגים את נושא הגנריות מ-2 כיוונים שונים במקביל:

1. הוא מאפשר להריץ סקריפט אחר שמתקבל כפרמטר.
2. הסקריפט שנשלח אליו כפרמטר ירוץ בנתיבים במערכת הקבצים על פי תוצאות שמתקבלות מהסקריפט הבסיסי ובאופן זה אם בעתיד נרצה לעדכן את הסקריפט הבסיסי, זה לא ישפיע על התקינות של סקריפט זה.

<u>שם הסקריפט:</u>

script-all.sh

<u>תוכן הסקריפט:</u>

```
#!/bin/bash
```

```
PATH_TO_GET_SITES_LIST_SCRIPT=/root/scripts/get-sites-list.sh

if ! [ -s "$PATH_TO_GET_SITES_LIST_SCRIPT" ]; then
  echo "ERROR! Internal problem!"
  echo "Cannot find the get-sites-list script! Aborting ..."
  echo "Make sure the script exists and accessible or edit this script."
  exit 1
fi

echoAListOfSitesDirsViaVhosts()
{
    $PATH_TO_GET_SITES_LIST_SCRIPT all dirs
}

runAScriptOnAllSites()
{
    # function params:
    # $1 = script full path
    # $2 = script params

    ALL_SITES=`$PATH_TO_GET_SITES_LIST_SCRIPT all dirs`

    for i in $(echo $ALL_SITES); do
        echo -e "\n============================================================"
        echo "$i"
        echo -e "============================================================\n"
        echo -e "Running the command '$1 $i $2' ...\n"
        echo "$1 $i $2" | sh
    done
}

if [ "$1" == "" ]; then
    echo "ERROR! Wrong input."
    echo "Usage 1: $0 <a_script_to_run_on_all_sites> [Optional: "arguments for the script. MUST be
between quotes or the script will take only the first word"]"
    echo "Usage 2: $0 show - show sites paths only"
else
    if [ "$1" == "show" ]; then
        echoAListOfSitesDirsViaVhosts $VHOSTS_DIR
    else
        if [ -s "$1" ]; then
            STATUS=`cat $1 | grep /bin/bash`

            if ! [ "$STATUS" == "" ]; then
                runAScriptOnAllSites $VHOSTS_DIR $1 $2
            else
                echo ERROR: \'$1\' is not a valid script. Aborting ...
            fi
        else
            echo ERROR: \'$1\' is not a valid file. Aborting ...
        fi
    fi
fi
```

סקריפט שמוסיף הגדרות לקובץ HTACCESS של APACHE

הסקריפט מדגים שימוש בפונקציות, במשפט CASE, בקבלת קובץ במספר אופנים (דרך מספר סוגי נתיבים), ובבניית פקודות באופן היררכי באמצעות שימוש במשתנים.

שם הסקריפט:

harden-htaccess-using-custom-errors.sh

תוכן הסקריפט:

```
#!/bin/bash

LIST_OF_ERROR_CODES_TO_WORK_WITH='403 404 400 500'

GREP_BIN=`which grep`
GREP_BASIC_CMD="$GREP_BIN -i"
GREP_IGNORE_COMMENTS="$GREP_BASIC_CMD -v \#"
GREP_find_AS_A_COMMENT="$GREP_BASIC_CMD \#"
GREP_find_LINE_NUMBER="$GREP_BASIC_CMD -n"

sed_BIN=`which sed`
sed_BASIC_CMD="$sed_BIN --follow-symlinks -i"

HARDENED_HTACCESS_FILES_LOCATION=/root/scripts/htaccess

DIRECTIVE_TO_WORK_WITH='ErrorDocument'
CUSTOM_ERROR_FILE_SPEC='/error.htm'
DRUPAL_ERROR_FILE_SPEC='/index.php'

APPEND_AFTER_LINE=16

showCurrentStatus()
{
    # params:
    # $1 - htaccess file spec

    echo -e "\nCurrent Status:"
    echo -e "----------------\n"

    $GREP_BASIC_CMD $DIRECTIVE_TO_WORK_WITH $1

    echo -e "\n"
}

removeAllDirectives()
{
    # params:
    # $1 - htaccess file spec

    $sed_BASIC_CMD "/${DIRECTIVE_TO_WORK_WITH}/d" $1
}

resetDirectivesToDrupalDefault()
{
    # params:
```

```
    # $1 - htaccess file spec

    removeAllDirectives $1

    $sed_BASIC_CMD "${APPEND_AFTER_LINE}i $DIRECTIVE_TO_WORK_WITH 404
$DRUPAL_ERROR_FILE_SPEC" $1
}

hardenDirectives()
{
    # params:
    # $1 - htaccess file spec

    removeAllDirectives $1

    SITE_PATH=`dirname $1`

    if ! [ -s "$SITE_PATH""$CUSTOM_ERROR_FILE_SPEC" ]; then

        cp $HARDENED_HTACCESS_FILES_LOCATION$CUSTOM_ERROR_FILE_SPEC
$SITE_PATH
    fi

    for i in $(echo $LIST_OF_ERROR_CODES_TO_WORK_WITH)
    do
        $sed_BASIC_CMD "${APPEND_AFTER_LINE}i $DIRECTIVE_TO_WORK_WITH $i
$CUSTOM_ERROR_FILE_SPEC" $1

    done
}

if [ "$1" == "" ]; then
    echo "ERROR! Wrong input."
    echo "Usage: $0 <htaccess_file_to_work_on> [optional: an_action]"
    echo "Example: $0 /data/foi.gov.il/.htaccess add"

    echo "an_action:"
    echo "add - harden settings using custom errors"
    echo "del - delete all custom errors related settings"
    echo "drupal - reset settings to drupal's default"
    echo "any other word or nothing at all - show current settings status"


    EXIT_CODE=1
else
    # check if the given argument is a relative path to a valid file
    if ! [ -s "$1" ]; then
        if [ -s `pwd`/"$1" ]; then
            FILE_SPEC=`pwd`/$1
        else
            FILE_SPEC=""
        fi
    else
        if ! [ -d "$1" ]; then
            FILE_SPEC=$1
        else
```

```
            echo "ERROR! the input '$1' is a folder."
            echo "Aborting ..."

            exit 1
        fi
    fi

    if ! [ "$FILE_SPEC" == "" ]; then

        case $2 in
            "drupal") resetDirectivesToDrupalDefault $FILE_SPEC

            showCurrentStatus $FILE_SPEC

            ;;

            "add") hardenDirectives $FILE_SPEC

            showCurrentStatus $FILE_SPEC

            ;;

            "del") removeAllDirectives $FILE_SPEC

            showCurrentStatus $FILE_SPEC

            ;;

            *) showCurrentStatus $FILE_SPEC

            ;;
        esac

        EXIT_CODE=0
    else
        echo "ERROR! the input '$1' is an invalid non-empty file."
        echo "Aborting ..."

        EXIT_CODE=1
    fi
fi

exit $EXIT_CODE
```

סקריפט שממיר תאריכים של השירות AUDITD לתאריכים קריאים

הסקריפט מדגים שימוש בפקודה של השפה PERL בתוך סקריפט של Bash.

שם הסקריפט:

convert-audit-times.sh

תוכן הסקריפט:

```
#!/bin/bash

cat /var/log/audit/audit.log | while read -r line; do

    TIME_STAMP=`echo $line | awk -F\( '{print \$2}' | awk -F. '{print \$1}'`

     READABLE_TIME=`echo -e "perl -e \"print scalar localtime "$TIME_STAMP\"`

    READABLE_TIME=`echo $READABLE_TIME | sh`

    echo $line | sed "s/$TIME_STAMP/$READABLE_TIME/g" >> new-audit.log
done
```

נספחים

רשימת התקנים מיוחדים

Device	Usage
/dev/null	Accepts and discards all input; produces no output.
/dev/zero	Accepts and discards all input; produces continuous stream of NUL (zero value) bytes.
/dev/full	Linux-specific; produces continuous stream of NUL (zero value) bytes when read, and returns a "disk full" message when written to.
/dev/random and /dev/urandom	Produce a variable-length stream of pseudo-random numbers.

רשימת משתני מערכת מיוחדים

Variable	Usage
$IFS	Sets the internal field separator. Default is whitespace chars (new line, tab and space).
$*	Expands to ALL parameters, starting from one. When the expansion occurs within double quotes, it expands to a single word with the value of each parameter separated by the first character of the IFS special variable.
$@	Expands to ALL parameters, starting from one. When the expansion occurs within double quotes, each parameter expands to a separate word.
$#	Expands to the number of positional parameters in decimal. E.g. $1 is the 1st parameter, $2 is 2nd etc.
$?	Expands to the exit status of the most recently executed foreground pipeline.
$-	A hyphen expands to the current option flags as specified upon invocation, by the set built-in command, or those set by the shell itself (such as the -i).
$$	Expands to the process ID of the shell.
$!	Expands to the process ID of the most recently executed background

Variable	Usage
	(asynchronous) command.
$0	Expands to the name of the shell or shell script.
$_	The underscore variable is set at shell startup and contains the absolute file name of the shell or script being executed as passed in the argument list. Subsequently, it expands to the last argument to the previous command, after expansion. It is also set to the full pathname of each command executed and placed in the environment exported to that command. When checking

רשימת אפשרויות IF

* לקוח מ- http://tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_01.html.

Condition	Meaning
[-a FILE]	True if FILE exists.
[-b FILE]	True if FILE exists and is a block-special file.
[-c FILE]	True if FILE exists and is a character-special file.
[-d FILE]	True if FILE exists and is a directory.
[-e FILE]	True if FILE exists.
[-f FILE]	True if FILE exists and is a regular file.
[-g FILE]	True if FILE exists and its SGID bit is set.
[-h FILE]	True if FILE exists and is a symbolic link.
[-k FILE]	True if FILE exists and its sticky bit is set.
[-p FILE]	True if FILE exists and is a named pipe (FIFO).
[-r FILE]	True if FILE exists and is readable.
[-s FILE]	True if FILE exists and has a size greater than zero.
[-t FD]	True if file descriptor FD is open and refers to a terminal.
[-u FILE]	True if FILE exists and its SUID (set user ID) bit is set.
[-w FILE]	True if FILE exists and is writable.
[-x FILE]	True if FILE exists and is executable.
[-O FILE]	True if FILE exists and is owned by the effective user

Condition	Meaning
	ID.
[-G FILE]	True if FILE exists and is owned by the effective group ID.
[-L FILE]	True if FILE exists and is a symbolic link.
[-N FILE]	True if FILE exists and has been modified since it was last read.
[-S FILE]	True if FILE exists and is a socket.
[FILE1 -nt FILE2]	True if FILE1 has been changed more recently than FILE2, or if FILE1 exists and FILE2 does not.
[FILE1 -ot FILE2]	True if FILE1 is older than FILE2, or is FILE2 exists and FILE1 does not.
[FILE1 -ef FILE2]	True if FILE1 and FILE2 refer to the same device and inode numbers.
[-o OPTIONNAME]	True if shell option "OPTIONNAME" is enabled.
[-z STRING]	True of the length if "STRING" is zero.
[-n STRING] or [STRING]	True if the length of "STRING" is non-zero.
[STRING1 == STRING2]	True if the strings are equal. "=" may be used instead of "==" for strict POSIX compliance.
[STRING1 != STRING2]	True if the strings are not equal.
[STRING1 < STRING2]	True if "STRING1" sorts before "STRING2" lexicographically in the current locale.
[STRING1 > STRING2]	True if "STRING1" sorts after "STRING2" lexicographically in the current locale.
[ARG1 OP ARG2]	"OP" is one of -eq, -ne, -lt, -le, -gt or -ge. These arithmetic binary operators return true if "ARG1" is equal to, not equal to, less than, less than or equal to, greater than, or greater than or equal to "ARG2", respectively. "ARG1" and "ARG2" are integers.

VI Cheat Sheet

הפעלה:

vi file1 file2 ….. fileN

מצבי עבודה:

המצב	פקודת ההפעלה	הערות
מצב פקודה Command Mode	מקש Esc	במצב פקודה מקשי המקלדת השונים מהווים פקודות עבור vi, זהו המצב הדיפולטיבי לאחר הפעלת vi
מצב הקלדה Insert Mode	האות i או המקש Insert	הקלדת טקסט חופשי

מצבי הקלדה מיוחדים:

מצב הקלדה	פקודת ההפעלה	הערות
מזיז את הסמן תו אחד קדימה	האות a	פתיחת מצב הקלדה לאחר הזזת הסמן תו אחד קדימה
פותח שורה חדשה מתחת לסמן	האות o קטנה	פתיחת מצב הקלדה לאחר פתיחת שורה חדשה מתחת לסמן
פותח שורה חדשה מעל לסמן	האות O גדולה	פתיחת מצב הקלדה לאחר פתיחת שורה חדשה מעל לסמן

פעולות במצב פקודה:

כל פעולה ניתן לבצע מספר פעמים על ידי הצמדת המספר הרצוי של פעמים לפקודה:

#cmd למשל - 10y

מחיקה:

הפעולה	פקודת ההפעלה	דוגמאות והערות
מחיקת תו	x	הפקודה 5x תמחק 5 תווים
מחיקת מילה	dw	הפקודה 4dw תמחק 4 מילים, מילה היא כל התווים ממיקום הסמן הנוכחי ועד לתו רווח
מחיקת שורה	dd	הפקודה 2dd תמחק 2 שורות

העתקה והדבקה:

הפעולה	פקודת ההפעלה	דוגמאות והערות
העתקת מילה	yw	קיצור של yank word, מילה היא כל התווים ממיקום הסמן הנוכחי ועד לתו רווח
העתקת שורה	yy	הפקודה 6yy תעתיק 6 שורות
הדבקה	p	הדבקת התוכן שהעתקנו. ניתן להכפיל גם את הפקודה הזו עם מספר. 3p למשל תדביק 3 פעמים את כל מה שהעתקנו.

UNDO ו- REDO:

הפעולה	פקודת ההפעלה
undo	u
redo	Ctrl+r

דילוג בין קבצים:

הפעולה	פקודת ההפעלה	דוגמאות והערות
מעבר לקובץ הבא	:n	n=next. שימו לב לנקודתיים לפני האות
מעבר לקובץ הקודם	:prev	prev=previous. שימו לב לנקודתיים לפני האות
יצירת קובץ חדש	:n a-new-file-name	a-new-file-name הוא השם של הקובץ החדש שייווצר.

חיפוש, החלפה וקפיצת שורה:

הפעולה	פקודת ההפעלה	דוגמאות והערות
חיפוש	/wordToLookFor	הביטוי word/ יחפש בקובץ את המילה word ויקפיץ את הסמן לביטוי הראשון שנמצא. האות **n** קטנה תמצא את הביטוי **הבא** והאות **N** גדולה תמצא מחדש את הביטוי **הקודם**.
החלפה	:%s/findMe/replaceWithMe/g	הביטוי %s/org/com/g: יגרום להחלפת המילה org במילה com לאורך כל הקובץ. שימו לב לנקודתיים שיש לכתוב לפני הביטוי.

הפעולה	פקודת ההפעלה	דוגמאות והערות
קפיצת שורה	:lineNumber	הביטוי 35: יגרום להקפצת הסמן אל שורה מספר 35. שימו לב לנקודתיים שיש לכתוב לפני ציון המספר.

הוראות נוספות:

הפעולה	פקודת ההפעלה	דוגמאות והערות
הוספת מספרי שורה	:set number	שימו לב לנקודתיים שלפני הפקודה, ניתן להשתמש גם בחלק מהשם (לכתוב למשל: set nu:), על מנת להסיר את מספרי השורות ניתן להוסיף את המילה no לפקודה כך: :set nonumber
ביטול מצב תאימות עם vi המקורי	:set nocompatible	כברירת מחדל Ubuntu מגדירה את vim במצב תאימות ל- vi המקורי וכתוצאה מכך לא ניתן להשתמש בחיצים של המקלדת על מנת לשוטט בקובץ. פקודה זו מסירה הגדרה זו.
הצגת נתונים אודות קובץ	Ctrl + g	צירוף המקשים הזה גורם להצגת שורה עם פרטי הקובץ כגון שמו ומספר השורות בו.

שמירה ויציאה:

שימו לב לתו נקודתיים (:) שחייב להופיע לפני הפקודות

הפעולה	פקודת ההפעלה
שמירת השינויים	**:w**
שמירה בשם	**:w <file_name>**
שמירה ויציאה	**:wq**
יציאה תוך שמירת כל הקבצים הפתוחים	**:wqall**
שמירה ויציאה גם אם הקובץ הינו לקריאה בלבד	**:wq!**
יציאה ללא שמירה	**:q!**
יציאה	**:q**

אפשרויות צבעים ב-Bash

Basic Structure:

di=1;4;31;42

The first number is the style (1=bold), followed by a semicolon, and then the actual number of the color.

The above example means:

Make directories appear in bold (1) underlined (4) red text (31) with a green background (42).

Possible styles are:

0 = default colour

1 = bold

4 = underlined

5 = flashing text

7 = reverse field

40 = black background

41 = red background

42 = green background

43 = orange background

44 = blue background

45 = purple background

46 = cyan background

47 = grey background

100 = dark grey background

101 = light red background

102 = light green background

103 = yellow background

104 = light blue background

105 = light purple background

106 = turquoise background

All possible colors:

31 = red

32 = green

33 = orange

34 = blue

35 = purple

36 = cyan

37 = grey

90 = dark grey

91 = light red

92 = light green

93 = yellow

94 = light blue

95 = light purple

96 = turquoise

Kinds of files that can be set to a different color / style:

di = directory

fi = file

ln = symbolic link

pi = fifo file

so = socket file

bd = block (buffered) special file

cd = character (unbuffered) special file

or = symbolic link pointing to a non-existent file (orphan)

mi = non-existent file pointed to by a symbolic link (visible when you type ls -l)

ex = file which is executable (ie. has 'x' set in permissions).

*.rpm = files with the ending .rpm

אפשרויות הפקודה sed

: # label

= # line_number

a # append_text_to_stdout_after_flush - append text

b # branch_unconditional

c # range_change

d # pattern_delete_top/cycle - delete text

D # pattern_ltrunc(line+nl)_top/cycle

g # pattern=hold - greedy - repeat for every match

G # pattern+=nl+hold

h # hold=pattern

H # hold+=nl+pattern

i # insert_text_to_stdout_now

l # pattern_list

n # pattern_flush=nextline_continue

N # pattern+=nl+nextline

p # pattern_print

P # pattern_first_line_print

q # flush_quit

r # append_file_to_stdout_after_flush

s # substitute

t # branch_on_substitute

w # append_pattern_to_file_now

x # swap_pattern_and_hold

y # transform_chars

אפשרויות הקובץ /etc/fstab

<u>options (4th column)</u>

defaults: Allow everything quota, read-write, and suid on this partition.

noquota: Do not set users quotas on this partition.

nosuid: Do not set SUID/SGID access on this partition.

nodev: Do not set character or special devices access on this partition.

noexec: Do not set execution of any binaries on this partition.

quota: Allow users quotas on this partition.

ro: Allow read-only on this partition.

rw: Allow read-write on this partition.

suid: Allow SUID/SGID access on this partition.

user: Allow standard users to mount this fs ('nouser' is the opposite but it is also the default).

<u>dump (5th column)</u>

0 = Do not auto backup using the dump utility

1 = Auto backup using the dump utility

<u>pass (6th column)</u>

0 = Do not auto check this fs on boot using the fsck utility.

1 = Auto check this fs on boot using the fsck utility with high priority (should be used only for the root partition).

2 = Auto check this fs on boot using the fsck utility with low priority (after the root partition).

ת.ד. 5361 הרצליה 4615301 **הזמנות באתר**
info@hod-ami.co.il www.**hod-ami**.co.il

קטלוג 2021

מחירי מבצע, מחירים מעודכנים והנחות באתר הוד-עמי

	כולל	עמ'	מחיר*
אינטרנט - מפתחי אתרים/גרפיקה			
אמא, אבא - בניתי אתר באינטרנט (HTML)		256	29
הגדל את הכנסות העסק שלך באמצעות **פרסום בגוגל Google AdWords**		300	249
HTML5 המדריך לבניית אתרים ולמערכות WEB, הדור הבא – מהד' 4		337	139
The **Java** Tutorial סדנת לימוד	CD	768	179
JavaScript סדנת לימוד		586	159
מדריך **ASP.NET MVC 4**		514	199
ASP.NET 3.5 סדנת לימוד בשפות C# ו-VB		824	99
תכנות			
Angular 8 בעשרה ימים		158	177
Code Complete – מדריך מעשי לפיתוח תוכנה		288	139
לחפש באגים, מדריך מעשי לבודק תוכנה, מהד' 3		350	169
Visual C# 3.0 סדנת לימוד		656	99
ללמוד **C** - מהד' 3		480	139
שפת **אסמבלי** למחשב האישי, מהד' 2		314	89
PC - חומרה, תוכנה ורשתות			
מדריך Hacking ואבטחת מידע, מהד' 2		428	169
מדריך **חומרה ותוכנה** לטכנאי PC - מהד' 5 (כולל חלונות 7/8)		752	189
מדריך **רשתות** לטכנאי PC ולמנהלי רשת - מהד' 4		608	219
Windows			
Windows 8.1 מדריך למשתמש		544	39
Windows 8 מדריך למשתמש		438	19
Windows 7 צעד-אחר-צעד		272	19
LINUX			
LINUX למתקדמים, טיפים, טריקים ותכנות ב-BASH		226	189

* מחיר מומלץ לצרכן כולל מע"מ. הספרים נמכרים בהנחה בהוצאה

היכנס לאתר להתעדכן בספרים החדשים ובמחירי המבצע בהוצאה

תוכן עניינים ופרקים לדוגמה www.**hod-ami**.co.il

09-9564716

	כולל	עמ'	מחיר*
גרפיקה			
Flash – ספר הדרכה ותרגילים		122	64
אינדיזיין – ספר הדרכה ותרגילים		132	72
Illusatrator – ספר הדרכה ותרגילים		120	64
Photoshop צעד אחר צעד (ש/ל, למתחילים), מהד' 3		200	89
מדריך לתוכנת העיצוב והאנימציה **3ds max** (2 כרכים)	CD	1400	289
OFFICE			
יישומי סטטיסטיקה בגיליון אלקטרוני Excel		86	69
סטטיסטיקה יישומית – א'+ב' עדכון 7/2020		202	97
טבלאות ציר – ניתוח נתונים חכם		116	87
Access 2016 צעד אחר צעד		384	169
Word 2016 צעד אחר צעד		150	59
Excel 2016 צעד אחר צעד		336	129
עוד ספרים בגרסאות קודמות (2007 ו-2003) ניתן למצוא באתר הוד-עמי			
ניהול, כלכלה ושונות			
יסודות המימון שרוני ומופקדי		560	175
הבטחת איכות תוכנה, מעקרונות ליישום חדש		480	169
לחפש באגים, מדריך מעשי לבודק תוכנה, מהד' 3		350	169
ניהול ממוקד לעשות יותר עם מה שיש (כריכה קשה) - מהד' 4		358	133
לי זה עולה יותר (תמחיר) (כריכה קשה) - מהד' 3		368	129
מערכות מידע			
Oracle SQL יכולות מתקדמות		626	249
SAS (**Statistical Analysis System**) – ספר לימוד		256	169
בסיסי נתונים ושפת **SQL** – עקרונות ועיצוב		648	149
ניתוח מערכות מידע כולל מתודולוגיית ה- **UML**		818	229
המדריך העברי השלם **UML**		346	329
ספרים דיגיטליים			
לרכישת ספרים לצפייה בפורמט PDF היכנס לאתר לקטגוריה "ספרים דיגיטליים"			
קבצי תרגול לספרים			
קבצי תרגול לספרים שונים תמצא באתר בקטגוריה "קבצי תרגול לספרים"			

* קטלוג 2021. מחיר מומלץ לצרכן כולל מע"מ. הספרים נמכרים בהנחה בהוצאה

היכנס לאתר להתעדכן בספרים החדשים ובמחירי המבצע בהוצאה

תוכן עניינים ופרקים לדוגמה www.**hod-ami**.co.il

09-9564716