Introduction to Data Science

Course 094201

Lab 1:

Bash

Spring 2019

Linux Operating System (OS)

- Taken from: https://www.linux.com/what-is-linux.
- Linux is an operating system, just like Windows and MAC OS.
- The purpose of the operating system is to manage computer hardware and provide software access to it.
- Linux is quite efficient, open-source and light-weight and very widely used, not just for personal computers and servers, but also of other devices such as printers, smartphones (Android is a Linux based OS), various control systems and even vehicles.
- Linux has a graphical environment: X Window System (also called X11); provides similar graphical functionality to that in Windows OS.
- Linux also has a command line terminal that runs some shell, usually **bash** ("Bourne-again shell"), which is a programming language that allows to work directly with files, folders, run different utilities and more.

Motivation - שאלות נפוצות

מדוע אנחנו לומדים להשתמש במערכת הפעלה לינוקס?

- חלק ניכר מהשרתים בעולם הם שרתי לינוקס. לינוקס היא מערכת הפעלה יציבה וקלה יותר מ Windows ויש בה הרבה יותר שליטה על צריכת משאבי החומרה וכו'. בפרט, הקוד של לינוקס הוא פתוח וניתן לשינוי ויש הרבה כלים וחבילות חינמיות בפרט עבור Data Science שנכתבו עבורה ורצות בה ביעילות חישובית רבה. יש הרבה יתרונות, אתם מוזמנים להעמיק לבד.

מהו bash?

- bash הוא – shell תוכנת מעטפת המאפשרת למשתמש לעבוד עם מערכת ההפעלה ללא ממשק גרפי מורכב.
- ניתן להריץ פקודות אחת אחת, דרך שורת פקודה או לחלופין לרכז את הפקודות יחד בקובץ הרצה (script) ולהריץ אותו. שימו לב: הקובץ הוא פשוט אוסף של פקודות שמתבצעות ברצף.
- קבצי script אינם עוברים קומפילציה, אלא מתבצעים שורה שורה, ולכן בעיות מתגלות בזמן ריצה. אם יש שגיאה תקבלו הודעת שגיאה על המסך ותצטרכו להתמודד עם תוצאות הפעולה החלקית (בשל נפילה) של ה script שכתבתם.

מדוע אנחנו צריכים לדעת bash?

- **כי כל** data scientist **הוא קצת** hacker:

  צריך לשלוט בקלות באלפי קבצים, פורמטים שונים של נתונים, לדעת לנייד אותם, לבדוק אותם, להריץ כלים שונים כדי לבצע חקר של הנתונים. עבודה נכונה עם bash באה לתת לכם כלים מאוד משמעותיים בעבודה עם נתונים ומידע.

האם אתם צריכים לדעת את התחביר של bash?

- אתם צריכים לדעת תחביר בסיסי ומשם אתם לומדים לבד את הכלים שאתם צריכים. מה שחשוב להבין הוא שיש המון כלים שונים שניתן להריץ אותם יחד באמצעות ה bash.

Starting to Work with Linux

We are working with a virtual machine on Azure servers. Instructions for connection placed at moodle.

- Start your machine and connect WinSCP to it.
- After connection open PuTTy terminal window.

First Command

- Type in your terminal: “pwd” + Enter
- The result will be the path of your home directory:

  /home/student
- In bash you can read the manual of any command, along with its options using the “man” command.
- Try it yourself:
  - Type: man pwd
  - View the manual page for the ”man” command. (http://www.computerhope.com/unix/uman.htm)

FILES, DIRECTORIES & COMMON COMMANDS

Meta Characters

- Meta characters have a special meaning in Unix. For example * and ? are meta characters.
- We use * to match 0 or more characters; a question mark ? matches a single character. Files are organized into directories.

```
$ls ch*.docx
```

- Example output:

```
ch_1.docx ch_2.docx ch_3.docx  ch_4.docx
```

- Try it yourself: run “ls” command in your terminal.

```
$ls /home/student/
```

Home Directory and Change Directory (cd) Command

- The directory in which you find yourself when you first login is called your home directory.
- You can go to your home directory anytime using the command "cd":
- Try to write and run:

```
$cd ~
```

- ~ indicates home directory.
- If you want to go to any other directory then use the command:

```
$cd /home/student
```

A **"trick"**: try to use the Tab button when you start typing.

Current Path, Create and Delete Directories

- Determine where you are within the filesystem hierarchy.

```
$pwd
/user/home/danny
```

- Directories are created by the command: `$mkdir mydir`
- Try to create a directory: `$mkdir dirname1 dirname2`
- Empty directories can be deleted using the “rmdir” command.

```
$rmdir dirname
```

```
$rmdir dirname1 dirname2 dirname3
```

- If the directory is not empty the “rm” command can be used with –r (recursive) option: `$rm –r dirname`
- The “rm” command can be also used to remove files. For “quiet” (without verification) delete, use: `$rm –f dirname`
- The options –f and –r can be combined: `$rm –fr dirname`

Be careful! This removal cannot be undone!

Create a New File - Touch

- The file system keeps for each file its access and modification times.
- If file.txt exists, ”touch” updates its access and modification times to the current time. If file.txt doesn't exist, it is created as a new, empty file. (http://www.computerhope.com/unix/utouch.htm)
- Step into the new directory:
- And create a new file:

```
$cd mydir
```

```
$touch file1.txt
```

- You just created file1.txt in your new directory.
- Open the file you created and write a few lines inside.

(**Wait.. What?** You can open it as you know from windows for now, just find the file and double click)

Directories and Files

- You can use "cd" to change to any directory by specifying a valid absolute or relative path.
- Step into the new directory:
- And come back (one level up):

```
$cd mydir
$cd ..
```

- Use "ls" to list the files in the directory

```
$ls
```

- Rename the directory ("mv" move command):

```
$mv mydir mynewdir
```

- The filename . (dot) represents the current working directory:

```
$mv ../file1.txt  .
```

  – This command moves file1.txt from the parent directory to the current directory.

Messages: Echo and Wall

- “echo” displays messages.
- Type “echo” without parameters to display the current echo setting.
- Try it out:

```
$ echo This is a message from myself
```

- http://www.computerhope.com/echohlp.htm
- “wall” sends a message to all terminals currently open
  - Note that you can have several terminal windows open
- Try it out:

```
$ wall Broadcast message
```

WORKING WITH FILES

Display the Content of a File

- Use the cat command to see the content of a file. (http://www.computerhope.com/unix/ucat.htm)

- Take a look at your file:

```
$ cat file1.txt
This is a unix file....I created it for the first
time.....
I'm going to save this content in this file.
```

- Display line numbers by using the -b option with "cat" command.
- Ever wonder about line numbers?

```
$ cat -b file1.txt
1   This is a unix file....I created it for the first time.....
2   I'm going to save this content in this file.
$
```

Edit a File

- One option is to open the file you created and write a few lines.
  - In a graphical environment you can open the file as you know from windows: open the directory, double click on the file and edit it.

- You can redirect an output of any command to a file using “>” and “>>”
- “>” overwrites the content of the file
- “>>” appends to the end of the file
- For example, type the following commands:

```
$ echo this is my file > file1.txt
$ echo this is a second line >> file1.txt
$ cat file1.txt
```

More, Head and Tail

- The next three commands display the content of textual files.
- "more" displays the contents of the file starting at a specified line (http://www.computerhope.com/unix/umore.htm)
- Take a look at your file from line 2 :

```
$more +2 file1.txt
 I'm going to save this content in this file.
$
```

- "head" displays the first ten lines of a file. (http://www.computerhope.com/unix/uhead.htm)
- See the 10 first rows:

```
$head file1.txt
```

```
$head -x file1.txt
```

- "tail" displays the last ten lines of a file. (http://www.computerhope.com/unix/utail.htm)
- See the 10 last rows:

```
$tail file1.txt
```

```
$tail -x file1.txt
```

You can specify the exact number of lines (replace x with a number)

Copying, Renaming and Deleting Files

- Make a copy of a file using the "cp" command. (http://www.computerhope.com/unix/ucp.htm)

  Copy the files from the first directory you created to the second:

```
$ cp /home/student/dirname1/* /home/student/dirname2/
```

Where to? (target)

What you want to copy (source)

- Change the name of a file using the "mv" command. (http://www.computerhope.com/unix/umv.htm)

```
$ mv old_file new_file
```

- Delete an existing file using the "rm" command. (http://www.computerhope.com/unix/urm.htm)

```
$ rm filename
```

Counting Words in a File

- Use the wc command to get a count of the total number of lines, words, and characters contained in a file. (http://www.computerhope.com/unix/uwc.htm)

Look at your file:

```
$ wc file1.txt
2  19 103 filename
```

- You can provide multiple files at a time.

```
$ wc filename1 filename2 filename3
```

- Note that text files encode ASCII characters. In general the binary information inside files can be decoded differently. Usually what indicates the OS how to decode a file is file's extension.
- The "wc" command with a –c option can be used to get the file size in bytes

History and Top

- “history” displays or manipulates the history list of commands with line numbers.

```
$history
```

- “fc” display the last 15 commands from the history list.

A “trick”: try to use the up arrow to see your last command.

- http://www.computerhope.com/unix/uhistory.htm

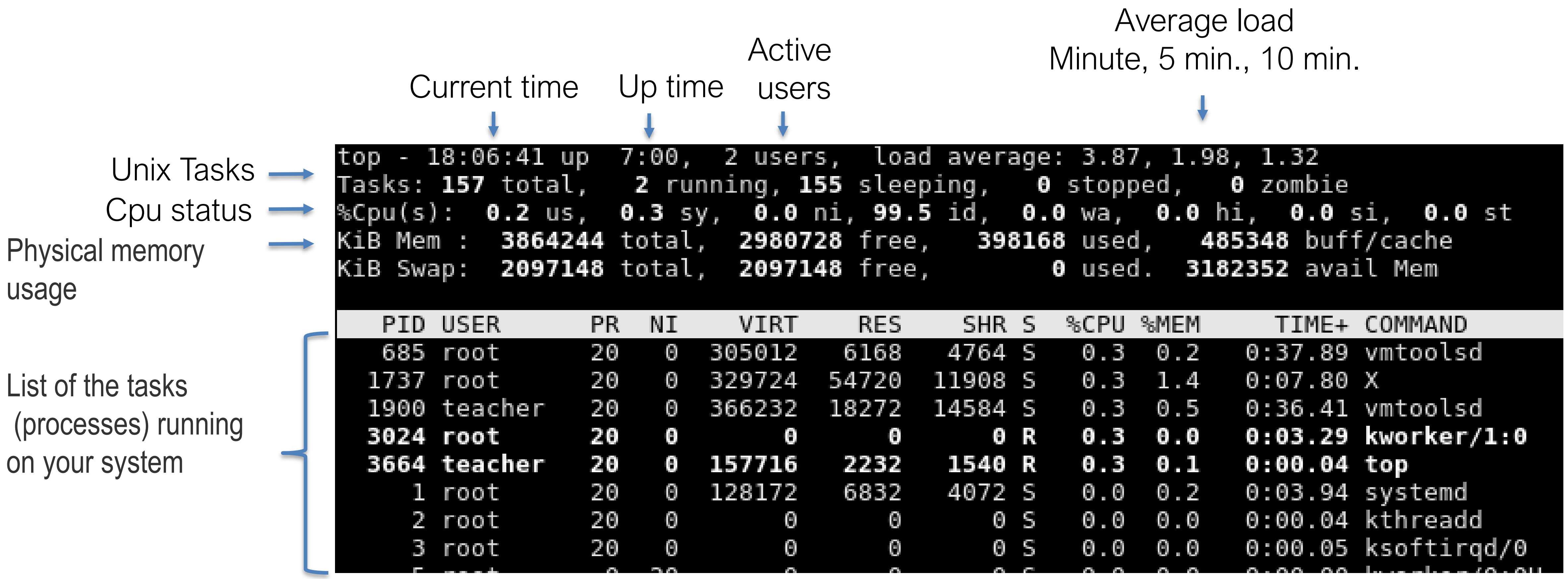
```
$fc -l
```

- If you want to see what is currently running on your computer write “top”

```
$top
```

  - What do we see?
  - Ctrl+c to exit.

Top


Current time
Up time
Active users
Average load Minute, 5 min., 10 min.
Unix Tasks
Cpu status
Physical memory usage
List of the tasks (processes) running on your system

```
top - 18:06:41 up  7:00,  2 users,  load average: 3.87, 1.98, 1.32
Tasks: 157 total,   2 running, 155 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.2 us,  0.3 sy,  0.0 ni, 99.5 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  3864244 total,  2980728 free,   398168 used,   485348 buff/cache
KiB Swap:  2097148 total,  2097148 free,        0 used.  3182352 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
  685 root      20   0  305012   6168   4764 S   0.3  0.2   0:37.89 vmtoolsd
 1737 root      20   0  329724  54720  11908 S   0.3  1.4   0:07.80 X
 1900 teacher   20   0  366232  18272  14584 S   0.3  0.5   0:36.41 vmtoolsd
 3024 root      20   0       0      0      0 R   0.3  0.0   0:03.29 kworker/1:0
 3664 teacher   20   0  157716   2232   1540 R   0.3  0.1   0:00.04 top
    1 root      20   0  128172   6832   4072 S   0.0  0.2   0:03.94 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.04 kthreadd
    3 root      20   0       0      0      0 S   0.0  0.0   0:00.05 ksoftirqd/0
```

See more information: https://www.unixtutorial.org/commands/top/

Pipeline

- Pipeline (“|”) allows to perform a sequence of processes chained together by their standard streams, so that the output of each process (*stdout*) feeds directly as input (*stdin*) to the next one.

```
$ process1| process2| process3
```

- Examples:
  - Try it out (type these commands one by one first, then together with a pipe “|”):

```
$ ls | head -3 | tail -1 > myoutput
$ cat myoutput
```

I/O Redirection

- Unzip a file: (http://www.computerhope.com/unix/utar.htm)
- Get to the file location and extract:

```
$ unzip lab1_students.zip
```

- To run a file (in the current directory) just write the following.

```
$ ./run_file < input > output
```

- Examples:

Try to run (does it look familiar?):

```
$ cd lab1_student/c++\ example/
$ chmod +x *
```

Next Slide..

```
$ ./main
```

```
$ cat ./main.cpp
```

```
$ ./main < in.txt
```

```
$ ./main < in.txt > out.txt
```

```
$ ./main < in_err.txt 2> out_err.txt
```

```
$ /.main < in_err.txt 1> out.txt 2> out_err.txt
```

Permissions

- File ownership is an important component of UNIX that provides a secure method for storing files.
  - Owner permissions − The owner's permissions determine what actions the owner of the file can perform on the file.
  - Group permissions − The group's permissions determine what actions a user, who is a member of the group that a file belongs to, can perform on the file.
  - Other (world) permissions − The permissions for others indicate what actions all other users can perform on the file.
- Read: Grants the capability to read (view) the content of the file.
- Write: Grants the capability to modify, or remove the content of the file.
- Execute: User with execute permissions can run a file as a program.

Changing Permissions (chmod command)

- Each file and directory have permissions (read/write/execute)
- To see file permissions, run “ls –l file”
- Change mode (chmod) command allows to change permissions.
- Note that you can not chmod everything: only files that you own in your directories.

chmod operator	Description
+	Adds the designated permission(s) to a file or directory.
-	Removes the designated permission(s) from a file or directory.
=	Sets the designated permission(s).

```
$chmod o+wx testfile
$chmod g=rx testfile
$chmod o+wx,u-x,g=rx testfile
```

My first Bash Script:

- Open a text editor, write the following code, and save (.sh):

```
#!/bin/bash
echo "this is my first bash script“
echo "I can write into files" > test.txt
cp test.txt test1.txt
echo "I am done, lets see what I have here“
ls
```

- Use the “cd” command if you are not in the right directory
- Now lets run it (did we forget anything? to see permissions. to grant permi `ls -l` on.)
- Run: `chmod +x myscript.sh`

`./myscript.sh`

My first Bash Script – Exercise :

- Change the script you created.
- Create a new file with the first line of test.txt.
- Add 2 new lines to the new file.
- Print a word count of all 3 files.

CONTROL SEQUENCES AND VARIABLES

Syntax - Variables

- You can set the value of a variable using the general syntax NAME=VALUE

```
$ x=5
```

- There is no need to declare the variable beforehand, or specify its data type. Bash determines the type of the variable based on its value.

```
$ x=hello
```

```
$ echo $x
hello
```

- http://www.computerhope.com/issues/ch001646.htm

Syntax - Control Structures

- Bash has the same control structures as the C programming language
- If:

```
$ if <condition1>
then
   ### code for condition1
elif <condition2>
then
   ### code for condition2
else
   ### code if the condition is not satisfied
fi
```

Syntax - Control Structures

- Bash has the same control structures as the C programming language
- For:

```
$ for arg in [list]
do
   ### code commands
done
```

- While:

```
$ while <condition>
do
   ### code commands
done
```

Syntax - Control Structures

Example: Which days are the weekend (run the script):

```
$ chmod +x days_example.sh
$ ./days_example.sh
$ cat days_example.sh
i=1
for day in Mon Tue Wed Thu Fri Sat Sun
do
 echo -n "Day $((i++)) : $day"
 if [ $i -eq 7 -o $i -eq 8 ]; then
   echo " (WEEKEND)"
   continue;
 fi
 echo " (weekday)"
done
```

Syntax - Control Structures

- Do it yourself: write a bash script that prints for each number between 1 and 1000 if it is divided by 2,3 and 5.

Sort

- The **sort** command arranges lines of text alphabetically or numerically.

```
$sort food
Afghani Cuisine
Bangkok Wok
Big Apple Deli
Isle of Java
Mandalay
Sushi and Sashimi
Sweet Tooth
$
```

Option	Description
-n	Sort numerically (example: 10 will be ranked after 2); ignore blanks and tabs.
-r	Reverse the order of sort.
-f	Sort upper and lowercase together.
-k	Specify a field to sort by. (sort –k 2 food will sort food by the second word)

כלים מרכזיים – מיד נראה אותם...

- awk שפה מאוד פשוטה וקלה לעבודה, בפרט מבחינת התחביר (syntax), ביחס ל bashעצמו. מאפשרת לבצע פעולות על קבצי נתונים טקסטואליים ביעילות רבה. היופי של השפה שהיא עובדת ברמת השורה הבודדת של קובץ הטקסט (אבל כמובן יש משתנים שמאפשרים לשמור נתונים משורה לשורה). עובדה זו מקלה מבחינת החשיבה ולכן השפה מאוד מתאימה לקבצים עם פורמט מובנה (כמו שהרבה פעמים קורה עם קבצי נתונים). יש לכם דוגמאות בשקפים ומומלץ ללמוד עוד. אפשר גם לכתוב awk בתוך script ולהריץ אותו לא רק מתוך שורת פקודה, אלא גם כתוכנית של ממש.
- sed – stream editor כלי לעריכת "זרם" (stream) של קלט. משמש לביצוע שינויי טקסט בסיסיים על הקלט כמו למשל החלפות של תווים/מילים.
- grep - global regular expression print הפונקציונליות העיקרית היא חיפוש של ביטוי רגולרי (regular expression) בתוך קבצי טקסט. דוגמאות לביטויים רגולריים אפשר למצוא בשפע ברשת.
- שימו לב שניתן לשלב את השימוש בכלים אלו באמצעות כתיבת bash script או באמצעות pipe.

קצת היסטוריה (מתוך Wikipedia)

awk היה אחד מהכלים הראשונים שהופיעו בגרסה 7 של מערכת ההפעלה יוניקס. הוא הוסיף תכונות חישוביות לשימוש במעטפת של יוניקס וזכה לפופולריות כיוון שמלבד Bourne Shell היה לשפת התסריט היחידה בסביבת UNIX הסטנדרטית. כיום awk מהווה רכיב חובה במערכות יוניקס ובכל הפצת לינוקס.

ל- awk קדמה תוכנה לעיבוד טקסט בשם sed, אשר נראתה לראשונה במערכות יוניקס בשנת 1974. כמו שפת awk, תוכננה sed לעבד טקסט שורה אחר שורה, ולשמש בעיקר לעיבוד טקסט מהיר בממשק שורת פקודה. awk המשיכה בקו דומה, אך היוותה שפה רחבה יותר. מאוחר יותר היא היוותה השראה לשפת Perl, אשר הייתה שפת תכנות מלאה, ואפשרה מספר פרדיגמת תכנות.

awk ראתה לראשונה אור בשנת 1977, אך תוקננה והורחבה משמעותית בשנים 1985-1988, כאשר יצאו nawk ("New AWK") ו-gawk ("GNU AWK"). במקביל פותחה Perl וראתה אור בשנת 1987, ובשנות ה-90 החלה להחליף את awk בנישת עיבוד הטקסט בסביבת יוניקס.

grep נכתבה על ידי קן תומפסון כיישום עצמאי מתוך מעבד הביטויים הרגולריים של עורך הטקסט ed, שהוא עצמו כתב. grep הופיעה לראשונה בתיעוד של UNIX בגרסה 4, בשנת 1973.

GREP, AWK AND SED

Grep

- The grep program searches a file or files for lines of a certain pattern. (http://www.computerhope.com/unix/ugrep.htm)

```
$ls -l | grep "Aug"
-rw-rw-rw-   1 john  doc     11008 Aug  6 14:10 ch02
-rw-rw-rw-   1 john  doc      8515 Aug  6 15:30 ch07
-rw-rw-r--   1 john  doc      2488 Aug 15 10:51 intro
-rw-rw-r--   1 carol doc      1605 Aug 23 07:35 macros
$
```

- Find all occurrences of Alice in Alice_book

  Try it out:

```
$grep Alice Alice_book
```

Grep

- Do it yourself: Find all occurrences of the term “Alice” in the first 10 lines of Alice_book

- Find all occurrences of the sequence “is” (even in upper-case and in a word) and the 3 lines after this line

Try it out:

```
$grep -A 3 -i is Alice_book
```

Awk

- Awk iterates over each line in the input file and enables to perform commands for each one. (http://www.computerhope.com/unix/uawk.htm)
- The basic structure:

```
$ awk ' {commands} ' filename
```

- Awk can also be used as a script.
- Example:

Try it out:

Prints the file Alice_book.

```
$ awk '{print}' Alice_book
```

Awk

- A richer structure:

```
awk 'BEGIN{ initializing code }{ commands that are performed for each line }END{commands to execute
after the file ended}'   file_name
```

- Example:

Try it out:

```
$ awk 'BEGIN{count=0;}{if (length($0)>70) count++; } END{ print count;}' Alice_book
```

Prints the number of lines in Alice_book that are longer then 70 characters.

Awk

- Examples of basic variables:

NF	number of fields in the current record
NR	ordinal number of the current record

- Examples of basic functions:

substr(*s*, *m*, *n*)	the *n*-character substring of *s* that begins at position *m;* 1 is the position.
tolower(*str*)	returns a copy of *str* with all upper-case characters transformed to their corresponding lower-case equivalents.

Awk – Basic Examples

- Print first 2 words in each line

```
awk '{print $1, $2}' Alice_book
```

- Do it yourself: Print out the number of lines in Alice_book

- Do it yourself: Average length of a line

- Print the average length of the 3rd word in each line.

```
awk 'BEGIN{count=0; sum=0;}{sum+=length($3); count++;}END{print sum/count;}' Alice_book
```

Awk – more examples

- Print each 200’th row in a book

```
$ awk '{if (NR%200==0) print;}' Alice_book
```

- **Do it yourself: Count how many 200’th lines exist.**
- Output each of these lines to a different file in the directory “files_d”

```
mkdir files_d

awk '{if (NR%200==0) print > "files_d/"NR;}' Alice_book
```

The number of lines will be the name of each file.

- For each file print the number of lines (combination of awk and bash commands)

```
for f in $(ls files_d/*); do echo -n $f" "; awk 'END{print NR;}' $f; done
```

Awk – Associative Array

- You can also use associative arrays in awk (key-value pairs).
  - The key can be of any type, not only numeric.
- Find the rows that start with the word Alice:
  - Create an array that maps a line number to the first word in this line (using an associative array).
  - Print the key if the value equals to ‘Alice’’

```
$ awk '{a[NR]=$1} END {for (k in a) if(a[k]=="Alice") print k;}' Alice_book
```

an associative array

Awk – Scripts

- You can also run awk as a script.
- An Awk script is basically an awk program that is too long or complex to be used in a single line.
- Example: (split each row by ‘,’ instead of ‘ ‘ and if the 1st part is different than the 3rd, print the 2nd part)
  - Command line inliner:

```
$ awk '-F,' '{if ($1!=$3) print $2; $4=$1; print;}END{print "number of lines: "NR;}' Alice_book
```

Awk – Scripts

- As an awk script:

```
$ chmod +x simple_script.awk
$ awk -f simple_script.awk file
$ cat simple_script.awk
BEGIN{
     FS=“,"
}
{ if ($1!=$3) print $2; $4=$1; print;}
END{print "number of lines: "NR;}
```

Awk – run script and arrays

- Do it yourself:
  - Transform this code to a script and run it:

```
$ awk '{a[NR]=$1;} END {for (k in a) if(a[k]=="Alice") print k;}' Alice_book
```

- See some more examples: https://www.tutorialspoint.com/awk/awk_basic_examples.htm

Sed

- sed, short for "stream editor", allows you to filter and transform text.

(http://www.computerhope.com/unix/used.htm)

```
sed OPTIONS... [SCRIPT] [INPUTFILE...]
```

Example: Creating a Hedva in Wonderland version:

```
sed 's/Alice/Hedva/' Alice_book > Hedva_book
```