Israeltech challenge

Linux

Making magic with the command-line

Goals

- Make sure you’re comfortable with Linux
- Whet your appetite with the magic shell can do
- Not a comprehensive Linux course, Will not touch:
  - Basic stuff – you’re not in pre-course anymore..
  - Admin stuff
  - Installations
  - Configuration
  - Multi-user environment
  - Working on a cluster
  - ...
- How to learn?

Quick Linux Intro

- What is Linux
  - GNU/Linux, to be exact
  - Open-source operating system, offspring of UNIX
  - Linux is the OS kernel, GNU is a set of tools to use it
    - Much like an engine needs a the rest of the car around it

- Linux has distributions (200+), each with it’s own:
  - Look & feel
  - Default software and configuration
  - **Community & Agenda**

Quick Shell Intro

- What is the Shell?
  - Command-line interface between you and the OS
  - Your way to tell the OS what you want it to do
  - Many alternatives, most commands remain the same
    - Bash is the most common
    - Flame-war alert!

- Linux is useful for:
  - Servers / services (mail, storage, networking)
  - Development (not only the OS is open source…)
  - Making the most out of your system (not just regular PC)
  - Handling data (analysis, computation and more!)

Shell – FAQ

- I asked the shell for X – it refused. What to do?
  - Read the error
  - Make sure you have permissions (use su/sudo).
  - Ask for help (“X -h”, or “man X” for the manual).
  - Man – opens the manual in ‘more’, in it:
    - page down: space
    - page up: ‘b’
    - search PATTERN: ‘/PATTERN’
    - quit: ‘q’
  - Stack Overflow..

May I?

- Permissions
  - structure: T UUU GGG OOO
  - chmod, chgrp

- Why bother with permissions? I can just use root.
  - You *can, but you shouldn’t.
  - Consider the difference between the following two:

    “rm –rf /mnt” and “rm –rf / mnt”
  - Only use root (via sudo) when you need it (and CAREFULLY!)
  - Only on your PC - most corporate users have no access

Shell – prompt, the almighty

- Prompt: line ending with a blinking cursor thingy.
  - - And it’s waiting for your input!

  - Each command is executed inside the current directory.
  - The prompt “hangs” until the execution is complete.
    - Use *ctrl-c* to stop execution before it completes.
    - Use *ctrl-z* to pause the execution.
      - Use *bg* to resume execution in the background
      - Use *fg* to restore execution (after *bg* or *ctrl-z*)
    - Use *jobs* to see running jobs
    - Kill %1
    - Monitoring: top (memory, swap)

History repeats

- You'll end up typing (and re-typing) commands, a few things can help:
  - Use the Tab button to auto-complete commands you type.
  - Use Up/Down arrows to scroll through past commands.
  - CTRL-r to search previous commands by prefix.
  - history
  - CTRL-a moves the cursor to beginning of the line
  - CTRL-e moves the cursor to get to the end

You are a unique snowflake

IsraelTech challenge

- Configure your workspace
  - .bashrc
  - source
- Environemnt variables
  - Use *env* to show them all
  - *echo $PATH* – shows list of executable directories
  - *rehash*

- Alias defines short-hand commands:
  - alias ls=“ls –alh”
  - *alias hungry=“mail –s ‘Let’s have lunch, NOW.’ fellows”*

Processes

- Processes are programs currently running
  - Listing them: *ps*
  - Watching a live list of them: *top*
  - Killing them: *kill 1234*
  - Killing without mercy: *kill -9 1234*

- A word about *signals:*
  - Originally designed for RPC
  - Now mostly used for all sorts of “interruptions”
  - Slow, but powerful.
    - None shall escape the powerful SIGKILL.

Shell – Pipes and redirection

IsraelTech challenge

- Redirection causes a file to be used for I/O
  - Writing output: *ls > file_list.txt*
  - Reading input: *cat < a.txt* (yes, equivalent to *cat a.txt*)
  - Concatenating: echo hey >> status_update.txt
- Pipes connect two (or more) processes
  - *ps* | *sort* – prints a sorted list or processes
- Trick question:
  - What would "*ps > sort*" do?

Shell – Standard interaction

- Each process is born with 3 descriptors:
  - 0 – Standard Input
  - 1 – Standard Output
  - 2 – Standard Error
- Each process also has the command-line arguments, <u>which is not the same as input</u>
- Descriptors can be referenced for specific redirect:
  - find . 1>output.txt 2>errors.txt
  - find . >all_output.txt 2>&1

Shell – Useful tools

- xargs converts from input (FD #0) to command-line parameters:
  - *find / -name "*.py" | rm –f* - This will not work
  - *find / -name "*.py" | xargs rm –f* - This might work
  - *find / -name "*.py" | xargs –n 1 rm –f* - This will work
- Which
- FOR Loops

```
for i in $( ls ); do
    echo item: $i
done
```

Shell – Wildcards and misc.

- Use wildcards to describe multiple files:
  - **.txt*
  - *a*c.txt* VS. *a?c.txt*

- Special directories:
  - '.' points to the current directory (not very useful)
  - '..' points to the parent directory (useful for traversal)
  - '~' points to the user home directory (for per-user scripts)
  - '-' points to the last directory you've been too
  - Use *pwd* to see the full path of your current directory

Time for some magic!

Israeltech challenge

- Day-to-day analogies:
  - *fridge* (browses fridge)
  - *fridge | grep apple* (grap an apple)
  - *fridge | sort –time | grep apple* (grab a fresh apple)
  - *fridge | sed s/juice/popsicle (decrease temperature…)*

Building blocks for magic

IsraelTech challenge

- Selection
  - cat – All the input (-n adds line numbers)
  - *head* – First lines of the input
  - *tail* – Last lines of the input (+2 all lines but the first)
  - *wc* – Counts the lines/words/characters.
  - *cut* – Parts each line of input (-f, -c)
  - *grep* – Lines of input matching a criteria.
- Rearangement
  - *sort* – Sort input by some criteria (-k 2nr,2)
  - *uniq* – Remove duplicate lines (uniq –c also adds line counts)
  - *fold* – Wrap input lines into a fixed width.

Text manipulation commands

IsraelTech challenge

- *tr* – “mono-alphabetic” replace (char-by-char)
  - cat old.txt | tr ‘a’ ‘b’ > new.txt
  - Can remove char: cat windows.txt | tr –d ‘\r’ > linux.txt

- *sed* – replace phrases (regular expressions)
  - *cat old.txt | sed ‘s/abc/def/g’ > new.txt*
  - template example

- *awk* – fully functional command-line scripting
  - *ps | awk ‘{ print $1 $5; }’*
  - *BEGIN, END*
  - *NR, NF…*

Stream editor sorcery

Israeltech challenge

- Regular expression playground

  - echo ab12 | sed ‘s/\([a-z]*\).*/\1/’ => Output: ab
  - sed ‘s/\([a-z]*\) \([a-z]*\)/\2 \1/‘ => Replaces order
  - sed ‘s/\([a-z]*\) \1/\1/‘ => Removes duplicates
  - sed ‘s/[^]*/(&)/’ <old >new => Parenthesize first word
  - sed 's/[^][^]*/(&)/g' <old >new => Parenthesize all words
  - sed 's/[a-zA-Z]* //2' <old >new => Remove second word
  - sed -n 's/a/A/2pw /tmp/new2' <old >new => replace the second a with A and output to both new and new2
  - sed -n '/PATTERN/p' file => Fancy “grep PATTERN”

Text processing wizardry

- Text parsing and logic
  - ps | awk '{print $1,$NF;}‘ => Print only PID and name
  - ps | awk ‘$1 >100’ => Lines with PIDs above 100
  - ps | awk ‘$NF ~ /b.*/’ => Processes staring with ‘b’
  - ps | awk 'BEGIN { count=0;}

    $NF ~/b.*/ { count++; }

    END { print "Number of procs =",count;}‘
  - awk -F ':' '$3 > maxuid { maxuid=$3; maxline=$0 }; END { print maxuid, maxline }' /etc/passwd => Max UID user

How to approach a data file?

- *file* – Guesses the type of file you have. Common:
  - ASCII text file
  - Gzip compressed file
- If it’s compressed/archived – extract the original:
  - *gunzip compressed_data.gz*
  - *tar –xvf archived_data.tar*
  - *unzip compressed_and_archived_data.zip*
- Take a peek:
  - *head massive_file.txt*
  - *zcat compressed_file.gz*
  - *less plaintext_file.txt*
  - *count stuff…*

Shell scripting – useful basics

- Variables & IF conditionals

```
#!/bin/bash
 if [ -z "$1" ]; then
     echo usage: $0 directory
     exit
 else
      echo User arguments OK
 fi
 SRCD=$1
 TGTD="/var/backups/"
 OF=home-$(date +%Y%m%d).tgz
 tar -cZf $TGTD$OF $SRCD
```

Shell scripting – useful basics

Israeltech challenge

- FOR Loops

```
for i in $( ls ); do
    echo item: $i
done
```

- WHILE Loops

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

Strength of Linux

- Open-source
- Maximal user flexibility
- Batch processing
- Holy Pipe or the Tower of Babylon

QUESTIONS

Let's play

- Personal exercises
- Demo
  - Many ways to do the same thing
  - Quick and dirty mode