מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

# Real-Time Group

## Linux Administration

רח` רוזנסקי 14 ראשל"צ    טל. 7067057 -077

# Contents

# Contents

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

# Contents

## Task control

- Full control on tasks
- Executing in background, suspending, resuming and aborting
- List of active tasks
- Killing processes
- Environment variables
- PATH environment variables
- Shell aliases, .bashrc file

## Miscellaneous

- Text editors (vi \ eclipse \ kdevelop )
- Compressing and archiving
- Comparing files and directories
- Looking for files
- Getting information about users

## Linux overview

## Why Linux?

- It’s free!
- Open Source (modifiability, extensibility, …)
- Portable code (a well defined layers of related and unrelated HW source code)
- Works on all major architectures
- Robustness (after several revisions, countless people working on it)
- Widespread Usage
- Compatibility with several other exotic platforms.

# Linux Features

- Monolithic kernel (but well-defined interfaces)
- Multi-tasking
- Multi-user capability
- Multi-processing Support (since 2.0, several processors at once)
- Architecture Independence (PCs, Alpha, Sparc,…)
- Shared Libraries (dll)
- Support for Posix standard
- Several Executables formats
- Several File Systems (ext2\ext3 …)
- Several network protocols

# Installing Linux and more ..

## Linux installation

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

1. Make sure you have enough free space (30 GB) on your local hard-drive (unallocated partition) or better yet get a new storage unit (HD or USB-disk on key).
2. Get the image from the distribution of your favor

   CentOS: http://CentOS.org Ubuntu: http://ubuntu.com Fedora: http://fedoraproject.org

   - For CentOS there are two images:
     - A live cd CentOS-5.4-i386-LiveCD.iso (Loads the OS from the disk without an installation)
     - For a complete CentOS DVD installation download CentOS-5.4-i386-bin-DVD.iso
   - For Ubuntu / Fedora, both are present on the same cd Image (download the last version)
3. Burn the ISO image on a DVD Disk (you can also burn the ISO installation disk to a DiskOnKey).

## Linux installation

4. Start the P.C. from the DVD (configure the Bios boot sequence accordingly)
5. An application (such as Anakonda) will automatically start installing linux on your hard-drive, choose the option “create custom layout”
6. Resize your partitions (shrink them), keep around at least 20-30 GB for the installaion, after shrinking you will have 20-30 GB not in use.
7. Create two new partitions: ext3/4 and a swap area. Use most of it for the ext partition, keep about 1-2GB for the swap.
8. Install the BootLoader, on the hard drive itself (not on the partition created)

# Linux installation

Common pitfalls installing linux:

Before you start the installation, it is recommended to back-up.

•Managing partitions:

- Non-standard disk layout (microsoft advanced partitioning / gpt layout) Check it - right click on my computer -> management -> disk management
- Standard layout, but cannot add a partition. It can happen if you have 4 primary partitions. Fix it to have 3 primary + 1 extended partition (logical partitions)
- The installation disk cannot shrink / resize the partitions. It is not supported in all linux installations. Very fixable using a third party applications.

Use **partition magic** or any other partition manager application, and you're ok.

Use **hiren boot cd** rescue disk to help in backing up / rescue in case of a failure.

## Linux installation

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

One more common pitfall - secure boot bios (usually comes with windows 8 ready laptops). You need a special installation (for example ubuntu rimix version) to install linux on your system.

Misc situations:

- Laptop without a cd/dvd – use usb installation (tell me now and I can solve it for you)
- Efi partitions / failure after boot – let me know, we may fix it on the spot.
- Encrypted / protected HD, or you prefer not to touch the layout – installation can be done on a disk-on-key, and no layout is changed.
- Your computer is at home and you don’t want to do it yourself – bring it to us and we will install it for you.
- Don’t want to install yet / at all ? Use virtual machine (vmware / virtual box) and your computer is guaranteed to be safe ..

## Linux Update/Install

open a terminal and type the commands:

- yum update (yum is explained in detail at the end of the book)
- yum install kernel-devel
- yum install kdevelop
- yum install gcc
- yum install gdb

**Note**

For Ubuntu use almost the same:

```
sudo apt-get install … / sudo apt-get update …
```

Install the package aptitude, and you’ll be able to search using:

```
sudo aptitude search [package]
```

# Linux Components And packages

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

In some cases you need to install an application that is unfamiliar to the distribution's repository (each community of linux has a database containing a list of installable packages called repository).

You can (and sometimes should) install it manually.

There are two major versions you'll find: **rpm** packages, and **debian** packages:

Installing an rpm package: rpm –ivh <package file path / url>

e.g:

```
rpm -ivh ftp://ftp.redhat.com/pub/redhat/rh-2.0-beta/RPMS/foobar-1.0-1.i386.rpm
rpm -ivh teamviewer_linux.rpm
```

Installing a debian package: dpkg –i <package file path>

e.g:

```
dpkg -i teamviewer_linux.deb
```

## Linux File-System

# Everything is a file

Almost everything in Unix is a file!

| | |
|---|---|
| **-** | Regular files |
| **d** | Directories (Directories are just files listing a set of files) |
| **l** | Symbolic links ( Files referring to the name of another file) |
| **c , b** | Devices and peripherals (Read and write from devices as with regular files) |
| **p** | Pipes (Used to cascade programs => cat *.log \| grep error) |
| **s** | Sockets (Inter process communication) |

# File names

File name features since the beginning of Unix

- Case sensitive
- No obvious length limit
- Can contain any character (including whitespace, except /).
- File name extensions not needed and not interpreted by linux.

Just used for user convenience. Applications may require an extension.

File name examples: `README` `.bashrc` `Windows Buglist` `index.html` `index.htm` `index.html.old`

File name with special characters (e.g spaces or uncommon symbols) can be referred with a '\' before the symbol or have a quotation (").

# File Content

Since the name doesn’t indicate its type, use the command file to identify what sort of file is it.

- file MyNotes.txt
- file helloworld.c
- file helloworld

For showing the content of a text file use: cat MyNotes.txt

If the file is an executable, you can display it’s content (usually lables and segmentations) through the following commands.

- nm helloworld
- readelf -a hellowrold
- xxd helloworld

Text files has several options, which will be covered shortly

## File paths

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

**path** is a sequence of nested directories with a file or directory at the end separated by the / character.

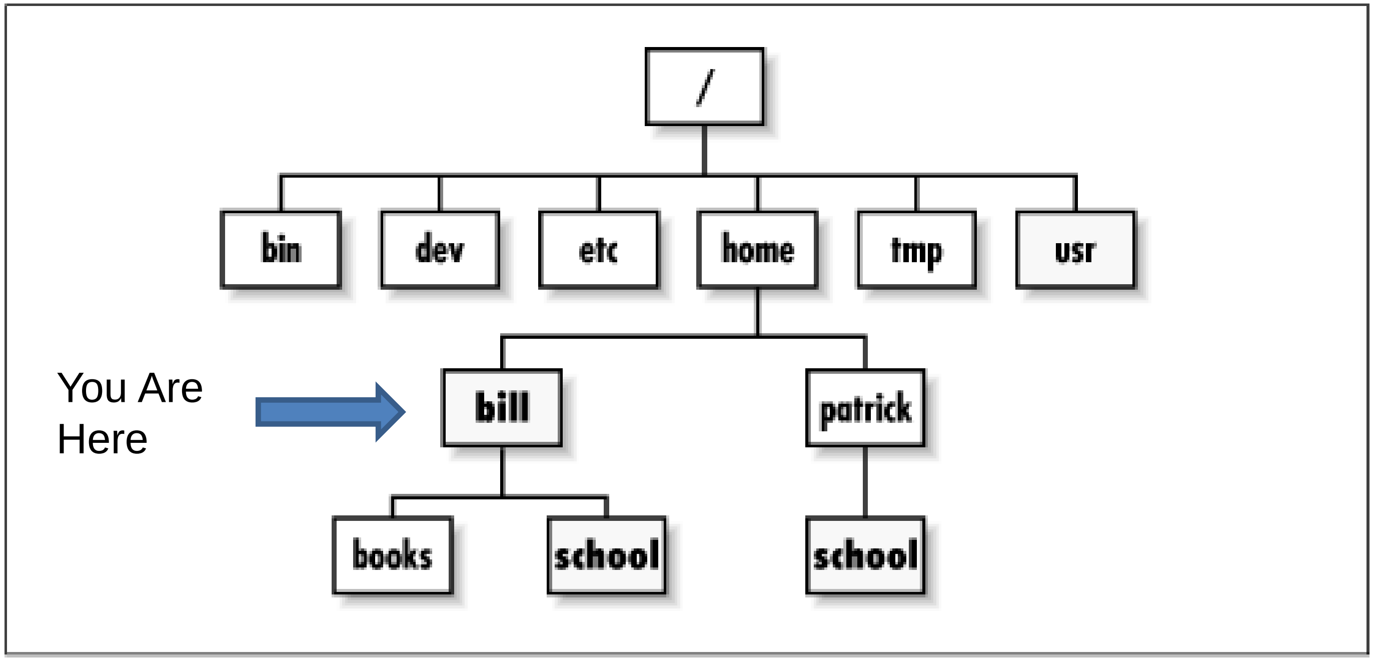
• Relative path:  Relative to the current directory.
Example:  ./documents/fun/microsoft_jokes.html

• Absolute path: Starting from the Root directory.
Example: /home/bill/bugs/crash9402031614568

• / : root directory: Start of absolute paths for all files on the system (even for files on removable devices or network shared).

# File paths (2)

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק



**Lets illustrate Absolute path and Relative path.**

**-You wish to refer to a file named XYZ.txt in the school directory that is found in patrick folder**
**-You wish to refer to a file named LinuxAdmin.zip in the books folder under bills**
**-You wish to refer to the folder school in patrick folder**

# Linux file-system structure (1)

Not imposed by the system. Can vary from one system to the other, even between two GNU/Linux installations!

| | |
|---|---|
| / | Root directory |
| /bin/ | Basic, essential system commands |
| /boot/ | Kernel images, initrd and configuration files |
| /dev/ | Files representing devices |
| /dev/hda: | first IDE hard disk |
| /etc/ | System configuration files |
| /home/ | User directories |
| /lib/ | Basic system shared libraries |

# Linux file-system structure (2)

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

| | |
|---|---|
| /lost+found | Corrupt files the system tried to recover |
| /media | Mount points for removable media (/media/usbdisk, /media/cdrom) |
| /mnt/ | Mount points for temporarily mounted filesystems |
| /opt/ | Specific tools installed by the sysadmin |
| /usr/local/ | often used instead |
| /proc/ | Access to system information (/proc/cpuinfo, /proc/version ...) |
| /root/ | root user home directory |
| /sbin/ | Administrator only commands |
| /sys/ | System and device controls (cpu frequency, device power, etc.) |

## Linux file-system structure (3)

| | |
|---|---|
| /tmp/ | Temporary files |
| /usr/ | Regular user tools (not essential to the system) /usr/bin/, /usr/lib/, /usr/sbin... |
| /usr/local/ | Specific software installed by the sys-admin (often preferred to /opt/) |
| /var/ | Data used by the system or system servers /var/log/, /var/spool/mail (incoming mail), /var/spool/lpd (print jobs)... |

The Unix file-system structure is defined by the File-system Hierarchy Standard (FHS):

http://www.pathname.com/fhs/

## Text editors

## Text editors

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

- Graphical text editors
  - gedit
  - Emacs, Xemacs
  - kdevelop (compiler enviroment)
  - eclipse (compiler enviroment)
- Text-only text editors
- Often needed for sysadmins and great for power users
  - vi, vim
  - nano

# The gedit text editor

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

http://www.gedit.org/

Best text editor for non vi or emacs experts

Feature highlights:

- Very easy text selection and moving
- Syntax highlighting for most languages and formats. Can be
- tailored for your own log files, to highlight particular errors and warnings.
- Easy to customize through menus
- Not installed by default by all distributions

# Emacs / Xemacs

Emacs and Xemacs are pretty similar (up to your preference)

- Extremely powerful text editor features
- Great for power users
- Less ergonomic than nedit
- Non standard shortcuts
- Much more than a text editor (games, email,shell, browser).
- Some power commands have to be learnt.

## vi

Text –mode text editor available in all Unix systems. Created before computers with mice appeared.

- Difficult to learn for beginners used to graphical text editors.
- Very productive for power users.
- usually used for editing files in system administration or in Embedded Systems, when you just have a text console.

Though vi is extremely powerful, its main 30 commands are easy to learn and are sufficient for 99% of everyone's needs!

# Vim- Vi improved

vi implementation now found in most GNU / Linux host systems

- Implements lots of features available in modern editors:

  - syntax highlighting,
  - command history,
  - help,
  - unlimited undo
  - much much more.

Cool feature example: can directly open compressed text files.

Comes with a GTK graphical interface (gvim)

Unfortunately, not free software (because of a small restriction in freedom to make changes)

## Miscellaneous
## Various shell commands

**Getting started in shell,**

**Start openning the terminal from now on ..**

## wget examples

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

wget -c\
http://microsoft.com/customers/dogs/winxp4dogs.zip
Continues an interrupted download.

wget –m http://lwn.net/
Mirrors a site.

wget –r –np http://www.xml.com/ldd/chapter/book/
Recursively downloads an online book for offline access.

-np: "noparent". Only follows links in the current directory.

# The wget command

Instead of downloading files from your browser, just copy and paste their URL and download them with wget!

wget main features

- http and ftp support
- Can resume interrupted downloads
- Can download entire sites or at least check for bad links
- Very useful in scripts or when no graphics are available (system administration, embedded systems)
- Proxy support (http_proxy and ftp_proxy env. variables)

# More stuff

sleep 60
Waits for 60 seconds (doesn't consume system resources).

wc report.txt (word count)
438 2115 18302 report.txt
Counts the number of lines, words and characters in a file or in standard input.

bc ("basic calculator?")
bc is a handy but full featured calculator. Even includes a programming language! Use the l option to have floating point support.

date
Returns the current date. Useful in scripts to record when commands started or completed.

# Shells and File Handling

# Command line interpreters

**Shells**: tools to execute user commands.

Called “shells” because they hide the details on the underlying operating system under the shell's surface.

Commands are input in a text terminal, either a window in a graphical environment or a text-only console.

Results are also displayed on the terminal. No graphics are needed at all.

Shells can be scripted: provide all the resources to write complex programs (variable, conditionals, iterations...)

# Well known shells

Most famous and popular shells

**sh**: The Bourne shell (obsolete)

Traditional, basic shell found on Unix systems, by Steve Bourne.

**csh:** The C shell (obsolete)

Once popular shell with a Clike syntax

**tcsh**: The TC shell (still very popular)

A C shell compatible implementation with evolved features (command completion, history editing and more...)

**bash**: The Bourne Again shell (most popular)

An improved implementation of sh with lots of added features too.

**Fishshell**: The Friendly Interactive Shell - http://www.fishshell.org/

# ls command

Lists the files in the current directory, in alphanumeric order, except files starting with the "." character.

ls -a (all) Lists all the files (including .* files)

ls -l (long) Long listing (type, date, size, owner, permissions)

ls -t (time) Lists the most recent files first

ls -s (size) Lists the biggest files first

ls -r (reverse) Reverses the sort order

ls -ltr (options can be combined) Long listing, most recent files at the end

Note: files starting with the "." character are hidden files in Linux

## File name pattern substitutions

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

Better introduced by examples!

ls *txt

The shell first replaces *txt by all the file and directory names ending by txt (including .txt), except those starting with “.” , and then executes the ls command line.

ls  d.*

Lists all the files and directories starting with . d

ls -d

tells ls not to display the contents of directories.

cat ?.log

Displays all the files which names start by 1 character and end by .log

find  . -name “*.pdf ”

Search for a file that ends by .pdf

# Special directories

./ - The current directory. Useful for commands taking a directory argument. Also sometimes useful to run commands in the current directory (see later), ./readme.txt and readme.txt are equivalent.

../ - The parent (enclosing) directory. Always belongs to the . Directory (see ls a). Only reference to the parent directory.

Typical usage: cd ..

~/ - Shells just substitute it by the home directory of the current user. Cannot be used in most programs, as it is not a real directory.

for example ~sydney/ is substituted by shells by the home directory of the sydney user.

# The cd and pwd commands

| | |
|---|---|
| cd <dir> | - Changes the current directory to <dir>. |
| cd .. | - Go up to parent directory. |
| cd - | - Gets back to the previous current directory. |
| cd / | - Goes to the root directory |
| pwd | - Displays the current directory ("working directory"). |

**Special Note:**

If you want more options to each command, or you don't know, ask Linux! itself.

Use one of the following:

```
[command] --help
info [command]
man [command]
help [built-in shell command]    # for built-in commands only (see man bash)
```

## The cp command

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

cp <source_file> <target_file>
Copies the source file to the target.

cp file1 file2 file3 ... dir
Copies the files to the target directory (last argument).

cp  -i
interactive, Asks for user confirmation if the target file already exists

cp –r <source_dir> <target_dir> (recursive)
Copies the whole directory.

# mv and rm commands

mv <old_name> <new_name> (move)

Moves \ renames the given file or directory.

mv –i (interactive)

If the new file already exits, asks for user confirm

rm file1 file2 file3 ... (remove)

Removes the given files.

rm  -i (interactive)

Always ask for user confirm.

**rm –r dir1 dir2 dir3** (recursive)

Removes the given directories with all their contents.

# Creating and removing directories

mkdir dir1 dir2 dir3 ... (make dir)

Creates directories with the given names.

rmdir dir1 dir2 dir3 ... (remove dir)

Removes the given directories

Safety: only works when directories are empty.

Alternative: rm -r (doesn't care if they are empty directories or not).

# Concatenating \ Displaying files

Several ways of displaying the contents of files.

cat file1 file2 file3 ... (concatenate)

Concatenates and outputs the contents of the given files.

more file1 file2 file3 ...

After each page, asks the user to hit a key to continue.

Can also jump to the first occurrence of a keyword (/ command).

less file1 file2 file3 ...

Does more than more with less (what??? ☺).

Doesn't read the whole file before starting.

Supports backward movement in the file.

## The head and tail commands

head [<n>] <file>

Displays the first <n> lines (or 10 by default) of the given file.
Doesn't have to open the whole file to do this!

tail [<n>] <file>

Displays the last <n> lines (or 10 by default) of the given file.
No need to load the whole file in RAM! Very useful for huge files.

tail -f <file> (followtxt)

Displays the last 10 lines of the given file and continues to display new lines when they are appended to the file.

Very useful to follow the changes in a log file, for example.

Examples: head windows_bugs.txt

tail -f ben.txt

# The grep command

grep <pattern> <files>

Scans the given files and displays the lines which match the given pattern.

grep error *.log

Displays all the lines containing error in the *.log files

grep –i  error *.log

Same, but case insensitive

grep –ri error

Same, but recursively in all the files and subdirectories

grep –v   info *.log

Outputs all the lines in the files except those containing info.

# The sort command

sort <file>

Sorts the lines in the given file in character order and outputs them.

sort –r <file>

Same, but in reverse order.

sort –ru <file>

u: unique. Same, but just outputs identical lines once.

## Standard output

All the commands outputting text on your terminal do it by writing to their **standard output**.

Standard output can be written (redirected) to a file using the > symbol

Standard output can be appended to an existing file using the >> symbol

# Standard I/O, Redirections, pipes

# Standard output examples

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

Copy a Directory’s file names to a file .

```
ls  directory/* > file_name.txt
```

Copy a file’s content to another file (over-writing) .

```
cat obiwan_kenobi.txt > starwars_biographies.txt
```

Append a file’s content to another file (not over-writing) .

```
cat han_solo.txt >> starwars_biographies.txt
```

Creating a new file without a text editor.

```
echo “README: No such file or directory” > README
```

## Standard input

Lots of commands, when not given input arguments, can take their input from standard input.

sort takes its input from the standard Input: in this case, what you type in the terminal (ended by [Ctrl][D])

```
Sort
    (input text) zzzz
                 aaaaa
    [Ctrl][D] (to end)
```

Arranges input text in alphabetic order

```
        aaaaa
        zzzz
```

sort < participants.txt -The standard input of sort is taken from the given file.

# Pipes

Unix pipes are very useful to redirect the standard output of a command to the standard input of another one.

Examples

```
cat *.log | grep –i error | sort
grep  –ri  error . | grep -v  “ignored” | sort u > serious_errors.log
```

```
cat /home/*.homework.txt | grep mark
```

This one of the most powerful features in Unix shells!

# Standard error

Error messages are usually output (if the program is well written) to standard error instead of standard output.

Standard error can be redirected through 2> or 2>>

Example: cat f1 f2 nofile > newfile 2> errfile:

Note: 0 is the descriptor for standard input

1 is the descriptor for standard output, (1> is equivalent to >)

2 is the descriptor for standard error

You can redirect both standard output and standard error to the same file using &> :

cat f1 f2 nofile &> wholefile.

# The yes command

Useful to fill standard input with always the same string.

yes <string> | <command>

Keeps filling the standard input of <command> with <string>
(y by default).

Examples

yes | rm -r dir /

yes “ " | make oldconfig

(equivalent to hitting [Enter] to accept all default settings)

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

# Symbolic Links,

# File Permissions

# And Special Files

# Symbolic links

A symbolic link is a special file which is just a reference to the name of another one (file or directory),

Useful to reduce disk usage and complexity when 2 files have the same content.

<u>Example:</u>

anakin_skywalker_biography > darth_vador_biography

How to identify symbolic links:

ls –l displays “=> “ and the linked file name.

GNU ls displays links with a different color.

# Creating soft symbolic links

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

To create a soft symbolic link (same order as in cp):

```
ln  -s  file_name  link_name
```

To create a link  to a file in another directory, with the same name:

```
ln  -s  ../README.txt
```

To create multiple links at once in a given directory:

```
ln  -s  file1 file2 file3 ... dir
```

To remove a link:

```
rm link_name
```

Of course, this doesn't remove the linked file!

.

# Hard links

The default behavior for ln is to create hard links

```
ln  file_name  hard_link_name
```

A hard link to a file is a regular file with exactly the same physical contents (two files whom share the same inode).

While they still save space, hard links can't be distinguished from the original files.

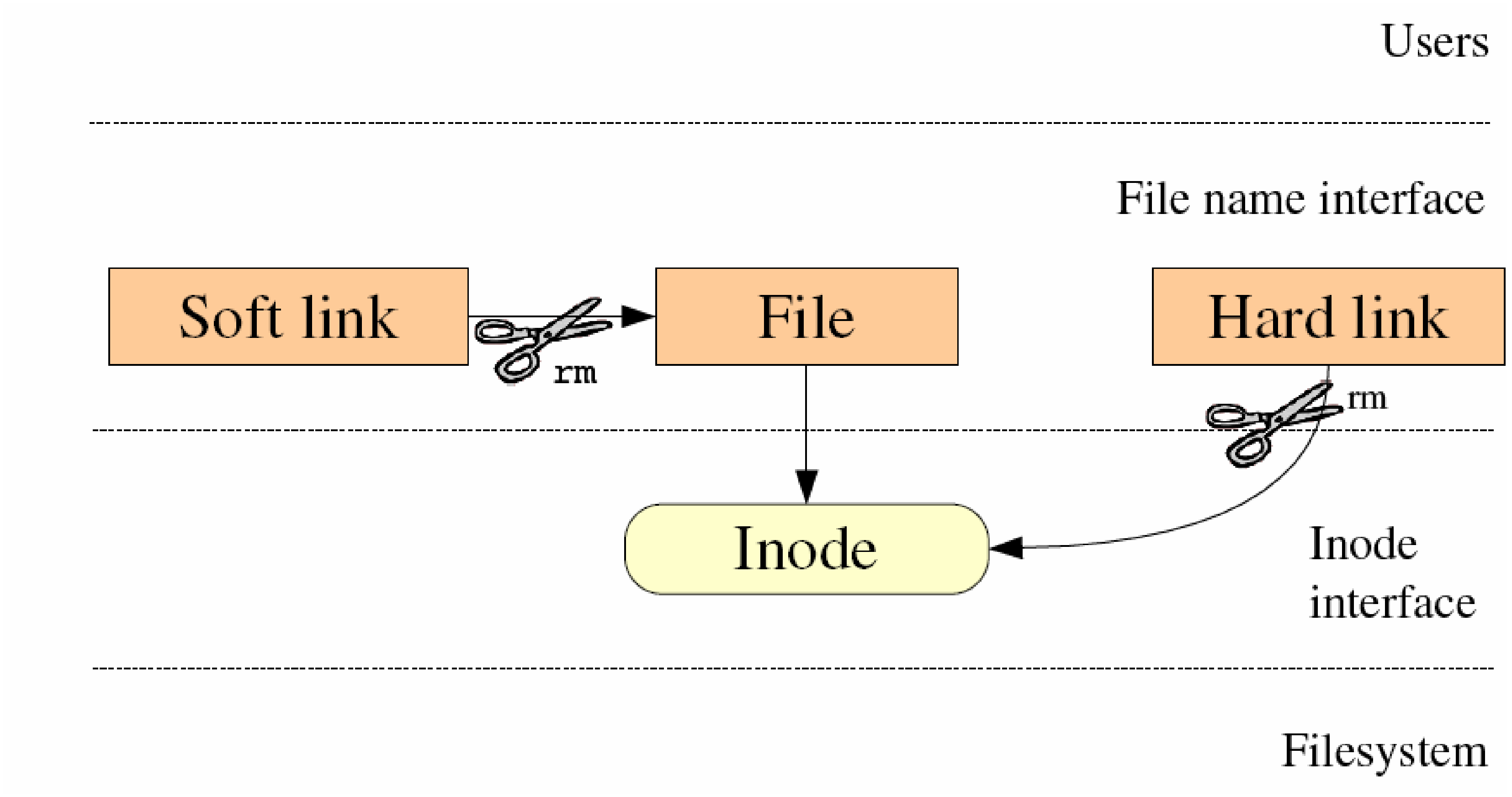
If you remove the original file, there is no impact on the hard link contents.

The contents are removed when there are no more files (hard links) to them.

# Files names and inodes

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

Makes hard and symbolic (soft) links easier to understand!



Inode – struct representing File / Directory

# File access rights

Use ls –l to check file access rights.

3 types of access rights

- Read access (r)
- Write access (w)
- Execute rights (x)

3 types of access levels

- User (u): for the owner of the file
- Group (g): each file also has a “group” attribute, corresponding to a given list of users
- Others (o): for all other users

# Access right constraints

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

- x without r is legal but is useless – In most cases you have to be able to read a file in order to execute it.
- Both r and x permissions needed for directories: x - to enter, r - to list its contents.
- **You can't rename, remove, copy files** in a directory if you don't have **w access** to this directory.
- If you have w access to a directory, you can remove a file even if you don't have write access to this file (remember that a directory is just a file describing a list of files). This even lets you modify (remove + recreate) a file even without w access to it.

## Access rights examples

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

-rw--r--r--

Readable and writable for file owner, only readable for others

-rw--r-----

Readable and writable for file owner, only readable for users  belonging to the file group.

drwx------

Directory only accessible  by its owner

-------r-x

File executable by others but neither by your friends nor by yourself. Nice protections for a trap...

# chmod: changing permissions

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

chmod <permissions> <files>

2 formats for permissions:

**Octal format (abc):**

r*4+w*2+x (r, w, x: booleans)

Example: chmod 644 <file>

(rw for u, r for g and o)

| **Permission** | Read | Write | eXecute |
|---|---|---|---|
| Letter | r | w | x |
| Octal value | 4 | 2 | 1 |

**symbolic format.**

chmod go+r: add read permissions to group and others.

chmod u-w: remove write permissions from user.

chmod a-x: (a: all) remove execute permission from all.

| **who** | **Act** | **Perm-ission** |
|---|---|---|
| u g o a | + - = | r w x |

# Changing users

You do not have to log out to log on another user account!

su ben
(Rare) Change to the ben account, but keeping the environment variable settings of the original user.

su - david
(More frequent) Log on the david account, with exactly the same settings as this new user.

su –
When no argument is given, it means the root user.

sudo  some_task – do the task with admin privileges (the task needs you to be root)

# Getting information about users

who
Lists all the users logged on the system.

whoami
Tells what user I am logged as.

groups
Tells which groups I belong to.

groups <user>
Tells which groups <user> belongs to.

finger <user>
Tells more details (real name, etc) about <user>
Disabled in some systems (security reasons).

# More chmod

-R: apply changes recursively

-X: but only for directories and files already executable Very useful to open recursive access to directories, without adding execution rights to all files.

t: (sticky). Special permission for directories, allowing only the directory/file owner to delete a file in a directory.

```
chmod a+t /tmp
```

Useful for directories with write access to anyone, like /tmp.

Displayed by ls –l with a t character.

Example makes directory linux and everything in it available to everyone!

```
chmod -R a+rX linux/
```

# Special devices (1)

Device files with a special behavior or contents

**/dev/null** - The data sink! Discards all data written to this file. Useful to get rid of unwanted output, typically log information:

```
mplayer black_adder_4th.avi &> /dev/null
```

**/dev/zero** - Reads from this file always return \0 characters

Useful to create a file filled with zeros:

```
dd if=/dev/zero of=disk.img bs=1k count=2048
```

See man null or man zero for details

# Special devices (2)

**/dev/random -** Returns random bytes when read. Mainly used by cryptographic programs. Uses interrupts from some device drivers as sources of true randomness (“entropy”).

Reads can be blocked until enough entropy is gathered.

**/dev/urandom -** For programs for which pseudo random numbers are fine.

Always generates random bytes, even if not enough entropy is available (in which case it is possible, though still difficult, to predict future byte sequences from past ones).

See man random for details.

# Special devices (3)

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

**/dev/full** - Mimics a full device (no more space).
Useful to check that your application properly handles this kind of situation.

See man full for details.

## Task Control

## Full control on tasks

Since the beginning, Unix supports true preemptive multitasking.

- Ability to run many tasks in parallel, and abort them even if they corrupt their own state and data.
- Ability to choose which programs you run.
- Ability to choose which input your programs takes, and where their output goes.

# Processes

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

"Everything in Unix is a file, Everything in Unix that is not a file is a process"

**Processes**

- Instances of a running programs
- Several instances of the same program can run at the same time

Data associated to processes: open files,
allocated memory,
stack,
process id,
parent,
priority,
state ...

## Running jobs in background

Same usage throughout all the shells

• Useful For command line jobs which output can be examined later,

especially for time consuming ones.

• To start graphical applications from the command line and then continue with the mouse. Starting a task: add & at the end of the line:

```
kdevelop    &
kcalc    &
```

[Ctrl] Z – stopping the job (not killing)

# Background job control

jobs - Returns the list of background jobs from the same shell

```
Examples:    [1]- Running/stopped   kdevelop    &
             [2]+ Running /stopped  kcalc    &
```

fg - Puts the nth background job in foreground mode, can’t run anything else in this terminal

fg n

bg cmd - moves the current task to the background, and lets me continue using the terminal

[Ctrl] Z

bg n

kill cmd - kills the job.

kill %<n>

# Measuring elapsed time

time - timing a simple command or give resource usage

```
time job_name

time ping 127.0.0.1

<...command output...>

real 0m2.304s (actual elapsed time)

user 0m0.449s (CPU time running program code)

sys 0m0.106s (CPU time running system calls)
```

real = user + sys + waiting

waiting = I/O waiting time + idle time (running other tasks)

# Listing all processes

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

Processes are displayed by the ps command

Example: ps -ux -Lists all the processes belonging to the current user

ps -aux -Lists all the processes running on the system

ps -aux | grep bart | grep bash

```
SER      PID     %CPU     %MEM      VSZ        RSS         TTY         STAT      START     TIME
COMMAND

bart       3039      0.0          0.2          5916      1380     pts/2        S
14:35          0:00        /bin/bash

bart       3134      0.0          0.2          5388      1380     pts/3        S
14:36          0:00        /bin/bash

bart       3190      0.0           0.2          6368     1360     pts/4        S
14:37          0:00        /bin/bash

bart       3416      0.0           0.0          0            0          pts/2        RW
15:07          0:00        [bash]
```

PID: Process id

VSZ: Virtual process size (code + data + stack)

RSS: Process resident size: number of KB currently in RAM

TTY: Terminal

STAT: Status: R (Runnable), S (Sleep), W (paging), Z (Zombie)...

# Live process activity(1)

**Top -** provides an ongoing look at processor activity in real time. It displays a listing of the most CPU-intensive tasks on the system.

**top** [**-**] [**d** delay] [**p** pid] [**q**] [**c**] [**C**] [**S**] [**s**] [**i**] [**n** iter] [**b**]

**-d** Specifies the delay between screen updates.

**-p** Monitor only processes with given process id.

**-i** Start **top** ignoring any idle or zombie processes.

**-C** display total CPU states instead of individual CPUs. This option only affects SMP systems.

**processes -** The total number of processes running at the time of the last update.

**"CPU states" -**Shows the percentage of CPU time in user mode, system mode, niced tasks, iowait and idle.

# Live process activity (2)

**Mem** -Statistics on memory usage, including total available memory, free memory, used memory, shared memory, and memory used for buffers.

**Swap** -Statistics on swap space, including total swap space, available swap space, and used swap space.

**PID -** The process ID of each task.

**PPID -** The parent process ID each task.

**UID -** The user ID of the task's owner.

**USER -** The user name of the task's owner.

**PRI -** The priority of the task.

**NI -** The nice value of the task. Negative nice values are higher priority.

**SIZE -** The size of the task's code plus data plus stack space, in kilobytes, is shown here.

# Live process activity (3)

Example:

```
top

top 15: 44:33 up 1:11, 5 users, load average: 0.98, 0.61, 0.59

Tasks: 81 total, 5 running, 76 sleeping, 0 stopped, 0 zombie

Cpu(s): 92.7% us, 5.3% sy, 0.0% ni, 0.0% id, 1.7% wa, 0.3% hi, 0.0% si

Mem: 515344k total, 512384k used, 2960k free, 20464k buffers

Swap: 1044184k total, 0k used, 1044184k free, 277660k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

3809 jdoe 25 0 6256 3932 1312 R 93.8 0.8 0:21.49 bunzip2

2769 root 16 0 157m 80m 90m R 2.7 16.0 5:21.01 X
```

# Killing processes (1)

kill <pids>

Sends an abort signal to the given processes. Lets processes save data and exit by themselves. Should be used first. Example:

kill 3039 3134 3190 3416

kill -9 <pids>

Sends an immediate termination signal. The system itself terminates the processes. Useful when a process is really stuck
(doesn't answer to kill -1).

kill -9 -1

Kills all the processes of the current user. -1: means all processes.

Under the hood Its really command to send signals => kill signal_name

# Killing processes (2)

killall [<signal>] <command>

Kills all the jobs running <command>.

Example:

```
killall bash
```

Xkill - Lets you kill a graphical application by clicking on it! Very quick! Convenient when you don't know the application command name.

## Killing processes (3)

pkill [<signal>] <pattern>

pgrep <pattern>

pgrep, pkill - look up or signal processes based on name and other

Examples:

pgrep –d “,” -u root ssh

Lists all the processes’ names issued by root, with a delimiter of “,”

pkill -9 –u root syslog

Sends signal TERMINATE to all processes issued by root named syslog

# Recovering from stuck graphics

If your graphical session is stuck and you can no longer type in your terminals, don't reboot!

It is very likely that your system is still fine. Try to access a different console by pressing the [Ctrl][Alt][F1] keys (or [F2],[F3] for more text consoles)

In the text console, you can try to kill the guilty application.
Once this is done, you can go back to the graphic session by pressing [Ctrl][Alt][F5] or [Ctrl][Alt][F7] (depending on your distribution)

If you can't identify the stuck program, you can also kill all your processes:

```
kill -9 -1
```

You are then brought back to the login screen.

# Sequential commands

- Can type the next command even when the current one is not over.
- Can separate commands with the ; symbol:

```
echo “I love thee”; sleep 10; echo “ not”
```

Conditionals: use || (or) or && (and):

```
cat God || echo “Sorry, God is busy”
```

Runs echo only if the first command fails

```
ls ~sd6  &&  cat ~sd6/* > ~sydney/recipes.txt
```

Only cats the directory contents if the ls command succeeds (means read access).

## Quoting (1)

**Double (") quotes** can be used to prevent the shell from interpreting spaces as argument separators, as well as substitute variables with values.

```
> echo "Hello World"
Hello World
> echo "You are logged as $USER"
You are logged as benny
> echo *.log
find_prince_charming.log cosmetic_buys.log
> echo "*.log"
*.log
```

## Quoting (2)

**Single quotes** (the ~ key) bring a similar functionality, but what is between quotes is never substituted

```
> echo 'You are logged as $USER'
You are logged as $USER
```

Back quotes (`) can be used to call a command within another

Example:

```
    a=`uname –r`
    echo $a
```

Back quotes can be used within double quotes

```
> echo "You are using Linux `uname -r`"
You are using Linux 2.6.91.6
```

# Shell & Env variables

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

Shells let the user define variables.

Those are called Shell Variables. They can be reused in shell commands.

```
my_shell_var="This is a shell variable"
echo "$my_shell_var"
```

Convention: lower case names

You can also define Environment Variables. Variables that are also visible within scripts or executables called from the shell.

```
export MY_ENV_VAR="This variable can be used in my app"
echo "$MY_ENV_VAR"
```

Convention: upper case names.

# Shell variables examples(1)

Shell variables (bash)

```
projdir=/home/marshall/coolstuff
ls $projdir; cd $projdir
```

Environment variables (bash)

```
cd $HOME; echo “I am $USER, and I’m at $PWD”
export DEBUG=1
./my_debuggable_application
```

(displays debug information if DEBUG is set)

```
env
```

Lists all defined environment variables and their value.

# Shell variables examples(2)

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

LD_LIBRARY_PATH - Shared library search path

DISPLAY - Screen id to display X (graphical) applications on.

EDITOR - Default editor (vi, emacs...)

HOME - Current user home directory

HOSTNAME - Name of the local machine

MANPATH - Manual page search path

PATH - Command search path

PRINTER - Default printer name

SHELL - Current shell name

USER - Current user name

## PATH environment variables

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

PATH
Specifies the shell search order for commands

/home/acox/bin:/usr/local/bin:/usr/kerberos/bin
:/usr/bin:/bin:/usr/X11R6/bin:/bin:/usr/bin

LD_LIBRARY_PATH

Specifies the shared library (binary code libraries shared by applications, like the C library) search order for ld

/usr/local/lib:/usr/lib:/lib:/usr/X11R6/lib

MANPATH
Specifies the search order for manual pages

/usr/local/man:/usr/share/man

# Alias

Shells let you define command aliases: shortcuts for commands you use very frequently.

**alias [-p] [name[=value] ...]**

**unalias [-a] [name ... ]**

Examples:

```
alias ll='ls -la‘
```
Useful to always run commands with default arguments.

```
alias frd='find_rambaldi_device –-asap --risky‘
```
Useful to replace very long and frequent commands.

```
alias cia='. /home/sydney/env/cia.sh‘
```
Useful to set an environment in a quick way

`unalias cia` - remove the alias name

## The which command

Before you run a command, which tells you where it is found (shows the full path of shell commands)

```
bash> which ls
        alias ls='ls color=tty‘ /bin/ls
tcsh> which ls
        ls: aliased to ls color=tty
bash> which alias
         /usr/bin/which: no alias in
        (/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin)
tcsh> which alias
        alias: shell builtin command.
```

## Command history

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

history - Displays the latest commands that you ran and their number.

- You can copy and paste command strings
- You can recall the latest command: !!
- You can recall a command by its number !1003

You can recall the latest command matching a starting string:

!cat (ctrl +r)

## HardDisk usage, Compressing and archiving

# HardDisk & Partitions

To see the layout of your connected Disk and Storage Hardware (root):

```
fdisk –l
```

Linux only see the mounted hardware, not all the hardware.

The all the logical partitions has to be mounted in order to be viewed.

The access to the hardware is by the special files in /dev usually named:

hda, hdb, hdc .. hda1, hda2, hda3, hdb1, hdb2 … sda, sdb, .. sda1, sda2 ..

To see the **mounted** (accessable) partitions, type and location use

```
mount
```

## Mounting devices (1)

All files accessible in a Linux system are arranged in one big tree. These files can be spread out over several devices.
The mount command serves to attach the file system found on some device to the big file tree. the umount command will detach it again.

To make filesystems on any device (internal or external storage) visible on your system, you have to moun them.

The first time, create a mount point in your system:
mkdir /mnt/usbdisk (example)

Now, mount it:
mount –t vfat /dev/sda1 /mnt/usbdisk

/dev/sda1 or /dev/sdb1 - physical device
t: specifies the filesystem (format) type (ext2, ext3, vfat, reiserfs, iso9660...)

## Mounting devices (2)

Mount options for each device can be stored in the /etc/fstab file.

The fstab file typically lists all available disks and disk partitions, and indicates how they are to be initialized by the system.

The mount command, reads the fstab file to determine which options should be used when mounting the specified device

```
# /etc/fstab: static file system information.
# <file system> <mount point> <type> <options> <dump> <pass>
     proc            /proc        proc     defaults    0          0
    /dev/hda3                     ext3     defaults,   0          1
    /dev/hda4         /home       ext3     defaults    0          2
    /dev/hda2         /root2      ext3     defaults    0          2
```

```
mount /proc
mount /media/cdrom0
```

# Creating file-systems

Examples

- mkfs.ext3  /dev/sda1

Formats your USB key (/dev/sda1: 1st partition raw data) in ext3 format.

- mkfs.vfat  -v –F 32 /dev/sda1 (-v:verbose)

Formats your USB key back to FAT32 format.

- mkfs.vfat –v –F 32 disk.img

Formats a disk image file in FAT32 format.

# Measuring disk space

df -h <dir>

Returns disk usage and free space for the file-system containing the given directory.

Similarly, the -h option only exists in GNU df.

Example:

df –h

```
➢Filesystem    Size     Used     Avail    Use%     Mounted on
  /dev/hda5      9.2G    7.1G     1.8G     81%        /
```

# Measuring disc usage

Caution: different from file size!

du -h <file> (disk usage)

-h: returns size on disk of the given file, in human readable format: K (kilobytes), M (megabytes) or G (gigabytes) .

Without h: du returns the raw number of disk blocks used

by the file (hard to read).

Note that the h option only exists in GNU du.

du -sh <dir>

s: returns the sum of disk usage of all the files in the given

directory.

# Compressing

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

Very useful for shrinking huge files and saving space

gzip <file> (compress) gunzip <file> (de-compress)

GNU zip compression utility. Creates .gz files (original file is deleted)

Ordinary performance (similar to Zip).

bzip2 <file> bunzip2 <file>

More recent and effective compression utility, Creates .bz2 files. Usually 2025% compression rate.

Using 7zip => Much better compression ratio than bzip2 (up to 10 to 20%).

See the 7zip page for details.

# Archiving (1)

Useful to backup or release a set of files into a single2 file

tar: originally “tape archive”

<u>Creating an archive</u>:

```
tar -cvf <archive> <files or directories>
```

-c: create. The opposite action is extract (-x)
-v: verbose. Useful to follow archiving progress.
-f: file. Archive created in/extracted from file (tape used otherwise).

Examples:

```
tar -cvf /backup/home.tar /home      #the folder /home has been tarred
bzip2 /backup/home.tar               #the tar file has been  zipped

bunzip2 /backup/home.tar.bz2  #the zipped file has been extracted
tar -xvf /backup/home.tar     #the tar file has been extracted
```

# Archiving (2)

Extracting and testing an archive:

-x: extract. Extract files from archive
-t: test

Extracting all the files from an archive:

```
tar -xvf <archive>
```

Extracting just a few files from an archive:

```
tar  -xvf <archive> <files or directories>
```

Viewing the contents of an archive or integrity check:

```
tar -tvf <archive>
```

Files or directories are given with paths relative to the archive's directory.

Use -C DIR or --directory=DIR, to specify the extraction location.

# Extra options in GNU tar

tar = gtar = GNU tar on GNU / Linux

Combination of compressing and archiving on the fly.
Useful to avoid creating huge intermediate files

Much simpler than with tar and bzip2\gzip!

- -j option: [un]compresses on the fly with bzip2
- -z option: [un]compresses on the fly with gzip

Examples (which one will you remember?)

```
gtar -jcvf bills_bugs.tar.bz2  bills_bugs
```

```
tar cvf bills_bugs | bzip2 > bills_bugs.tar.bz2
```

# 7zip (1)

http://www.7zip.org/

Now the best solution for your archives! License: GNU GPL

7zip compresses much better than bzip2 (up to 10 or 20%) and of course zip (30 to 50 %).

Benchmark compressing Knoppix 5.0.1: 22% (vs. bzip2)!

*Caution: 7zip cannot replace tar for archiving on Unix.

It doesn't keep file owner and group information,

but of course keeps file permissions.

Use it to compress tar archives!

## 7zip (2)

7zip supports strong AES256 encryption.

No need to encrypt in a separate pass.

At last a solution available for Unix and Windows!

The tool supports most other compression formats:

zip, cab, arj, gzip, bzip2, tar, cpio, rpm and deb.

## Miscellaneous
## Checking File Integrity and Comparing Files / Directories

## Checking file integrity

Very low cost solution to check file integrity

```
md5sum  files*.iso > MD5SUM
```

Computes a MD5 (Message Digest Algorithm 5) 128 bit checksum of the given files. Usually redirected to a file.

Example output:

```
db8c7254beeb4f6b891d1ed3f689b412 file1.iso
2c11674cf429fe570445afd9d5ff564e file2.iso
f88f6ab5947ca41f3cf31db04487279b file3.iso
6331c00aa3e8c088cc365eeb7ef230ea file4.iso
```

```
md5sum –c MD5SUM
```

Checks the integrity of the files in MD5SUM by comparing their actual MD5 checksum with their original one.

# Comparing files and directories

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

```
diff file1 file2
```

Reports the differences between 2 files, or nothing if the files are identical.

```
diff –r dir1/ dir2/
```

Reports all the differences between files with the same name in the 2 directories.

In order to see the differences in detail, better use graphical tools!

For instance kDiff3

## compare files and merge differences

tkdiff - http://tkdiff.sourceforge.net/

kompare

kDiff3

Gvimdiff

Much more

## Miscellaneous
## Looking for files

# The find command

Lists all the  files with the pattern requested

```
find  . -name “*.pdf ”
```

Lists all the *.pdf files in the current (.) directory or subdirectories. You need the double quotes to insure the shell treats the * character as a wild card.

```
find docs -name "*.pdf" -exec xpdf {} ';‘    # opens each in a different command
find docs -name "*.pdf" -exec xpdf {} ‘+‘    # opens all in the same command
```

Application

Find Results

Finds all the *.pdf files in the docs directory and displays one after the other.

# The locate command

Much faster regular expression search alternative to find

locate keys
Lists all the files on your system with keys in their name.

locate “*.pdf”
Lists all the *.pdf files available on the whole machine

locate “*/eclipse” *** very useful for finding files / directories fast ***
Lists all the files named eclipse

locate “/home/fridge/*beer*”
Lists all the *beer* files in the given directory (absolute path)

locate is much faster because it indexes all files in a dedicated database, which is updated on a regular basis. To update manualy use **updatedb**

find is better to search through recently created files.

## Network Administration Basics

# Network setup (1)

ifconfig -a
Prints details about all the network interfaces available on your system.
ifconfig eth0
Lists details about the eth0 interface

ifconfig eth0 up / down
Power ups / Shuts down the eth0 interface (frees its IP address).

ifconfig eth0 192.168.0.100 netmask 255.255.255.0
Assigns the 192.168.0.100 IP address to eth0 ( IP address per eth).

$ sudo dhclient eth0
Now obtain fresh IP
sudo dhclient –r eth0
releases the current lease, and once the lease has been released, the client exits

# Network testing - ping

```
ping rt-ed.co.il
ping 192.168.1.1
```

Tries to send packets to the given machine and get acknowledgment packets in return.

```
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=0 ttl=150 time=2.51 ms
64 bytes from 192.168.1.1: icmp_seq=1 ttl=150 time=3.16 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=150 time=2.71 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=150 time=2.67 ms
```

When you can ping your gateway, your network interface works fine.
When you can ping an external IP address, your network settings are correct!

# Network setup (2)

route add default gw 192.168.0.1

Sets the default route for packets outside the local network.
The gateway (here 192.168.0.1) is responsible for sending them to the next gateway, etc., until the final destination.

route

Lists the existing routes

route del default
route del <IP>

Deletes the given route
Useful to redefine a new route.

# Network setup summary

Used only for simple cases with 1 interface, no dhcp server...

- Connect to the network (cable, wireless card or device...)

- Identify your network interface:

```
ifconfig ethx
```

- Assign an IP address to your interface (assuming eth0)

```
ifconfig eth0 192.168.0.100
```
(example)

- Add a route to your gateway (assuming 192.168.0.1) for packets outside the network:

```
route add default gw 192.168.0.1
```

- Test the network using ping

# SSH - Secure SHell

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

• Used to access a remote computer on most Linux/Unix systems
(write shell commands just as if you were sitting at the workstations) just like telnet.

• Telnet method poses a danger in that everything that you send or receive over that session is visible in plain text so anyone can easily "sniff" the connection.

• Not only does it encrypt the session, it also provides better authentication facilities, as well as features like secure file transfer, X session forwarding, port forwarding and more .

Syntax:

```
ssh  user_name@computer_name
ssh  user_name@computer_ip_address
```

Windows:

use PuTTY for SSH

# SCP - Secure Copy

- scp - remote file copy program, copies files between hosts on a network.

- uses ssh for data transfer authentication and provides the same security as SSH does.

- Unlike rcp (another network copy utility), scp will ask for passwords for authentication..

- Copies between two remote hosts are permitted.

<u>Syntax:</u>

```
scp  -r user_name@computer_name:/home/*  /home
```

<u>Windows :</u>

```
pscp.exe  bennyc@192.168.42.38:/home/minicom.log   c:
```

## Name resolution

Your programs need to know what IP address corresponds to a given host name (such as kernel.org)

Domain Name Servers (DNS) take care of this.
You just have to specify the IP address of 1 or more DNS servers in your /etc/resolv.conf file:

```
nameserver 217.19.192.132
nameserver 212.27.32.177
```

The changes takes effect immediately!

## Tweaking kernel’s Networking params

All TCP/IP tuning parameters are located under /proc/sys/net/...
With these parameters you can change kernel network configuration.

here is a list of the most important tunning parameters:

/proc/sys/net/core/rmem_max - Maximum TCP Receive Window
/proc/sys/net/core/wmem_max - Maximum TCP Send Window
/proc/sys/net/ipv4/tcp_rmem - memory reserved for TCP receive buffers
/proc/sys/net/ipv4/tcp_wmem - memory reserved for TCP send buffers

# sysctl

sysctl is used to modify kernel parameters at runtime.
The parameters available are those listed under /proc/sys/.

Example for modifying kernel receiver buffer to 16Mbyte:

```
  sysctl -q -w net.core.rmem_max = 16777216
  sysctl -q -w net.core.rmem_default=16777216
```

# iperf

iperf is a tool to measure the bandwidth and the quality of a network link.
The network link is delimited by two hosts running iperf.

The quality of a link can be tested as follows:
- Latency (response time or RTT): can be measured with the Ping command.
- Jitter (latency variation): can be measured with an Iperf UDP test.
- Datagram loss: can be measured with an Iperf UDP test.

The bandwidth is measured through TCP tests.

To be clear, the difference between TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) is that TCP use processes to check that the packets are correctly sent to the receiver whereas with UDP the packets are sent without any checks but with the advantage of being quicker than TCP.
Iperf uses the different capacities of TCP and UDP to provide statistics about network links.

# iperf

```
*******************  sending host*****************************
iperf -c 2.2.2.11 -u -b 2G


*******************  receiving host*****************************
  sysctl -q -w net.core.rmem_max=16777216
  sysctl -q -w net.core.rmem_default=16777216
  iperf -s –u
```

## minicom

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

minicom is a communication on serial devices, supports script language interpreting, capturing to file, multiple users with individual configurations, and more.
-s  used for setup ,edits the system-wide defaults in /etc/minirc.dfl

```
ctrl+a =>  o – options
           q – quit
```

Use apt-get under Debian / Ubuntu Linux, enter:

```
$ sudo apt-get install minicom
```

If you are using Red hat Linux (RHEL) / CentOS / Fedora Linux, enter:

```
# yum install minicom
```

## System Administration Basics

# LSPCI Utility

display information about PCI buses in the system and devices connected to them.

You can display a brief list of devices, or a more detailed one.

-v – Be verbose, Include the Kernel Drivers for each device and Modules.
-vv – Be more verbose. Show a more detailed version of the last command.
-vvv – Be as verbose as possible. Show all the details.

-t – Show a tree-like diagram containing all buses, bridges, devices and connections between them.

-b – Bus-centric view. Show all IRQ numbers and addresses as seen by the cards on the PCI bus instead of as seen by the kernel

# Kernel Modules

Add Code \Functionality to the Linux kernel while it is running.

lsmod  - list kernel modules

insmod module_name.ko - load kernel modules

rmmod  module_name - unload kernel modules

# dmesg

print the kernel ring buffer

dmesg [ -c ] [ -n level ] [ -s bufsize ]

dmesg is used to examine or control the kernel ring buffer.
The program helps users to print out their bootup/device driver messages.
instead of copying the messages by hand.

**- c:** Clear the ring buffer contents after printing.
-**sbufsize**: Use a buffer of size bufsize to query the kernel ring buffer. This is 16392 by default.
-**nlevel** : Set the level at which logging of messages is done to the console.
For example, -n 1 prevents all messages, expect panic messages, from appearing on the console.
All levels of messages are still written to /proc/kmsg

# strace

Used for tracing the system calls of a program.
when it is run in conjunction with a program, it outputs all the calls made to the kernel by the program. In many cases, a program may fail because it is unable to open a file or because of insufficient memory. And tracing the output of the program will clearly show the cause of either problem.

```
$ strace <name of the program>
```

For example, I can run a trace on 'ls' as follows :

```
$ strace ls
```

this will output a great amount of data on to the screen. If it is hard to keep track of the scrolling mass of data, then there is an option to write the output of strace to a file instead which is done using the -o option. For example,

```
$ strace -o strace_ls_output.txt ls
```

.. will write all the tracing output of 'ls' to the 'strace_ls_output.txt' file.

# Linux services

A "service" is a program that starts automatically when you start your computer, and runs in the background. For example, the "network" service sets up your connection to the Internet and keeps it running correctly.

Using the command line (replace service-name with the name of the service you want, You must be root for this to work).

service service-name status - Check if a service is running:
service service-name start - Starting a service:
service service-name stop - Stopping a service:

For example:
Service network [start |stop| restart]

you can modify services by /etc/init.d/service-name action_taken (status, stop and start too).  look in /etc/init.d/ for the services available.

# RPM Packages

## RPM Packages - what are they

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

RPM (RPM Package Manager) is the most common software package manager used for Linux distributions. it allows you to distribute software already compiled, a user can install the software with a single command.

nowadays YUM is often used as an alternative.
An rpm consists of basically three parts: **a header**, **a signature**, and the (generally compressed) **archive itself**.

<u>**header**</u> - contains a complete file list, a description of the package, a list of the features and libraries it provides, a list of tools it requires (from other packages) what (known) other packages it conflicts with, and more.

**Signature** – verification of the downloaded rpm

RPM is not the only package format around; other popular formats are:

- **.deb packages** used by Debian Linux
- **pkgadd** format used by Solaris

# RPM Packages

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

RPM enables to have of a compiled binary files along with the necessary configuration files that can be unpacked on the system with "one click" in working package installation and later upgraded to a new version or de-installed without remembering all the places where the fie went.

An RPM database of all available applications is installed on your computer (in /var/lib/rpm)

Most people use this RPM repository together with a tool that allows to automatically download an install RPM packages and resolve dependencies. You have the choice of different tools, like Apt, Smart, Yum, up2date or Red Carpet.

# RPM Packages- Installation/Removal

One of the more complex but highly useful commands allows you to install packages via FTP or HTTP, to install **foobar-1.0-1.i386.rpm**

```
rpm -ivh ftp://ftp.redhat.com/pub/redhat/rh-2.0-beta/RPMS/foobar-1.0-1.i386.rpm
```

Install the package with dependencies

```
rpm -ivh --aid #  option --aid installs dependencies
```

To see what package provides required dependencies

```
rpm -qp --provides http-2.0.4-9.i386.rpm
```

Force installation of the package disregarding dependencies.

```
rpm -ivh --force --nodeps ***.rpm
```

## RPM Packages- Upgrade/Uninstall

Notice it's a capital U. Upgrades an installed program with a newer package.

```
rpm -U   packagename
rpm -Uvh http: <server>
```

Uninstall the package

rpm {-e | --erase} [--allmatches] [--nodeps] [--noscripts] [--notriggers] [--test] PACKAGE_NAME...

```
 rpm --erase (-e) [name]
rpm -e packagename   # Note: Sometimes you need to delete dependencies first.
rpm -ev  --nodeps postfix  # removing ...
```

Verification / integrity checking

`rpm Va` -- verify all packages (you need to grep for interesting information)

`rpm --verify (-V)[name]` will tell you if all files of the packages are still (or at all) installed.

`rpm -Vp rpmfile - verifyes` installed package against zip package headers

## yum – Yellow-Dog Updater Modified

An easy way to install RPMs along with associated dependencies is to use yum or Debian based apt-get) for RPM.

YellowDog Linux is a distribution of Linux created for the PPC (Power PC) architecture and is rpm based

It has a small code base than apt for RPM, it is written in python and it makes upgrading to new Red Hat releases relatively easy

For help use- `man yum`

```
yum –y install gcc
```

# yum Basic Commands

Don't bother to ask me for verification

```
yum –y install …
```

update all installed packages

```
yum  update
```

Get information specific to the supplied package name.

```
yum info package_name
```

Install the supplied package name.

```
yum install package_name
```

Remove the supplied package name.

```
yum remove package_name
```

list all the installed packages

```
yum list installed | less
```

# yum Basic Commands

מרכז להכשרה מקצועית והשמה בתעשיית ההייטק

show you what packages need to be updated.  do this before the ‘yum update’ command

```
yum list updates | less
```

By default, yum is messy. It tends to leave behind extra files. So, every so often, it’s a good idea to tell yum to clean itself up.

```
yum clean all
```

# clam Antivirus

Linux is much less vulnerable than other O.S.'s since your processes are limited to your privileges,  if you don't go running unknown downloaded applications there is a good chance you'll stay safe.

But just to stay on the safe side You can and should use the known clamav / clamd (clam anti-virus/clam daemon)

## Installing clamav / clamd

By default clamav doesn't come with centos or (yum for that matter). You have to find rpm repository and install it. Here is how you install clam antivirus (freely available) in centos

1. rpm -Uhv http://apt.sw.be/redhat/el5/en/i386/rpmforge/RPMS/rpmforge-release-0.3.6-1.el5.rf.i386.rpm
2. after installing  issue  **yum install clamd  or  yum install clamav**

## Basic commands using the clam

1. **freshclam -** update the antivirus database
2. **clamav -r /home -** run the clam antivirus

מרכז להכשרות מקצועיות והשמה בתעשיית ההייטק

Real Time College

# Bash Scripting

# דברי פתיחה

**מרכז להכשרות מקצועיות והשמה בתעשיית ההייטק**

עם פתיחת קורס**Bash Scripting**
צוות **Real Time Group** מאחל לכם חווית לימוד איכותית ופרודוקטיבית.

קורסי ההכשרה מקצועית מכוונים לנתב את הקריירה שלך למסלול שבו תוכל/י
להשתלב בתעשייה במקצוע שמעניין אותך ולמקסם את הידע בצורה אופטימאלית האפשרית.

תרגישו נוח לפנות לצוות בכל נושא .

בני כהן
מנכ"ל

## Introduction

## References

מרכז להכשרות מקצועיות והשמה בתעשיית ההייטק

- Bash Beginners Guide (http://tldp.org/LDP/Bash-Beginners-Guide/)
- http://linux.tnc.edu.tw/techdoc/

# Outline

מרכז להכשרות מקצועיות והשמה בתעשיית ההייטק

## ? Why Shell

- Shell program interprets user commands.
- Can be read from a file called the shell script or shell program. Shell scripts are interpreted, not compiled.
- In Linux, Bash (Bourne Again shell) is the default

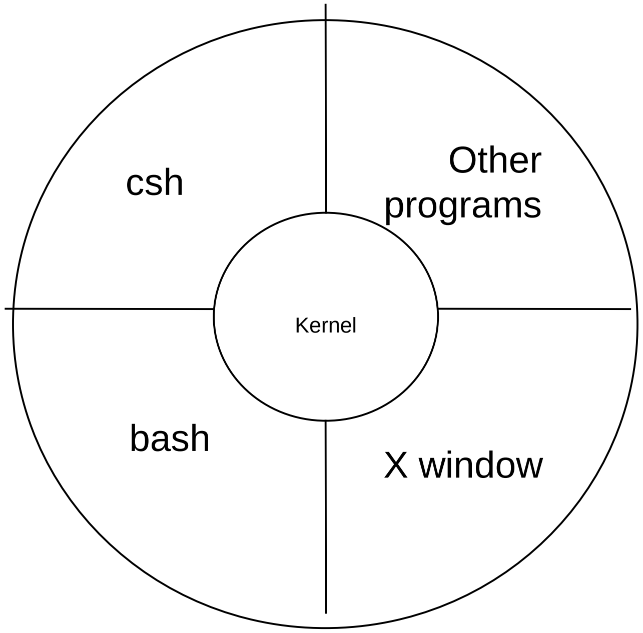
- Why Shell?
  - For routing jobs, such as system administration, without writing programs
  - However, the shell script is not efficient, therefore, can be used for prototyping the ideas

- For example,

```
$ ls –al | more              (better format of listing directory)
$ man bash | col –b | lpr    (print man page of man)
```

## ?What is a Shell

- A shell script is a sequence of commands for which you have a repeated use.
- This sequence is typically executed by entering the name of the script on the command line. Alternatively, you can use scripts to automate tasks.
- The Shell is the interface between end user and the Linux system, similar to the commands in Windows.
- Scripts are read and executed line per line and should have a logical structure.

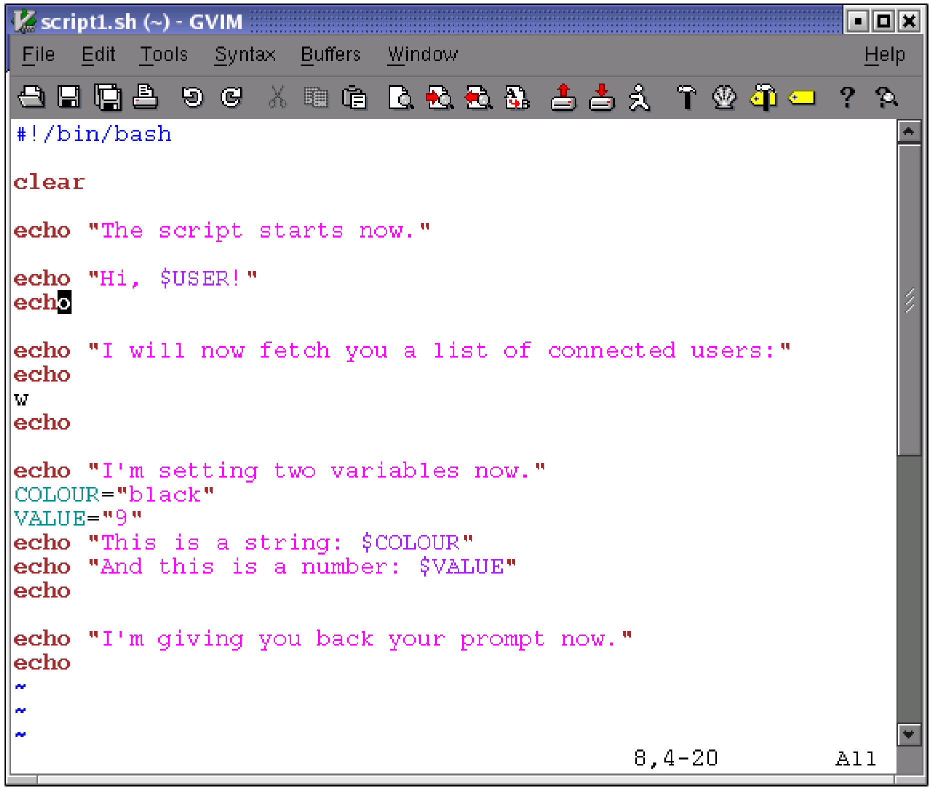


# ?What is BASH

- Bash is the GNU shell, compatible with the Bourne shell and incorporating many useful features from other shells.
- When the shell is started, it reads its configuration files. The most important are:
  - /etc/profile
  - ~/.bash_profile
  - ~/.bashrc

- Bash is installed as in /bin/sh
- Check the version: `% /bin/sh –version`

- **To create a shell script**:
  - open a new empty file in your editor
  - Give your script a sensible name that gives a hint about what the script does.
  - Make sure that your script name does not conflict with existing commands.
  - Make sure that your script end in .sh.



```
#!/bin/bash

clear

echo "The script starts now."

echo "Hi, $USER!"
echo

echo "I will now fetch you a list of connected users:"
echo
w
echo

echo "I'm setting two variables now."
COLOUR="black"
VALUE="9"
echo "This is a string: $COLOUR"
echo "And this is a number: $VALUE"
echo

echo "I'm giving you back your prompt now."
echo
~
~
~
                                                      8,4-20        All
```

מרכז להכשרות מקצועיות והשמה בתעשיית ההייטק

- Add a comment-add a hash mark then write your comment, all
- scripts will start with the line : #!/bin/bash
- It might be a good idea to create a directory ~/scripts to hold your scripts. Add the directory to the contents of the PATH variable:
  **export PATH="$PATH:~/scripts"**
- Executing the script
  $ chmod u+x script_name.sh
  $ ./script_name.sh

## Writing a Script

- Use text editor to generate the “first” file

```
#!/bin/bash
# file name is first - this file looks for the files containing ERR and prints them
for file in *
do
   if grep –q ERR $file
   then
      echo $file
   fi
done
exit 0
```

exit code, 0 means successful

```
bin/bash  first$
      chmod +x first$
first$
```

make sure . is include in PATH parameter

# Shell as a Language

- We can write a script containing many shell commands
- Interactive Program:
  - grep files with POSIX string and print it

```
for file in *
do
      if grep –l POSIX $file
      then
      more $file
      fi
done
```

There is a file with POSIX in it - ‘*’ is wildcard

```
$ more `grep –l POSIX *`
$ more $(grep –l POSIX *)
$ more –l POSIX * | more
```

# Pipe and Redirection

מרכז להכשרות מקצועיות והשמה בתעשיית ההייטק

- **Redirection** (< or >)

  `$ ls –l > lsoutput.txt` (save output to lsoutput.txt)
  `$ ps >> lsoutput.txt` (append to lsoutput.txt)
  `$ more < killout.txt` (use killout.txt as parameter to more)
  `$ kill -l 1234 > killouterr.txt 2>&1` (redirect to the same file)
  `$ kill -l 1234 >/dev/null 2>&1` (ignore std output)

- **Pipe** (|)
  - Process are executed concurrently

  `$ ps | sort | more`
  `$ ps –xo comm | sort | uniq | grep –v sh | more`
  `$ cat mydata.txt | sort | uniq | > mydata.txt` (generates an empty file !)

## Variables

מרכז להכשרות מקצועיות והשמה בתעשיית ההייטק

- shell variables are in uppercase characters by convention. Bash keeps a list of two types of variables:
  - **Global variables -** or environment variables are available in all shells. The env or printenv commands can be used to display environment variables.
  - **Local variables** - are only available in the current shell, Using the **set** built-in will display a list of all variables (including environment variables ) and functions

- To set a variable in the shell, use:

```
> VARNAME="value"
```

Putting spaces around the equal sign will cause errors. It is a good habit to quote content strings when assigning values to variables

```
> echo $VARNAME
```

## Creating Variables

מרכז להכשרות מקצועיות והשמה בתעשיית ההייטק

Variables are needed to be declared, note it is case-sensitive
(e.g. foo, FOO, Foo)

- Add '$' for storing values

```
$ variable=Hello
$ echo $variable
Hello
$ variable=7+5
$ echo $variable
7+5
$ variable="yes dear"
$ echo $variable
yes dear
$ read variable
Hola!
$ echo $variable
Hola!
```

## Creating Variables

מרכז להכשרות מקצועיות והשמה בתעשיית ההייטק

Variables are needed to be declared, note it is case-sensitive (e.g. foo, FOO, Foo)

- Add '$' for storing values

```
$ variable=Hello
$ echo $variable
Hello
$ variable=7+5
$ echo $variable
7+5
$ variable="yes dear”
$ echo $variable
yes dear
$ read variable
Hola!
$ echo $variable
Hola!
```

## Exporting Variables

מרכז להכשרות מקצועיות והשמה בתעשיית ההייטק

- Variables created like the ones in the example above are local variables, they are only available to the current shell (child processes of the current shell will not be aware of this variable)

- In order to create environment variables (pass variables to a subshell), we need to export them using the export built-in command.

  – **export VARNAME="value"**

# Environment Variables

מרכז להכשרות מקצועיות והשמה בתעשיית ההייטק

Environment variables in the bash shell help you in several ways. Certain built-in variables change the shell in ways that make your life a little easier, and you can define other variables to suit your own purposes. Here are some examples of built-in shell variables:

$HOME - home directory

```
echo $HOME
    /home/chemie
```

$PATH - directories to search for exec commands

```
PATH=$PATH:$HOME/bin
```

- So if PATH was set to **/bin:/usr/bin:/usr/local/bin** beforehand, it would now have the value**/bin:/usr/bin:/usr/local/bin:/home/hermie/bin**.

env - list the current values of all environment variables

## Using Parameters

מרכז להכשרות מקצועיות והשמה בתעשיית ההייטק

:Example of a script using variables

Edit file try_var.sh

```
#!/bin/bash
variable="Hello"
echo $variable
echo "The program $0 is now running"
echo "The parameter list was $*"
echo "The second parameter was $2"
echo "The first parameter was $1"
echo "The user’s home directory is $HOME"
echo "The PATH environment variable is $PATH"
echo "Please enter a new greeting"
read variable
echo $variable
echo "The script is now complete"
exit 0
```

## Using Parameters

Running the script with additional external parameters

- $\underbrace{\text{\$./try\_var}}_{\%0}\ \underbrace{\text{foo}}_{\%1}\ \underbrace{\text{bar}}_{\%2}\ \underbrace{\text{baz}}_{\%3}$

Output:

```
Hello
The program ./try_var is now running
The second parameter was bar
The first parameter was foo
The parameter list was foo bar baz
The user's home directory is /home/ychuang
Please enter a new greeting
Hola
Hola
The script is now complete
```

## Quoting characters

- A lot of keys have special meanings in some context or other.
  Quoting is used to remove the special meaning of characters or words:
- quotes can disable special treatment for special characters, they can prevent reserved words from being recognized as such.

**Single quotes**

used to preserve the literal value of each character

```
$ echo '$date'
$ date
```

**Double quotes**

Using double quotes the literal value of all characters enclosed is preserved, except for the dollar sign, the backticks (backward single quotes, ``) and the backslash.

```
$ a=5; echo "$a"
$ 5
```

## Quoting

Edit a “vartest.sh” file

```
#!/bin/bash

myvar="Hi there"

echo $myvar
echo "$myvar"
echo `$myvar`
echo \$myvar

echo Enter some text
read myvar

echo '$myvar' now equals $myvar
exit 0
```

**Output:**

```
Hi there
Hi there
./vartest.sh: line 7: Hi:
   Command not found

$myvar
Enter some text
Hello world
$myvar now equals hello world
```

# Quoting

מרכז להכשרות מקצועיות והשמה בתעשיית ההייטק

Edit a “vartest.sh” file

```
#!/bin/bash

myvar="Hi there"

echo $myvar
echo "$myvar"
echo `$myvar`
echo \$myvar

echo Enter some text
read myvar

echo '$myvar' now equals $myvar
exit 0
```

**Output:**

```
Hi there
Hi there
./vartest.sh: line 7: Hi:
   Command not found

$myvar
Enter some text
Hello world
$myvar now equals hello world
```

## Aliases

- An alias allows a string to be substituted for a word when it is used as the first word of a simple command.
- The shell maintains a list of aliases that may be set and unset with the **alias** and **unalias** built-in commands.

- Handling Aliases:

  to display a list of aliases - issue the alias cmd without options

```
$  alias
```

  to creating use:

```
$  alias dh='df -h'
```

  to remove aliases use:

```
$  unalias dh
```

# Conditional statements

The following topics are included:

1. The **if** statement
2. Using the exit status of a command
3. Comparing and testing input and files
4. if/then/else constructs
5. if/then/elif/else constructs
6. Using and testing the positional parameters
7. Nested **if** statements
8. Boolean expressions
9. Using **case** statements

# Introduction to if

- The if construction allows you to specify conditions in which you need to specify different courses of action to be taken depending on the success or failure of a command.

- Syntax:

```
if expression
      then statement ;
fi
```

'statement' is only executed if 'expression' evaluates to true

```
if expression
      then statement1
else
      statement2
fi
```

'statement1' is executed if 'expression' is true, otherwise 'statement2' is executed.

## Introduction to if (2)

מרכז להכשרות מקצועיות והשמה בתעשיית ההייטק

```
if expression1; then
        statement1
elif expression2
        then statement2
else
        statement3
fi
```

In this form there's added only the else if (elif) 'expression2' then 'statement2' which makes statement2 being executed if expression2 evaluates to true

```
#!/bin/bash
T1="foo"
T2="bar"
 if [ "$T1" = "$T2" ]; then
      echo expression evaluated as true
 else
       echo expression evaluated as false
 fi
```

Conditionals with variables

## Introduction to if (3)

```
#!/bin/bash
echo "Is it morning? Please answer yes or no"
read timeofday
if [ $timeofday = "yes" ]; then
  echo "Good morning"
else
  echo "Good afternoon"
fi
exit 0
```

output:

```
Is it morning? Please answer yes or no
yes
Good morning
```

## Checking files

מרכז להכשרות מקצועיות והשמה בתעשיית ההייטק

```
#!/bin/bash
echo "This scripts checks the existence of the messages file."
echo "Checking..."
if [ -f /var/log/messages ]
    then
            echo "/var/log/messages exists."
else
    echo
    echo "File Doesn't Exist"
fi
echo
echo "...done."
```

## Testing exit status

- The ? variable holds the exit status of the previously executed command (the most recently completed foreground process).

```
#!/bin/bash
if [ $? -eq 0 ]
      then
      echo 'That was a good job!'
fi

if  ! grep $USER /etc/passwd
      then echo "your user account is not managed locally";
fi

grep $USER /etc/passwd
if [ $? -ne 0 ] ; then
      echo "not a local account" ;
fi
```

# Numeric comparisons

מרכז להכשרות מקצועיות והשמה בתעשיית ההייטק

You can perform various numeric comparison using the following operators:

| Description | Syntax | Operator |
|---|---|---|
| Int 1 is equal to int2 | int eq int | eq |
| Int 1 is great or equal to int2 | int ge int | ge |
| Int 1 is great than  int2 | int gt int | gt |
| Int 1 is little or equal to int2 | int le int | le |
| | | |

## Numeric comparisons

מרכז להכשרות מקצועיות והשמה בתעשיית ההייטק

```
if [ $weight -le $idealweight ]
    then
    echo "eat a bit more fat."
else
    echo "eat a bit more fruit."
fi
```

```
if [ $weight -gt $idealweight ]
    then
    echo "you are over ideal weight."
else
    echo " you are under or equal ideal weight "
fi
```

```
year=2016
if [ $[$year % 400] -eq "0" ]; then
    echo "This is a leap year. February has 29 days."
elif [ $[$year % 4] -eq 0 ]; then
    if [ $[$year % 100] -ne 0 ]; then
            echo "This is a leap year, February has 29 days."
    else
            echo "This is not a leap year. February has 28 days."
    fi
fi
```

# String comparisons

Here are some examples of comparing strings

```
#!/bin/bash
if [ "$var" == "" ];then
    echo variable is null
fi
```

```
#!/bin/bash
if [ "$var" != "" ];then
    echo variable is null
fi
```

```
if [ "$var" == "value" ] then
     echo is the same
fi
```

```
if [ "$var" != "value" ] then
    echo not the same
fi
```

```
if [ "$(whoami)" != 'root' ]; then
    echo "You have no permission to run $0 as non-root user."
exit 1;
fi
```

## Case Statements

```
#!/bin/bash
if [ $# -lt 2 ]
then
        echo "Usage : $0 Signalnumber PID"
        exit
fi
case "$1" in
1)  echo "Sending SIGHUP signal"
    kill -SIGHUP $2
     ;;
2)  echo  "Sending SIGINT signal"
    kill -SIGINT $2
     ;;
9) echo  "Sending SIGKILL signal"
     kill -SIGKILL $2
     ;;
*) echo "Signal number $1 is not processed"
     ;;
esac
```

Nested if statements might get confusing , its a good alternative to multilevel if-then-else-fi   statement.

## For Loop Structure

Allows repetitive statement to be run for a list of values.
A list of commands is executed for each value in the list.

**Syntax:**

```
for variable
do
statement
done
```

```
#!/bin/bash
for count in 1 2 3 4 5 6 7 8 9 ; do
    echo  $count
done
exit 0
```

```
ls *.xml > list
for i in `cat list`;
    do cp "$i" "$i".bak ;
 done
```

demonstrating the use of a **for** loop that makes a backup copy of each .xml file

Output:

```
1
2
.
.
9
```

Basically the variable count will get the next value in each iteration.

# While Loop Structure

The while construct allows for repetitive execution of a list of commands, as long as the command controlling the while loop executes successfully (exit status of zero).

**Syntax:**

A. While condition
B. do
C. statement
D. done

```
foo=1
while [ "$foo" -le 10 ]
do
  echo "here we go again"
  foo=$(($foo+1))
done
```

**Syntax:**

A. until condition
B. do
C. statement
D. done

As long as this command fails, the loop continues

```
foo=1
until [ "$foo" -ge 10 ]
do
  echo "here we go again"
  foo=$(($foo+1))
done
```

## Loop example

```
# Calculate the average of a series of numbers.
SCORE="0"
AVERAGE="0"
SUM="0"
NUM="0"
while true; do
    echo -n "Enter your score [0-100%] ('q' for quit): "; read SCORE;
    if (("$SCORE" < "0")) || (("$SCORE" > "100")); then
        echo "Be serious. Common, try again: "
    elif [ "$SCORE" == "q" ]; then
        echo "Average rating: $AVERAGE%."
        break
    else
        SUM=$[$SUM + $SCORE]
        NUM=$[$NUM + 1]
        AVERAGE=$[$SUM / $NUM]
    fi
done
```

## List

מרכז להכשרות מקצועיות והשמה בתעשיית ההייטק

A. AND (&&)

B. statement1 && statement2 && statement3 …

```
#!/bin/bash
touch file_one
rm -f file_two
if [ -f file_one ] && echo "Hello" && [ -f file_two ] && echo " there"
then
  echo "in if"
else
  echo "in else"
fi
exit 0
```

## List

מרכז להכשרות מקצועיות והשמה בתעשיית ההייטק

- OR (||)

A. statement1 || statement2 || statement3 …

```
#!/bin/bash
rm -f file_one
if [ -f file_one ] || echo "Hello" || echo " there"
then
  echo "in if"
else
  echo "in else"
fi
exit 0
```

# Arithmetical Logical Commands

- exprevaluate expressions

```
$ x=`expr $x + 1`
```
(Assign result value expr $x+1 to x)

Also can be written as

```
$  x=$(expr $x + 10 )
```

| | |
|---|---|
| expr1 \| expr2 (or) | expr1 != expr2 |
| expr1 & expr2 (and) | expr1 + expr2 |
| expr1 = expr2 | expr1 – expr2 |
| expr1 > expr2 | expr1 * expr2 |
| expr1 >= expr2 | expr1 / expr2 |
| expr1 < expr2 | expr1 % expr2 (module) |
| expr1 <= expr2 | |

# Printing Commands

A.printf    format and print data

- Escape sequence
  - \\ backslash
  - \a beep sound
  - \b - backspace
  - \f - form feed
  - \n - newline
  - \r - carriage return
  - \t - tab
  - \v - vertical tab
- Conversion specifier
  - %d    decimal
  - %c    character
  - %s    string
  - %%    print %

```
$ printf "%s\n" hello
Hello
 $ printf "%s %d\t%s" "Hi There" 15 "people"
 Hi There 15     people
```

# Trapping Commands

מרכז להכשרות מקצועיות והשמה בתעשיית ההייטק

- trap   action after receiving signal

**A. Syntax**: trap command signal

| Signal | description |
|---|---|
| HUP (1) | hung up |
| INT (2) | interrupt  (Crtl + C) |
| QUIT (3) | Quit (Crtl + \) |
| ABRT (6) | Abort |
| ALRM (14) | Alarm |
| TERM (15) | Terminate |

# Trap Command example

מרכז להכשרות מקצועיות והשמה בתעשיית ההייטק

```
#!/bin/bash
trap 'rm -f /tmp/my_tmp_file_$$' INT
echo creating file /tmp/my_tmp_file_$$
date > /tmp/my_tmp_file_$$
echo "press interrupt (CTRL-C) to interrupt"
while [ -f /tmp/my_tmp_file_$$ ]; do
        echo File exists
        sleep 1
done
echo The file no longer exists
```

.

.

.

.

:OUTPUT

```
creating   file /tmp/my_file_141
press interrupt (CTRL-C) to interrupt …
File exists
File exists
File exists
File exists
^CThe file no longer exists
```

# Trap Command example (2)

```
#!/bin/bash
trap "" INT
echo creating file /tmp/my_tmp_file_$$
date > /tmp/my_tmp_file_$$
echo "press interrupt (CTRL-C) to interrupt"
while [ -f /tmp/my_tmp_file_$$ ]; do
echo File exists
sleep 1
done
echo we never get there
exit 0
```

:OUTPUT

```
Creating file /tmp/my_file_141
Press interrupt (CTRL-C) to interrupt …
File exists
File exists
^C^CFile exists
File exists
```

# Pattern Matching

מרכז להכשרות מקצועיות והשמה בתעשיית ההייטק

- find search for files in a directory hierarchy

A. find [path] [options] [tests] [actions]

options

| | |
|---|---|
| -depth | find content in the directory |
| -follow | follow symbolic links |
| -maxdepths N | fond N levels directories |
| -mount | do not find other directories |

tests

| | |
|---|---|
| -atime N | accessed N days ago |
| -mtime N | modified N days ago |
| -newer otherfile | name of a file |
| -type X | file type X |
| -user username | belong to username |

## Pattern Matching

מרכז להכשרות מקצועיות והשמה בתעשיית ההייטק

Find files newer than while2 then print only files

```
$ find . -newer "while2" -type f  -print
```

Find files either newer than while2, start with ‘_’

```
$  find . \( -name “_*”  -or -newer while2 \)  -type f  -print
```

Find files newer than while2 then list files

```
$ find . -newer "while2" -exec ls -l {} \;
```

# Pattern Matching

מרכז להכשרות מקצועיות והשמה בתעשיית ההייטק

- grep print lines matching a pattern

  (General Regular Expression Parser)

  grep [options] PATTERN [FILES]

option

| | |
|---|---|
| -c | print number of output context |
| -E | Interpret PATTERN as an extended regular expression |
| -h | Supress the prefixing of filenames |
| -i | ignore case |
| -l | surpress normal output |
| -v | invert the sense of matching |

```
$ grep in words.txt
$ grep –c in words.txt words2.txt
$ grep –c –v in words.txt words2.txt
```

# END