Shell

Using the command line

Orna Agmon

`ladypine at vipe.technion.ac.il`

Haifux

TOC

What is the shell?

- The shell is the wrapper around the system: a communication means between the user and the system
- The shell is the manner in which the user can interact with the system through the terminal.
- The shell is also a script interpreter. The simplest script is a bunch of shell commands.
- Shell scripts are used in order to boot the system.
- The user can also write and execute shell scripts.

Shell - which shell?

- There are several kinds of shells. For example, bash (Bourne Again Shell), csh, tcsh, zsh, ksh (Korn Shell). The most important shell is bash, since it is available on almost every free Unix system. The Linux system scripts use bash.
- The default shell for the user is set in the /etc/passwd file. Here is a line out of this file for example: `dana:x:500:500:Dana,,,:/home/dana:/bin/bash`
- This line means that user dana uses bash (located on the system at /bin/bash) as her default shell.

Starting to work in another shell

If Dana wishes to temporarily use another shell, she can simply call this shell from the command line:

```
[dana@granada ~]$ bash
dana@granada:~$ #In bash now
dana@granada:~$ exit
[dana@granada ~]$ bash
dana@granada:~$ #In bash now, going to hit ctrl D
dana@granada:~$ exit
[dana@granada ~]$ #In original shell now
```

chsh - Changing the default shell

If Dana wishes to change her default shell, she can use the chsh command:

```
[dana@granada ˜]$ echo $SHELL
/bin/bash
[dana@granada ˜]$ chsh
Password:
Changing the login shell for dana
Enter the new value, or press return for the default
        Login Shell [/bin/bash]: /bin/tcsh
[dana@granada ˜]$ echo $SHELL
/bin/bash
[dana@granada ˜]$ su dana
Password:
[dana@granada ˜]$ echo $SHELL
/bin/tcsh
```

Every time you run it

For many programs, there is a file called .{program-name}rc. This file contains commands to execute automatically every time the program starts running.
For example:

- .vimrc (used for gvim as well as vim)
- .bashrc
- .cshrc (used for both tcsh and csh)

Where are my .*rc files?

- Those files are usually located in the home directory.
- All those files begin with a period, so they are not listed using `ls`, only `ls -a`.

```
[dana@granada ~]$ ls
dummy
[dana@granada ~]$ ls -a
.    .alias          .bash_profile  .cshrc   .pi
..   .bash_history   .bashrc        dummy    .vi
```

Every time the shell starts - example

When updating the runcom file, it does not take effect immediately in the terminal you are using, and you need to *source* it (read it explicitely).
Let's watch Dana teach her shells to sing:

```
[dana@granada ~]$ tcsh
[dana@granada ~]$ unalias lll
[dana@granada ~]$ alias lll echo yehezkel
[dana@granada ~]$ lll
yehezkel
[dana@granada ~]$ bash
dana@granada:~$ unalias lll
bash: unalias: lll: not found
dana@granada:~$ alias lll="echo yehezkel"
dana@granada:~$ lll
yehezkel
dana@granada:~$
```

Permanent Changes

To make this change happen every time we start the shell, we insert the change in the .*rc file:

```
[dana@granada ˜]$ unalias lll
[dana@granada ˜]$ vi .cshrc #Here we add alias l
to the bottom of the .cshrc file
[dana@granada ˜]$ lll
lll: Command not found.
[dana@granada ˜]$ source .cshrc
[dana@granada ˜]$ lll
yehezkel
```

In bash, we state the same line (alias lll='echo yehezkel') in the .bashrc file, and source it using the . command.

Caution when sourcing rc file

- If you make bad syntax error in the rc file of an application, you may not be able to re-run it until you have fixed the rc file.
- This is most problematic when the program is the shell.
- keep a back up copy of your rc file. Even better to keep versions. See *rcs*, for example.
- For shell rc files: keep an open terminal working without sourcing the rc file, in case you messed up your own shell.

alias

- Create short ways to say a long command using alias:

```
[ladypine@granada ~]$ grep efnet ~/.cshrc
alias efnet 'BitchX -Nan ladypine irc.inter.r
[ladypine@granada ~]$ which efnet
efnet:   aliased to BitchX -Nan ladypine irc.
[ladypine@granada ~]$
```

- Remember to run a command in a certain way using alias:

```
[ladypine@granada ~]$ grep rm ~/.cshrc
alias rm 'rm -i'
```

- To use the original command once, escape the command: `\rm`
- To stop aliasing use *unalias*.

alias in programming

- Use full paths to commands when possible - on POSIX systems, utilities are located in specific places.
- When using commands without paths - use escaped commands - you never know how the users aliased their commands. aliases are not always available. Depends on the shell.

Shell variables

- There are two kinds of shell variables: regular variables, which are local, and environment variables, which are inherited by the programs executed from the shell.
- Setting environment variables: In bash

  ```
  export var=value
  ```

  In tcsh

  ```
  setenv var value
  ```

echo

The *echo* command quotes back what you told it to say. Useful for debugging as well as communicating with the user.

```
[ladypine@granada ~]$ echo DISPLAY
DISPLAY
[ladypine@granada ~]$ echo $DISPLAY
:0.0
[ladypine@granada ~]$ echo $(DISPLAY)
Illegal variable name.
```

What went wrong in the last one??

Hold fast to your output

backticks ‘command‘ holds the output of the command.

```
ladypine@granada:˜$ whatis pwd
pwd (1)    - print name of current/working
directory
ladypine@granada:˜$ a=‘whatis pwd‘
ladypine@granada:˜$ echo $a
pwd (1) - print name of current/working
directory
ladypine@granada:˜$ a=$(whatis pwd)#bash specifi
ladypine@granada:˜$ echo $a
pwd (1) - print name of current/working
directory
ladypine@granada:˜$
```

What is this command?

Use the “which” command to know what the command invokes really.

```
[dana@granada ˜]$ tcsh
[dana@granada ˜]$ which lll
lll:     aliased to echo yehezkel
[dana@granada ˜]$ which swriter
/usr/lib/openoffice/program/swriter
```

To know more about commands:

- man - old style, one long file
- info - new, browseable
- pinfo - even newer
- whatis -short format
- apropos - search the man pages

grep - searching for text patterns

grep searches for regular expressions in the input. It can be used to search for a line in a file, as seen in the previous example:

```
[ladypine@granada ~]$ grep rm ~/.cshrc
alias rm 'rm -i'
```

It can also be used to find the file in which the expression is mentioned. For example, in order to search my mail folders (each is a file) for the word shell:

```
[ladypine@granada ~]$ grep shell mail/*
```

Important switches: -n to give the line number. -i for case insensitivity.

|||||pipeline|||||

The output of one command can be piped into another command by using a pipe. The output is then passed by blocks to the next command.
Concating commands via a pipe is one of the most powerful features of the shell.

Example: *apropos | grep*

```
[dana@granada ˜]$ apropos keymap
install-keymap (8)    - expand a given keymap and install it as boot-time k
keymaps (5)           - keyboard table descriptions for loadkeys and dumpke
XChangeDeviceKeyMapping (3x) - query or change device key mappings
XFreeModifierMap XModifierKeymap (3x) [XChangeKeyboardMapping] - manipulat
XGetDeviceKeyMapping (3x) - query or change device key mappings
XKeymapEvent (3x)     - KeymapNotify event structure
XModifierKeymap (3x) - manipulate keyboard encoding and keyboard encoding
xmodmap (1x)          - utility for modifying keymaps and pointer button ma
XQueryKeymap (3x)     - manipulate keyboard settings and keyboard control s


[dana@granada ˜]$ apropos keymap | grep mod
xmodmap (1x)          - utility for modifying keymaps and pointer button ma
[dana@granada ˜]$
```

which shell am I using now?

```
[dana@granada ~]$ ksh
$ ps -p $$ |  tail -1| awk '{ print $4 }'
ksh
$ echo $SHELL
/bin/tcsh
$
```

How did you pronounce that??

```
$ ps -p  $$
  PID TTY          TIME CMD
 2310 pts/3    00:00:00 ksh
$  ps -p $$| tail -1
 2310 pts/3    00:00:00 ksh
$  ps -p $$| tail -1 |awk '{ print $4 }'
ksh
$
```

Reading File contents

- *lpr* will print the whole file to the printer.
- *cat* will print (to screen) the whole file.
- *zcat* will do the same for gzipped files.
- *more* and *less* will show the contents of the file by pages, with limited ability of searching and scrolling it.

Heads or tails?

- *head* and *tail* will show the ends of the file.

```
[dana@granada ˜]$ head -1 dummy
a b a
dana@granada:˜$ tail -2 dummy
a c

dana@granada:˜$
```

- Use *tail -f* to watch the end of a file which is being updated. For example:

```
tail -f /var/log/messages
```

Standard Input, Output, Error

- The standard input (0) is usually the keyboard
- the standard output (1) is usually the terminal screen
- The standard error is (2) usually also the terminal screen

All this can be changed.

Output Redirection

In tcsh and bash:

```
ls > tmp
```

This means the same as writing in bash:

```
ls 1> tmp
```

Appending in tcsh and bash:

```
ls >> tmp
```

Error and Output Redirection

In tcsh:

```
ls >& tmp
```

In bash:

```
ls 2>&1 > tmp
```

Example - Error redirection in bash:

```
dana@granada:~$ ls kuku kukiya 2>tmp
kukiya
dana@granada:~$ ls kuku kukiya
ls: kuku: No such file or directory
kukiya
dana@granada:~$ cat tmp
ls: kuku: No such file or directory
dana@granada:~$
```

Command lists: list , ||

- ; - perform a list of tasks.

```
dana@granada:~$ ls a*; ls d*
a
d  dummy
dana@granada:~$
```

- || - perform the next only if the previous commands failed:

```
dana@granada:~$ ls a* || ls d*
a
dana@granada:~$
```

Command lists, &&, subshell

- && - perfrom the next in the list only if the previous commands succeded:

```
dana@granada:~$ ./configure && make && make install
```

Or, in order to prepare and test this lecture:

```
[ladypine@granada shell]$ latex shell && dvips -G0 -Ppdf
 shell.dvi && ps2pdf shell.ps && xpdf shell.pdf
```

- () - a subshell. parameters do not take effect on the outside.

```
dana@granada:~$ export animal="king" ;
dana@granada:~$ (export animal="lion"; echo $animal); echo $animal
lion
king
dana@granada:~$
```

The ground we work on

A process can run in the *foreground* or in the *background*. When in the foreground, no other command can be dealt with until the command returns. It can be moved to the background:

```
dana@granada:~$ sleep 300
Talk to me! Please, respond?
Maybe I will move you to the background by typing ctrl z?
[1]+  Stopped                 sleep 300
dana@granada:~$ bg
[1]+ sleep 300 &
dana@granada:~$ jobs
[1]+  Running                 sleep 300 &
```

What shall we do with a foreground process?

It can also be killed:

```
dana@granada:~$ sleep 500
[1]+  Stopped                 sleep 500
dana@granada:~$ bg
[1]+ sleep 500 &
dana@granada:~$ sleep 400
I will now kill the process in the foreground with ctrl c

dana@granada:~$ jobs
[1]+  Running                 sleep 500 &
dana@granada:~$
```

Behind the Scenes - More Job Control

- Tasks can be sent in the background to begin with:

```
dana@granada:~$ sleep 4&
[1] 12175
dana@granada:~$ fg
sleep 4
dana@granada:~$
```

- Tasks can be *nice* to begin with, or made nicer in due time using *renice*.

Special Variables

- ˜ , $HOME, ˜dana - home directory for the user (or for dana, in this case)
- $$ - the shell's process ID
- ! - in bash - the process ID of the most recently executed background (asynchronous) command.
- Positional Variables: $1, $2 ...

! also uses to repeat a command which starts with a letter combination:

```
[dana@granada ˜]$ ls -lt a*
-rw-r--r--    1 dana     dana                6 2004-02-28 09:48 a
[dana@granada ˜]$ ls -la a*
-rw-r--r--    1 dana     dana                6 2004-02-28 09:48 a
[dana@granada ˜]$ !l
ls -la a*
-rw-r--r--    1 dana     dana                6 2004-02-28 09:48 a
```

history

```
[dana@granada ~]$ history
      8  19:35   ls -lt a*
      9  19:36   ls -la a*
     10  19:36   ls -la a*
     11  19:36   history
```

PATH

The PATH is the list of directories that are searched for when an application is requested.

```
dana@granada:~$ echo $PATH
/usr/local/sbin:/usr/sbin:/sbin:/usr/local/bin:/
/usr/bin/X11:/usr/games:/home/dana/bin:/home/dar
/home/dana/dl/rpm/splint-3.1.1/bin
```

Adding dot '.' in the path is dangerous for two reasons:

- Security: what if somebody placed an exeutable in your directory, called ls, and it hides all other changes from you?
- Proper functioning. what if you created a program called test, and it comes first in the path? then /usr/bin/test will be ignored.

If you do decide to add . in your path, do it in the end of the $PATH, to minimize mistakes.

Everyone has it and I don't!?

If something is installed on the system, but not for you, there may be several reasons:

- The program is installed, but it is not in your $PATH. Add it to your $PATH or add an *alias* to find the program.
- The man page is installed, but you cannot find it. Correct your $MANPATH.
- You are not sure what to add to your path. You are not even sure if it is installed at all. Use *locate* to find traces of the mysterious program.
- It was installed, but locate does not find it. Update *locatedb* using *updatedb*, or wait for it to be updated - this (usually happens|should happen) at night.
- In the meantime, or from files not covered by locate, use *find*.

find

```
[ladypine@granada ~]$ find . -name haifux -print
./haifux
[ladypine@vipe ~]find . -name 'linux*' -print
./public_html/linux4me.html
./public_html/linux4me_present.html
```

Remote Shell

- Secure: ssh, putty (ssh client for Windows)
- Insecure: rsh, rlogin,telnet

Display from a remote host

- **Your** computer as an Xserver. The X server is the computer which gives X services. Even if the “real” server is a fast computer which provides CPU services, mail services, etc.
- Check the display on a local machine for two users. ladypine owns the console, and dana does not:

```
[ladypine@granada ~]$ echo $DISPLAY
:0.0
[ladypine@granada ~]$ su - dana
Password:
[dana@granada ~]$ echo $DISPLAY
DISPLAY: Undefined variable.
```

Setting the display

- Set the display on the terminal using the ip or domain name of the computer you are sitting at:
  - In bash:

    ```
    export DISPLAY=granada.merseine.nu:0.0
    ```
  - In tcsh:

    ```
    setenv DISPLAY granada.merseine.nu:0.0
    ```
- Note the $ before the name of the variable when it is evaluated.

Allow X forwarding

Allowing X forwarding is done on behalf of the Xserver - the computer that is about to let others take over its screen.

- Allow a terminal on vipe to use my x server:

  ```
  xhost +vipe.technion.ac.il
  ```

- Open a secure connection to a remote host, asking it to display graphics on the current terminal as a Xserver:

  ```
  ssh -X
  ```

- Check the display:

  ```
  xeyes
  ```

Environment Variables

- Environment variables are shell variables which are passed on to child processes.
- To find all environment variables use *env* (without paramaters).
- In tcsh also setenv (without paramaters).
- Example:

```
dana@granada:~$ env
HOST=granada
SHELL=/bin/tcsh
LC_ALL=he_IL
MAIL=/var/mail/dana
PATH=/usr/local/sbin:/usr/sbin:/sbin:/usr/local/sbin:/usr/sbin
PWD=/home/dana
HOME=/home/dana
LOGNAME=dana
```

Gluing files together: cat,paste

```
[dana@granada ~]$ cat a
a1
a2
[dana@granada ~]$ cat b
b1
b2
[dana@granada ~]$ cat a b
a1
a2
b1
b2
[dana@granada ~]$ paste a b
a1      b1
a2      b2
```

tac, sort

```
[dana@granada ~]$ tac a
a2
a1
dana@granada:~$ cat d
a1      4
a2      0
b       3
d       9
dana@granada:~$ sort -k 2 d
a2      0
b       3
a1      4
d       9
dana@granada:~$
```

Differing files

- *diff*. Useful keys: -B to ignore blanks, -u for unified format.
- *patch*. Apply a patch in the format given by diff -u.
- *cmp*. Just tell me if they differ
- *zcmp*. For gzipped files.

Non Interactive Editors

- awk = "Aho Weinberger and Kernighan", gawk.
- Perl = "Practical Extraction and Report Language" with the -e switch.
- sed = Stream Editor

Perl -e

```
[dana@granada ~]$ perl -e '$i="Dana Nama\n"; pri
                  $i=~s/N/K/; print $i;'
Dana Nama
Dana Kama
```

sed

sed is useful for changing files automatically by a set of instructions. You can also describe it as a filter.

```
$sed ’s/to-be-replaced/replaced/g’ dummyfile
```

For example:

```
[dana@granada ˜]$ more dummy
a b a
a c
[dana@granada ˜]$ sed ’s/a/A/’ dummy
A b a
A c
[dana@granada ˜]$ sed ’s/a/A/g’ dummy
A b A
A c
[dana@granada ˜]$ more dummy
a b a
a c
```

Editing in the terminal

- vi, vim
- xemacs -mw (or if the DISPLAY is not set)
- pico

Textual clients

- nap - Linux napster client
- lynx - textual browser (show accessing a journal and printing it or sending it.
- BitchX- irc client
- mutt, pine - mail clients.

Lynx - a textual html browser

```
          Haifux - Haifa Linux Club - What is the Haifa Linux Club? (p1

   Haifux Logo
     * Where do we meet?
     * Upcoming Lectures
     * Mailing Lists
     * Give a Lecture
     * Events
     * Projects
     * Logo
     * Israeli Linux Links
     * Israeli Linux HOWTO-s
     * Linux Links
     * Site Code
    -- press space for next page --
  Arrow keys: Up and Down to move.  Right to follow a link; Left to go bac
 H)elp O)ptions P)rint G)o M)ain screen Q)uit /=search [delete]=history li
```

Pack to go

- uuencode and uudecode
- gzip and gunzip
- dos2unix and unix2dos
- convert

Having a split personality

- *logname* - the name of the logged in user.
- *whoami* - the name of the user atached to the current process

```
[ladypine@granada ~]$ su dana
Password:
[dana@granada ladypine]$ whoami
dana
[dana@granada ladypine]$ logname
ladypine
```

- *last* - who logged in lately
- *who* - who is currently logged in

```
[dana@granada ~]$ who
muli     pts/1        Feb 28 17:52 (alhambra)
ladypine :0           Feb 20 19:26
ladypine pts/3        Feb 22 18:31 (:0.0)
```

References used for this lecture

- GNU's bash
  http://www.gnu.org/software/bash/bash.html
- Linux Documentation project
  http://www.tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html
- Advanced bash programing
  http://www.tldp.org/LDP/abs/html/
- Working more productively with bash 2.x by Ian Macdonald http://www.caliban.org/bash/
- Why not use csh for prgramming?
  http://www.etext.org/Quartz/computer/unix/csh.harmful.gz
- What does some strange unix command name stand for? http://www.faqs.org/faqs/unix-faq/faq/part1/section-3.html

More references

- Learning the bash Shell, 2nd Edition by Cameron Newham, Bill Rosenblatt

  `http://www.amazon.com/exec/obidos/ASIN/156592`
  `calibanorg-20/102-5966084-5605729?creative=12`

- What’s up’s Hebrew shell guide

  `http://whatsup.org.il/modules.php?op=modload&`
  `&myfaq=yes&id_cat=50&parent_id=0`