Lezione 8

Bioinformatica

Mauro Ceccanti[‡] e Alberto Paoluzzi[†]

[†]Dip. Informatica e Automazione – Università "Roma Tre" [‡]Dip. Medicina Clinica – Università "La Sapienza"

Esercitazione

Introduzione al linguaggio di shell

Λ

Sommario

Esercitazione

Introduzione al linguaggio di shell

Shell command language

- A Unix shell is a command-line interpreter (see shell) and script host that provides a traditional user interface for the Unix operating system and for Unix-like systems.
- The most generic sense of the term shell means any program that users employ to type commands.
- In the Unix operating system users may select which shell to use for interactive sessions.
- Many shells created for other operating systems (e.g. DOS for Windows) offer rough equivalents to Unix shell functionality.



Shell command languages

Unix shells can be broadly divided into Bourne-like and C shell-like

Bourne shell compatible

- sh Bourne shell Written by Steve Bourne, while at Bell Labs. First distributed with Version 7 Unix, circa 1978,
- bash Bourne-Again shell Written as part of the GNU project to provide a superset of Bourne Shell functionality.
- zsh Z shell considered as the most complete shell: it is the closest thing that exists to a superset of sh, ash, bash, csh, ksh, and tcsh.

C shell compatible

- csh C shell Written by Bill Joy, while at the University of California, Berkeley. First distributed with BSD, circa 1979.
- tcsh Tenex shell It is essentially the C shell with programmable command line completion and command-line editing.

Introduction to shell command language

First Unix commands

- 1 <prompt>\$ ls
- 2 Desktop Movies Sites lib
- 3 Documents Music System libexec
- 4 Downloads Pictures bin scipy
- 5 Library Public ebooks share
- 6 <prompt>\$ pwd
- 7 /Users/paoluzzi
- 8 <prompt>\$ ls Users
- 9 <prompt>\$ ls /Users
- 10 Shared paoluzzi
- 11 <prompt>\$ ls /
- 12 Applications Volumes net
- 13
- 14 <prompt>\$ cd /
- 15 . . .

1 1s is a command to list files in Unix and Unix-like operating systems

- 6 pwd short for print working directory
- 9 1s /Users absolute path of a directory
- 14 cd / stands for change directory to root (directory)



a typical prompt to the user is structured as <host>:<path> <account>\$, where the char tilde (~) stands for the user's home directory:

1 baruc3:~ paoluzzi\$

The Bourne shell is immediately recognized when active by its characteristic default command line prompt character, the dollar sign (\$). The default for new Mac OS X accounts is bash.

- 1 <prompt>\$ echo \$SHELL
- 2 /bin/bash

Λ

A command is followed by the shell's answer on the following row. No answer just means that the command was executed with no errors. Conversely, the shell complains quite strongly for errors.

Shell Variables and Environment Variables

These variables cause the shell to work in a particular way

- 1 <prompt>\$ cd Users/
- 2 -bash: cd: Users/: No such file or directory
- 3 <prompt>\$ ls
- 4 Desktop Movies Sites lib
- 5 Documents Music System libexec
- 6 Downloads Pictures bin scipy
- 7 Library Public ebooks share
- 8 <prompt>\$ echo \$PATH
- 9 /opt/local/bin:/opt/local/sbin:/usr/bin:/bin:/usr/sbin:/ sbin:/usr/local/bin:/usr/X11/bin:/usr/local/bin:/ Users/paoluzzi/bin
- 10 <prompt>\$ cd .
- 11 <prompt>\$ cd ..
- 12 <prompt>\$ pwd
- 13 /Users
 - 8 echo shows the contents (\$) of the shell variable PATH searched for executing programs (including shell commands). Paths are separated by colon ":" punctuation mark

Λ

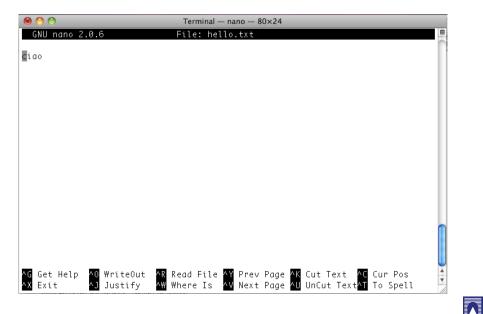
Change directory

```
prompt>$ cd ...
1
   <prompt>$ cd Volumes/
2
   <prompt>$ ls
3
   Macintosh HD
4
   prompt>$ ls /bin/ls
5
   /bin/ls*
6
   prompt>$ ls /bin/
7
   [ df launchctl pwd tcsh bash domainname link rcp test
8
   cat echo ln rm unlink chmod ed ls rmdir wait4path
9
   cp expr mkdir sh zsh csh hostname mv sleep
10
   date kill pax stty dd ksh ps sync
11
   <prompt>$ cd
12
13
   <prompt>$ ls
14
   Desktop Movies Sites ebooks share
15
   . . . . . . . . .
     1 cd to .. (parent directory)
     5 no results for
```

- 6 OK
- 11 cd without parameters changes the directory to the user's home

Nano screen editing

Useful commend menus. Easy to use



Output redirection and text editing

- <prompt>\$ echo "ciao"
- ciao 2
- <prompt>\$ echo "ciao" > hello.txt 3
- <prompt>\$ ls 4
- <prompt>\$ cat hello.txt 5
- ciao 6

- <prompt>\$ vi hello.txt 7
- <prompt>\$ emacs hello.txt 8
- <prompt>\$ nano hello.txt 9

1 echo of the string at console

- 3 output of echo command redirected to the hello.txt file
- 5 cat (concatenate) the contents hello.txt file on the console
- 7 editor vi opens the hello.txt file (:q<enter> to quit)
- 8 editor emacs opens hello.txt (<cntl>x<cntl>c to close)
- 9 editor nano opens hello.txt
- (uses menus: ^ stands for <cntl>)

Copy command

2

3

4

5

6 7

13

14

<prompt>\$ cp hello.txt hello2.txt <prompt>\$ cp hello.txt hello3.txt <prompt>\$ cp hello.txt hello4.txt <prompt>\$ cp hello.txt hello5.txt <prompt>\$ ls Desktop Music bin hello3.txt scipy Documents Pictures cd hello4.txt share Downloads Public ebooks hello5.txt Library Sites hello.txt lib 9 Movies System hello2.txt libexec 10 <prompt>\$ cat hello.txt hello2.txt hello3.txt 11 ciao 12 ciao ciao

1 copy <input file name> <output file name>

- 5 the working directory now contains the new files
- 11 cat concatenates several input files

Move command

the \mathbf{mv} command just changes the file name. It is used to rename files and directories

- 1 <prompt>\$ mv hello2.txt hello21.txt
- 2 <prompt>\$ mv hello3.txt hello31.txt
- 3 <prompt>\$ mv hello4.txt hello41.txt
- 4 <prompt>\$ mv hello5.txt hello51.txt
- 5 <prompt>\$ ls
- 6 Desktop Music bin hello31.txt scipy
- 7 Documents Pictures cd hello41.txt share
- 8 Downloads Public ebooks hello51.txt
- 9 Library Sites hello.txt lib
- 10 Movies System hello21.txt libexec
- 11 <prompt>\$ cat hello.txt hello21.txt hello31.txt
- 12 ciao
- 13 ciao
- 14 ciao

1 move <input file name> <output file name>

Permissions and chmod

chmod is short for change mode. When executed, it can change file system modes of files and directories: \$ chmod <references><operator><modes> file1 ...

```
<prompt>$ chmod u-r hello.txt
   <prompt>$ ls -1 hello.txt
2
   --w-r--r-- 1 paoluzzi staff 5 Nov 9 19:14 hello.txt
3
   <prompt>$ chmod u+r hello.txt
4
   <prompt>$ ls -1 hello.txt
5
   -rw-r--r-- 1 paoluzzi staff 5 Nov 9 19:14 hello.txt
6
   <prompt>$ cat hello.txt
7
   cat: hello.txt: Permission denied
8
   <prompt>$ chmod a-r hello.txt
9
   <prompt>$ ls -1 hello.txt
10
   --w----- 1 paoluzzi staff 5 Nov 9 19:14 hello.txt
11
   <prompt>$ chmod a+r hello.txt
12
13
   -rw-r--r-- 1 paoluzzi staff 5 Nov 9 19:14 hello.txt
```

Iteration

using the for cycle, the "; " command terminator, and a shell variable i

 A number of characters are interpreted by the Unix shell before any other action takes place. These characters are known as wildcard characters. Usually these characters are used in place of filenames or directory names.

1	<prompt>\$ for i in *.txt; do ls \$i; done</prompt>
2	hello.txt
3	hello2.txt
4	hello3.txt
5	hello4.txt
6	hello5.txt
7	<prompt>\$ ls -l hello*</prompt>
8	w-rr 1 paoluzzi staff 5 Nov 9 19:14 hello.txt
9	-rw-rr 1 paoluzzi staff 5 Nov 9 19:17 hello21.txt
10	-rw-rr 1 paoluzzi staff 5 Nov 9 19:17 hello31.txt
11	-rw-rr 1 paoluzzi staff 5 Nov 9 19:17 hello41.txt
12	-rw-rr 1 paoluzzi staff 5 Nov 9 19:17 hello51.txt

1 Notice the different use of i and \$i.

7 An asterisk matches any number of characters in a filename, including none

Executable script

The first line (bang command #!) tells the shell where to find the program to interpret the file



modes r (read) | w (write) | x (execute)

references u (user) | g (group) | o (others) | a (all)

operator + (add) | - (remove) | = (no change)

 $\overline{\Lambda}$

Executing the script

assign execution permission & correct the path of the sh command in a.sh file

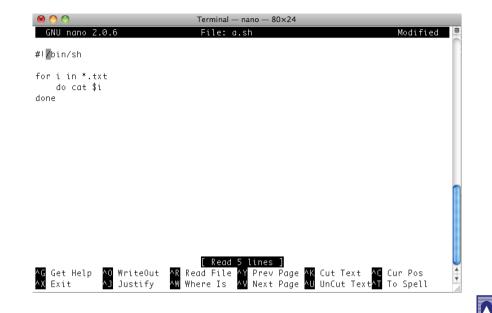
1 \$ nano a.sh 2 \$ ls -l a.sh -rw-r--r-- 1 paoluzzi staff 50 Nov 10 09:43 a.sh 3 \$ a.sh 4 -bash: a.sh: command **not** found 5 \$ pwd 6 /Users/paoluzzi 7 \$ /Users/paoluzzi/a.sh 8 -bash: /Users/paoluzzi/a.sh: Permission denied 9 \$ chmod a+x a.sh 10 \$ ls -l a.sh 11 -rwxr-xr-x 1 paoluzzi staff 50 Nov 10 09:43 a.sh 12 \$ /Users/paoluzzi/a.sh 13 -bash: /Users/paoluzzi/a.sh: /usr/bin/sh: bad 14 interpreter: No such file **or** directory 15 \$./a.sh 16 -bash: ./a.sh: /usr/bin/sh: bad interpreter: No such file **or** directory \$ which sh 17 /bin/sh 18

Executing the script

assign execution permission & correct the path of the sh command in a.sh file

Executable script

correct the path of the sh program and save the file



Executing the script

REMARK: launch the script with a path

- 1 \$ a.sh
- 2 \$./a.sh
- 3 ciao
- 4 ciao
- 5 ciao
- 6 ciao
- 7 ciao
 - 1 no effect, even if the file exists in the current directory, and is provided with permission for execution.
 - 2 it executes if launched from the current directory, because now the shell knows where to find it

Executable programs are searched in the directories listed in the \mathtt{PATH} variable. The current directory (.) and its father (..) are not included by default for security reasons.

Λ



Executing on a remote machine

Using a secure shell command **ssh**

- 1 baruc3:~ paoluzzi\$ ssh paoluzzi@plm.dia.uniroma3.it
- 2 Password: 3
- 4 Last login: Fri May 22 12:29:46 2009 from authentication .uniroma3.it
- 5
- 6 paoluzzi@plm:~\$
- 7 paoluzzi@plm:~\$ ls
- 8 Desktop/ Music/ download/ local/ tower-last/
- 9
- 10 paoluzzi@plm:~\$ exit
- 11 logout
- 12 Connection to plm.dia.uniroma3.it closed by remote host.
- 13 Connection to plm.dia.uniroma3.it closed.
 - 1 <prompt>\$ **ssh** <account>@<remotehost>
 - 2 Of course you need an account on the remote host
 - 6 Notice the change in the user prompt

Fundamental programming in the Bourne again shell (bash)

for a professional introduction to bash shell programming, see:

Bash by example, Part 1 Bash by example, Part 2 Bash by example, Part 3

Λ

