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

Sommario

Esercitazione

Introduzione al linguaggio di shell

Shell command language

Introduction

- 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 language

Introduction

- 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 language

Introduction

- 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 language

Introduction

- 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.

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.

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.

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.

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.

Unix shells are Bourne-like or C shell-like

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

```
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.

```
<prompt>$ echo $SHELL
/bin/bash
```

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.

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 **`ls`** is a command to list files in Unix and Unix-like operating systems

6 **`pwd`** short for print working directory

9 **`ls /Users`** absolute path of a directory

14 **`cd /`** stands for change directory to root (directory)

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 **`ls`** is a command to list files in Unix and Unix-like operating systems

6 **`pwd`** short for print working directory

9 **`ls /Users`** absolute path of a directory

14 **`cd /`** stands for change directory to root (directory)

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 **`ls`** is a command to list files in Unix and Unix-like operating systems

6 **`pwd`** short for print working directory

9 **`ls /Users`** absolute path of a directory

14 **`cd /`** stands for change directory to root (directory)

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 **`ls`** is a command to list files in Unix and Unix-like operating systems

6 **`pwd`** short for print working directory

9 **`ls /Users`** absolute path of a directory

14 **`cd /`** stands for change directory to root (directory)

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

Change directory

```
1  <prompt>$ cd ..
2  <prompt>$ cd Volumes/
3  <prompt>$ ls
4  Macintosh HD
5  <prompt>$ ls /bin/ls
6  /bin/ls*
7  <prompt>$ ls /bin/
8  [ df launchctl pwd tcsh bash domainname link rcp test
9  cat echo ln rm unlink chmod ed ls rmdir wait4path
10 cp expr mkdir sh zsh csh hostname mv sleep
11 date kill pax stty dd ksh ps sync
12 <prompt>$ cd
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

Change directory

```
1  <prompt>$ cd ..
2  <prompt>$ cd Volumes/
3  <prompt>$ ls
4  Macintosh HD
5  <prompt>$ ls /bin/ls
6  /bin/ls*
7  <prompt>$ ls /bin/
8  [ df launchctl pwd tcsh bash domainname link rcp test
9  cat echo ln rm unlink chmod ed ls rmdir wait4path
10 cp expr mkdir sh zsh csh hostname mv sleep
11 date kill pax stty dd ksh ps sync
12 <prompt>$ cd
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

Change directory

```
1  <prompt>$ cd ..
2  <prompt>$ cd Volumes/
3  <prompt>$ ls
4  Macintosh HD
5  <prompt>$ ls /bin/ls
6  /bin/ls*
7  <prompt>$ ls /bin/
8  [ df launchctl pwd tcsh bash domainname link rcp test
9  cat echo ln rm unlink chmod ed ls rmdir wait4path
10 cp expr mkdir sh zsh csh hostname mv sleep
11 date kill pax stty dd ksh ps sync
12 <prompt>$ cd
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

Change directory

```
1  <prompt>$ cd ..
2  <prompt>$ cd Volumes/
3  <prompt>$ ls
4  Macintosh HD
5  <prompt>$ ls /bin/ls
6  /bin/ls*
7  <prompt>$ ls /bin/
8  [ df launchctl pwd tcsh bash domainname link rcp test
9  cat echo ln rm unlink chmod ed ls rmdir wait4path
10 cp expr mkdir sh zsh csh hostname mv sleep
11 date kill pax stty dd ksh ps sync
12 <prompt>$ cd
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

Output redirection and text editing

```
1 <prompt>$ echo "ciao"
2 ciao
3 <prompt>$ echo "ciao" > hello.txt
4 <prompt>$ ls
5 <prompt>$ cat hello.txt
6 ciao
7 <prompt>$ vi hello.txt
8 <prompt>$ emacs hello.txt
9 <prompt>$ nano hello.txt
```

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>)

Output redirection and text editing

```
1 <prompt>$ echo "ciao"
2 ciao
3 <prompt>$ echo "ciao" > hello.txt
4 <prompt>$ ls
5 <prompt>$ cat hello.txt
6 ciao
7 <prompt>$ vi hello.txt
8 <prompt>$ emacs hello.txt
9 <prompt>$ nano hello.txt
```

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>)

Output redirection and text editing

```
1 <prompt>$ echo "ciao"
2 ciao
3 <prompt>$ echo "ciao" > hello.txt
4 <prompt>$ ls
5 <prompt>$ cat hello.txt
6 ciao
7 <prompt>$ vi hello.txt
8 <prompt>$ emacs hello.txt
9 <prompt>$ nano hello.txt
```

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>)

Output redirection and text editing

```
1 <prompt>$ echo "ciao"
2 ciao
3 <prompt>$ echo "ciao" > hello.txt
4 <prompt>$ ls
5 <prompt>$ cat hello.txt
6 ciao
7 <prompt>$ vi hello.txt
8 <prompt>$ emacs hello.txt
9 <prompt>$ nano hello.txt
```

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>)

Output redirection and text editing

```
1 <prompt>$ echo "ciao"
2 ciao
3 <prompt>$ echo "ciao" > hello.txt
4 <prompt>$ ls
5 <prompt>$ cat hello.txt
6 ciao
7 <prompt>$ vi hello.txt
8 <prompt>$ emacs hello.txt
9 <prompt>$ nano hello.txt
```

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>)

Output redirection and text editing

```
1 <prompt>$ echo "ciao"
2 ciao
3 <prompt>$ echo "ciao" > hello.txt
4 <prompt>$ ls
5 <prompt>$ cat hello.txt
6 ciao
7 <prompt>$ vi hello.txt
8 <prompt>$ emacs hello.txt
9 <prompt>$ nano hello.txt
```

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>)

Nano screen editing

Useful commend menus. Easy to use

```
Terminal — nano — 80×24
  GNU nano 2.0.6              File: hello.txt

ciao

^G Get Help   ^O WriteOut   ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit       ^J Justify    ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Copy command

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

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

5 the working directory now contains the new files

11 `cat` concatenates several input files

Copy command

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

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

5 the working directory now contains the new files

11 `cat` concatenates several input files

Copy command

```
1  <prompt>$ cp hello.txt hello2.txt
2  <prompt>$ cp hello.txt hello3.txt
3  <prompt>$ cp hello.txt hello4.txt
4  <prompt>$ cp hello.txt hello5.txt
5  <prompt>$ ls
6  Desktop Music bin hello3.txt scipy
7  Documents Pictures cd hello4.txt share
8  Downloads Public ebooks hello5.txt
9  Library Sites hello.txt lib
10 Movies System hello2.txt libexec
11 <prompt>$ cat hello.txt hello2.txt hello3.txt
12 ciao
13 ciao
14 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 **mv** command just changes the file name. It is used to rename files and directories

```
<prompt>$ mv hello2.txt hello21.txt
<prompt>$ mv hello3.txt hello31.txt
<prompt>$ mv hello4.txt hello41.txt
<prompt>$ mv hello5.txt hello51.txt
<prompt>$ ls
Desktop Music bin hello31.txt scipy
Documents Pictures cd hello41.txt share
Downloads Public ebooks hello51.txt
Library Sites hello.txt lib
Movies System hello21.txt libexec
<prompt>$ cat hello.txt hello21.txt hello31.txt
ciao
ciao
ciao
```

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

Iteration

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

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

Iteration

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

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

Iteration

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

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

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 -l hello.txt
--w-r--r--  1 paoluzzi  staff  5 Nov  9 19:14 hello.txt
<prompt>$ chmod u+r hello.txt
<prompt>$ ls -l hello.txt
-rw-r--r--  1 paoluzzi  staff  5 Nov  9 19:14 hello.txt
<prompt>$ cat hello.txt
cat: hello.txt: Permission denied
<prompt>$ chmod a-r hello.txt
<prompt>$ ls -l hello.txt
--w-------  1 paoluzzi  staff  5 Nov  9 19:14 hello.txt
<prompt>$ chmod a+r hello.txt
-rw-r--r--  1 paoluzzi  staff  5 Nov  9 19:14 hello.txt
```

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

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

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

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 -l hello.txt
--w-r--r--  1 paoluzzi  staff  5 Nov  9 19:14 hello.txt
<prompt>$ chmod u+r hello.txt
<prompt>$ ls -l hello.txt
-rw-r--r--  1 paoluzzi  staff  5 Nov  9 19:14 hello.txt
<prompt>$ cat hello.txt
cat: hello.txt: Permission denied
<prompt>$ chmod a-r hello.txt
<prompt>$ ls -l hello.txt
--w-------  1 paoluzzi  staff  5 Nov  9 19:14 hello.txt
<prompt>$ chmod a+r hello.txt
-rw-r--r--  1 paoluzzi  staff  5 Nov  9 19:14 hello.txt
```

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

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

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

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 -l hello.txt
--w-r--r--  1 paoluzzi  staff  5 Nov  9 19:14 hello.txt
<prompt>$ chmod u+r hello.txt
<prompt>$ ls -l hello.txt
-rw-r--r--  1 paoluzzi  staff  5 Nov  9 19:14 hello.txt
<prompt>$ cat hello.txt
cat: hello.txt: Permission denied
<prompt>$ chmod a-r hello.txt
<prompt>$ ls -l hello.txt
--w-------  1 paoluzzi  staff  5 Nov  9 19:14 hello.txt
<prompt>$ chmod a+r hello.txt
-rw-r--r--  1 paoluzzi  staff  5 Nov  9 19:14 hello.txt
```

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

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

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

Executable script

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

```
Terminal — nano — 80×24
  GNU nano 2.0.6                File: a.sh                          Modified

#!/usr/bin/sh

for i in *.txt
    do cat $i
done

^G Get Help   ^O WriteOut   ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit       ^J Justify    ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Executing the script

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

```
$ nano a.sh
$ ls -l a.sh
-rw-r--r--  1 paoluzzi  staff  50 Nov 10 09:43 a.sh
$ a.sh
-bash: a.sh: command not found
$ pwd
/Users/paoluzzi
$ /Users/paoluzzi/a.sh
-bash: /Users/paoluzzi/a.sh: Permission denied
$ chmod a+x a.sh
$ ls -l a.sh
-rwxr-xr-x  1 paoluzzi  staff  50 Nov 10 09:43 a.sh
$ /Users/paoluzzi/a.sh
-bash: /Users/paoluzzi/a.sh: /usr/bin/sh: bad
    interpreter: No such file or directory
$ ./a.sh
-bash: ./a.sh: /usr/bin/sh: bad interpreter: No such
    file or directory
$ which sh
/bin/sh
```

Executable script

correct the path of the sh program and save the file

```
Terminal — nano — 80×24
  GNU nano 2.0.6                File: a.sh                          Modified

#!/bin/sh

for i in *.txt
    do cat $i
done

                              [ Read 5 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Executing the script

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

Executing the script

REMARK: launch the script with a path

```
$ a.sh
$ ./a.sh
ciao
ciao
ciao
ciao
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 `$PATH` variable. The current directory (.) and its father (..) are not included by default for security reasons.

Executing the script

REMARK: launch the script with a path

```
$ a.sh
$ ./a.sh
ciao
ciao
ciao
ciao
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 `$PATH` variable. The current directory (.) and its father (..) are not included by default for security reasons.

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

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

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