

```
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define MAXPAROLA 30
#define MAXRIGA 80

int main(int argc, char *argv[])
{
    int freq[MAXPAROLA]; /* vettore di contatori
delle frequenze delle lunghezze delle parole */
    char riga[MAXRIGA];
    int i, inizio, lunghezza;
    FILE *f;

    for(i=0; i<MAXPAROLA; i++)
        freq[i]=0;

    if(argc != 2)
    {
        fprintf(stderr, "ERRORE, serve un parametro con il nome del file\n");
        exit(1);
    }
    f = fopen(argv[1], "r");
    if(f==NULL)
    {
        fprintf(stderr, "ERRORE, impossibile aprire il file %s\n", argv[1]);
        exit(1);
    }

    while( fgets( riga, MAXRIGA, f ) != NULL )
```



UNIX/Linux Operating System

Shells

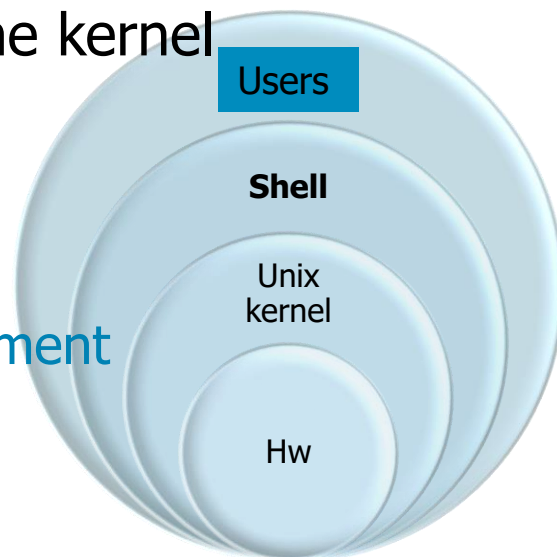
Stefano Quer, Pietro Laface, and Stefano Scanzio
Dipartimento di Automatica e Informatica
Politecnico di Torino

skenz.it/os

stefano.scanzio@polito.it

Introduction to shells

- ❖ The outermost layer of the OS
 - It provides the user interface, which interprets the user commands
 - It was the unique interface before the introduction of graphics servers
- ❖ In Unix, a shell is not part of the kernel
 - It is a normal user process
 - Similar to DOS but more powerful
 - Offers a programming environment "native of the OS"



Introduction to shells

❖ A shell allows

- Submitting commands on command line
 - The shell automatically understands when the command ends and executes it immediately
- Writing shell programs (**scripts**)
 - Storing commands in a script file
 - Script execution by submitting the script file

❖ Writing a script avoids

- Typing complex command sequences repeatedly
- Automating tedious, repetitive and error prone tasks

Introduction to shells

- ❖ Different shells may accept slightly different commands
- ❖ Often **/bin/sh** is a link to the current shell
 - The default shell can be modified
 - `chsh` (change login shell)
 - Version in use
 - `/bin/bash -version`
 - `echo $BASH_VERSION`

tcsh

```
set myVar = "ciao"
```

```
setenv MY_DIR /home/usr/
```

```
if ($str1==$str2) then ... else ...  
endif
```

bash

```
myVar="ciao"
```

```
export MY_VAR=/home/usr/
```

```
if test $str1=$str2 then ... else ... fi  
if [ $str1=$str2 ]; then ... else ... fi
```

shell execution

- ❖ A shell can be activated
 - Automatically at login
 - Nested within another shell
 - As a user program
 - `/bin/tcsh`, `/bin/bash`, ...
- ❖ A shell exit by typing
 - Command `exit`
 - The EOF character (usually Ctrl-d)
 - Exiting an inner shell will return to the outer shell

Introduction to bash

- ❖ At login (and exit) a shell looks for, and executes, some configuration files that contain initialization (or termination) commands
- ❖ **Startup files** differ in
 - **Login files**
 - Shell is executed after authentication in the system (password)
 - **Non-login files**
 - The shell is executed through an icon or system menu

Introduction to bash

- ❖ For each login with password, the shell executes
 - Global scripts
 - `/etc/profile`
 - User scripts (executes the first existing file among)
 - `~/.bash_profile`
 - `~/.bash_login`
 - `~/.profile`
 - There is an error in case of incorrect or unreadable file

Introduction to bash

- ❖ For each login without a password, the shell executes
 - `~/ .bashrc`
 - This file often refers to `~/ .bashrc_profile`
 - It is also the file typically executed in remote login
- ❖ For each logout, the shell executes
 - `~/ .bash_logout`

shell command expansion

- ❖ Some characters have special meaning within the shell
- ❖ bash provide complex substitution mechanisms
 - After dividing the command line into tokens, the shell expands or solves these tokens, i.e., it applies different types of replacement
 - Braces, tilde, variables and parameters, commands, arithmetic expressions, etc.
 - The substitution is complex and takes place with a specific order

Parentheses

❖ Parentheses (), [], {}

- Enclose variables, arithmetic operations, etc.
- In some cases, they are subject to automatic expansion (brace expansion)

```
➤ name=Jean  
➤ echo $namePaul  
  
➤ echo {$name}Paul  
{Jean}Paul  
➤ echo ${name}Paul  
JeanPaul
```

echo: print command

This variable
does not exist

Quoting

❖ "Quoting" means the use of for quotation marks

➤ Quotes ' '

- Variables within quotes are **not expanded**
- They cannot be nested

➤ Double quotes " "

- Variables within double quotes are **expanded**
- They can be nested

➤ Backslash \

- Identifies the escape character, which remove the special meaning of the character that follows it

Examples

```
➤ myVar="A string"
➤ echo $myVar
A string

➤ echo 'v = $myVar'
v = $myVar

➤ echo "v = $myVar"
v = A string

➤ echo \$myVar
$myVar
➤ echo "double quote\""
double quote"
```

Variable usage:
- set without \$
- used with \$

' ... ' →
no expansion

" ... " → expansion

\ cancels the meaning
of the next character,
which becomes a
"meta-character"

Using the output of command

- ❖ The standard output of a command can be **captured** by
 - Enclosing the command in `$(...)`
 - Enclosing the command in **backquotes** `` ``
- ❖ In particular, the output of a command can be stored in a variable

```
➤ d=$(date)
➤ echo $d
➤ Fri Nov 22 10:00:0 \
    CET 2013
➤ d=`date`
...

```

```
➤ out=`cat file.txt`
➤ echo $out
➤ ... file content ...

➤ out=`< file.txt`
➤ echo $out
➤ ... file content ...

```

Command execution

❖ In a shell, a command can be executed

➤ Directly

▪ `cd /home ; ls`

The current shell executes the command; change directory to `/home`; executes `ls`; at the end the working directory is `/home`

➤ Indirectly

▪ `(cd /home ; ls)`

The current shell executes the command **in a subprocess**; change directory to `/home`; executes `ls`; at the end the working directory is **the original directory**

❖ A shell

- Keeps the list of the last submitted commands
 - In bash, the list is stored in file `.bash_history`
 - Stored in the user home directory
- Shell commands allow to reference this list

Command	Meaning
<code>history</code>	Displays the list of the last submitted commands
<code>!n</code>	Executes command number <code>n</code> in the history list
<code>!str</code>	Executes last command beginning by <code>str</code>
<code>^str1^str2</code>	Executes last command replacing <code>str1</code> by <code>str2</code>

Aliasing

❖ In shell you can define new names to existing commands

➤ The **alias** command allows defining these names

No blanks near symbol =

▪ **alias name="string"**

- defines a new alias for "string"

➤ The shell maintains a list of aliases

▪ **alias**

- provides the list of active aliases used in the shell

➤ Old aliases can be deleted

▪ **unalias name**

- Deletes the alias name from the shell

Examples

Existent aliases

➤ **alias**

```
alias egrep='egrep --color=auto'  
alias emacs='emacs -r -geometry 100x36 -fn 9x15 &'  
alias fgrep='fgrep --color=auto'  
alias grep='grep --color=auto'  
alias ls='ls --color=auto'  
alias mx='xdvi -mfmode ljfour:1200'
```

Definition of a new alias

➤ **alias ll= "ls -la"**

➤ **unalias emacs**

➤ **unalias ll**

Deletion of a pre-existing alias
(the eventual command returns
to be what it was)