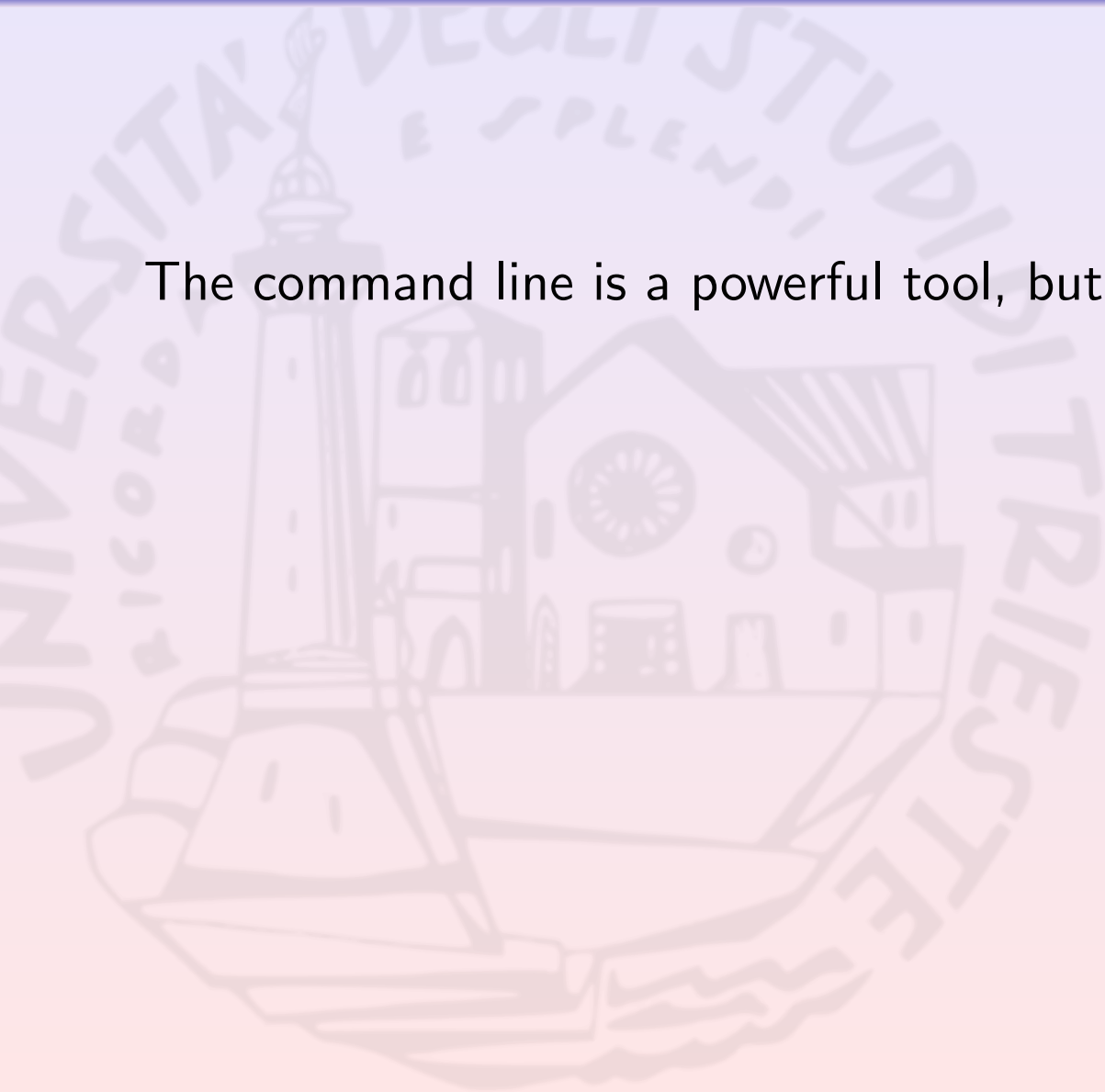
Unit 8

Bash Programming and Git

Alberto Casagrande
Email: acasagrande@units.it

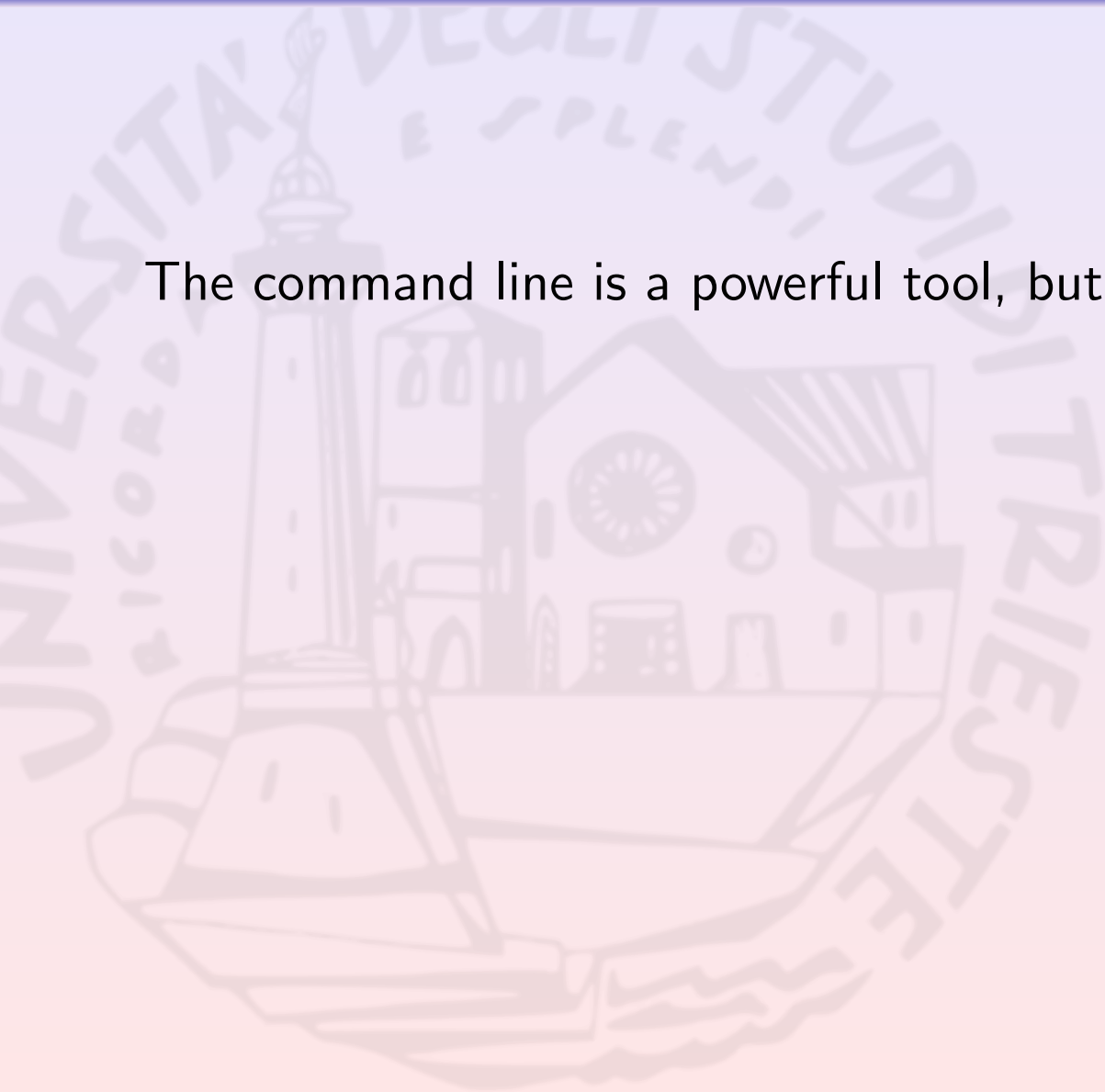
A.Y. 2021/2022

Interactive vs Scripting



The command line is a powerful tool, but error prone

Interactive vs Scripting



The command line is a powerful tool, but error prone

Interactive vs Scripting

The command line is a powerful tool, but error prone

Repeatitive tasks are problematic because of typos and fatigue.

You must be focused along all the interactive process.

Any command provided to a shell can be included into a shell script

A Very Simple BASH Script

```
#!/bin/bash

# This is a comment

A=1    # this is a variable assignment
B=2;   # another one

echo $A" $B test ${B} test"
```

A Very Simple BASH Script (Cont'd)

- save it as `first.sh`
- make it executable

```
al@foo:~> chmod +x first.sh
```

- execute it

```
al@foo:~> ./first.sh
```

The result will be

```
1   2_test
```

A Very Simple BASH Script (Cont'd)

```
#!/bin/bash

# This is a comment

A=1    # this is a variable assignment
B=2;   # another one

echo $A" "$B_test "${B}_test"
```

We got

- the content of `A`
- two empty spaces because `B_test` does not exist
- the content of `B` followed by `_test`

Arithmetic Expressions

Integer arithmetic evaluations are supported through the `$((...))` syntax

```
#!/bin/bash

A=1    # this is a variable assignment
B=2;   # another one

echo ${A}/${B}=$((A/B))
```

Script Parameters

$# stores the number of script parameters

$1, $2, ... store parameters

$@ is the list of the script parameters

```
#!/bin/bash

echo $# $2 $@
```

Conditional Statement

```
#!/bin/bash

if [ $# -ne 2 ]; then
  (>&2 echo "Error: Expected 2 parameters")

  exit 1
fi

echo $(($1 + $2))
exit 0
```

Boolean Expressons

Bash supports many many kind of Boolean expressions

E.g.,

- `[ -a FILE ]` tests whether the file exists
- `[ -x FILE ]` tests whether the file exists and is executable
- `[ -d DIR ]` tests whether the directory exists
- `[ S1 == S2 ]` tests whether the two strings are equal
- `[ v1 -gt v2 ]` tests whether the first value is greater than the second one
- `[ E1 -o E2 ]` performs the non-exclusive disjuntion of the two expressions

Boolean Expressons

Bash supports many many kind of Boolean expressions

E.g.,

- `[ -a FILE ]` tests whether the file exists
- `[ -x FILE ]` tests whether the file exists and is executable
- `[ -d DIR ]` tests whether the directory exists
- `[ S1 == S2 ]` tests whether the two strings are equal
- `[ v1 -gt v2 ]` tests whether the first value is greater than the second one
- `[ E1 -o E2 ]` performs the non-exclusive disjuntion of the two expressions

See online for a full list of them

While-Loop Statement

```
#!/bin/bash

i=0

while [ $i -ne 10 ]; do
  echo $i

  i=$((i+1))
done

exit 0
```

For-Loop Statement

```
#!/bin/bash

for name in "a" "b" "c"; do
  echo $name
done

exit 0
```

Evaluating a Command

Commands can also be evaluated by using the `$(...)` syntax

```
#!/bin/bash

for name in $(ls $1); do
  bn=$(basename ${name} .sh)

  echo "The base name of ${name} is ${bn}"
done

exit 0
```

Functions Can Be Defined Too

```
#!/bin/bash

function test_function() {
    echo "This call has $# parameters"
    echo "Parameter 1 is $1"
    echo "Parameter 2 is $2"

    for i in $@; do
            echo $i
    done
}

test_function $2 $1 $3
```

GIT

It is a distributed version control system

It is meant to keep track of your project revisions (e.g., for bug tracking)

Many developpers can work on the same project at the same time

It can handle many “version” of the same project

Initializing a GIT project

Enter in the project directory and execute `git init`

```
al@foo:~/prj$ ls
program.c
al@foo:~/prj$ git init
Initialized empty Git repository in
   /home/al/prj/.git/
```

The git repository is the place where all the revisions and versions of the project are stored.

It is automatically handled by git.

Repository Status

None of the files are tracked by the git repository yet.

We can see the repository status by executing `git status`

```
al@foo:~/prj$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will
be committed)

        program.c

nothing added to commit but untracked files present
(use "git add" to track)
```

Adding a file

Files can be added to the project by using `git add`

```
al@foo:~/prj$ git add program.c
al@foo:~/prj$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   program.c
```

The changes are now planned to be included into next revision

But not commited yet to the repository

Commiting a revision

To commit the first revision of your project

```
al@foo:~/prj$ git commit -m "First commit"
[master (root-commit) 62fcde6] First commit
 1 file changed, 8 insertions(+)
 create mode 100644 program.c
```

An the status is now

```
al@foo:~/prj$ git status
On branch master
nothing to commit, working tree clean
```

Changing a repository file

Any file change is reported by `git status`

```
al@foo:~/prj$ git status
Sul branch master
Changes not staged for commit:
  (use "git␣add␣<file>..." to update what will
be committed)
  (use "git␣checkout␣--␣<file>..." to discard
changes in working directory)

        modified:   program.c

no changes added to commit (use "git␣add" and/or
"git␣commit␣-a")
```

What about the differences?

The differences between the current revision of the file and the last repository revision can be seen as follows

```
al@foo:~/prj$ git diff
diff --git a/program.c b/program.c
index 9dfdaf4..f8ef5d6 100644
--- a/program.c
+++ b/program.c
@@ -2,7 +2,7 @@

 int main(int argv, char *argv[])
 {
-   printf("Hello, world!\n");
+   printf("Hello, world!!!\n");

    return 0;
 }
```

Branches

From time to time, having different versions of the same project in development may be usefull

E.g., when adding a feature that potentially breaks the project itself or bug fixing different version of the same software

Branches are branches in the development flow of a project.

Any project have one “main” branch named master

Creating a branch

```
al@foo:~/prj$ git branch longer_string
```

To see all the branches use `git branch` again

```
al@foo:~/prj$ git branch
  longer_string
* master
```

We are still on the master branch

Moving between branches

To move to `longer_string` branch use `git checkout`

```
al@foo:~/prj$ git checkout longer_string
M       program.c
Moved to branch 'longer_string'
git branch
* longer_string
  master
```

All the changed can be commited to this branch as usual

```
al@foo:~/prj$ git commit -a -m "New branch"
[longer_string 26d7695] New branch
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Tracking changes

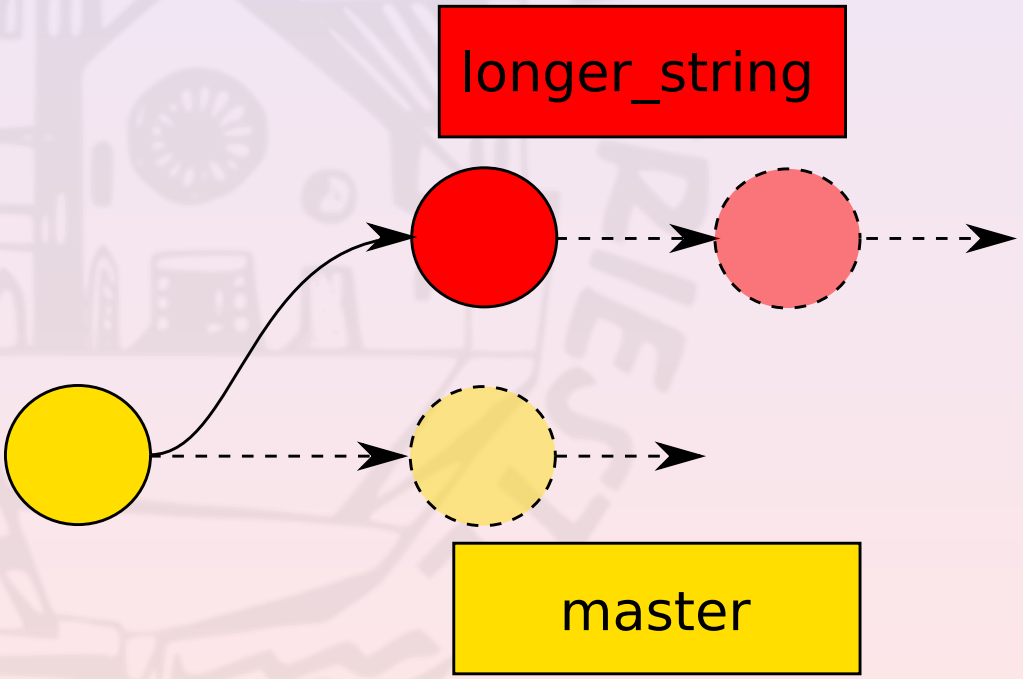
`git diff` can be used to see the differences between branches

```
al@foo:~/prj$ git diff master
diff --git a/program.c b/program.c
index 9dfdaf4..f8ef5d6 100644
--- a/program.c
+++ b/program.c
@@ -2,7 +2,7 @@

 int main(int argv, char *argv[])
 {
-   printf("Hello, world!\n");
+   printf("Hello, world!!!\n");

    return 0;
 }
```

Branches evolutions

The branches evolve independently

Every commit exclusively concerns the active branch



Switching branches

Use `git checkout` to move from a branch to another one

```
al@foo:~/prj$ git branch
* longer_string
  master
al@foo:~/prj$ git checkout master
Switched to branch 'master'
```

Getting branch history

`git log` shows branch history

```
al@foo:~/prj$ git log
commit aa9ed46f...fc2a73 (HEAD -> longer_string)
Author: Alberto Casagrande <al@foo.bar>
Date:   Fri Sep 27 14:20:04 2019 +0200

    longer_string third commit

...
commit ba7f80f2...8e4189
Author: Alberto Casagrande <al@foo.bar>
Date:   Fri Sep 27 14:09:40 2019 +0200

    First commit
```

Reverting to a previous revision

`git checkout` can also be used to revert to a previous revision

```
al@foo:~/prj$ git log
...
commit 1a4619d8595....e70e9ee6
Author: Alberto Casagrande <al@foo.bar>
Date:   Fri Sep 27 14:19:28 2019 +0200

    longer_string second commit
...
al@foo:~/prj$ git checkout 1a4619d
Note: checking out '1a4619d'.

....
```

Branches merging

If you want to merge the changing in two branching since the last merge/split, use `git merge`

```
al@foo:~/prj$ git branch
* longer_string
  master
al@foo:~/prj$ git merge master
Auto-merging program.c
CONFLICT (content): Merge conflict in program.c
Automatic merge failed; fix conflicts and then
commit the result.
```

Branches merging (cont'd)

. . . if needed, fix all the conflicts by editing the files . . .

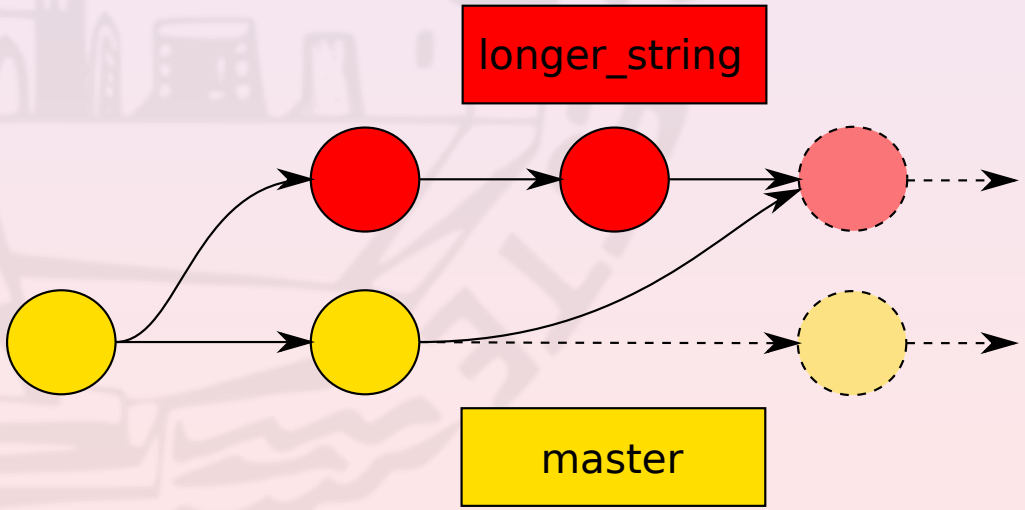
```
#include <stdio.h>
#include <string.h>

#include <math.h>

int main(int argv, char *argv[])
{
<<<<<<< HEAD
  printf("Hello, world!!!\n");
=======
  printf("Hello, world! Boh!\n");
>>>>>>> master

  return 0;
}
```

Branches merging (cont'd 2)

. . . and commit the result

```
al@foo:~/prj$ git commit -m "Merge from master" -a
[longer_string 78f8c6f] Merge from master
```

All the changes done in the merging branch since the last merge/split will be applied to the current branch.



Deleting a branch

A branch can be deleted by using the `git branch` command

```
al@foo:~/prj$ git checkout master
Switched to branch 'master'

al@foo:~/prj$ git branch -D longer_string
Deleted branch longer_string (was 78f8c6f).

git branch
* master
```

Git on a remote server

Your local git repositories can be saved on remote servers (e.g., github.org, gitlab.org, etc.) to:

- avoid data loss
- work on shared project

To this goal, at least a remote should be defined

E.g.

```
al@foo:~/prj$ git remote

al@foo:~/prj$ git remote add github https://github.
org/myuser/prj.git

al@foo:~/prj$ git remote -v
github  https://github.org/myuser/prj.git (fetch)
github  https://github.org/myuser/prj.git (push)
```

Pushing and Pulling

Once remotes have been defined, the last commits on the repository can be pushed on the server by issuing

```
al@foo:~/prj$ git push github
```

None of the commits in the local repository are sent to the remote until the push

Updated version of the repository can be downloaded from the server by writing

```
al@foo:~/prj$ git pull github
```

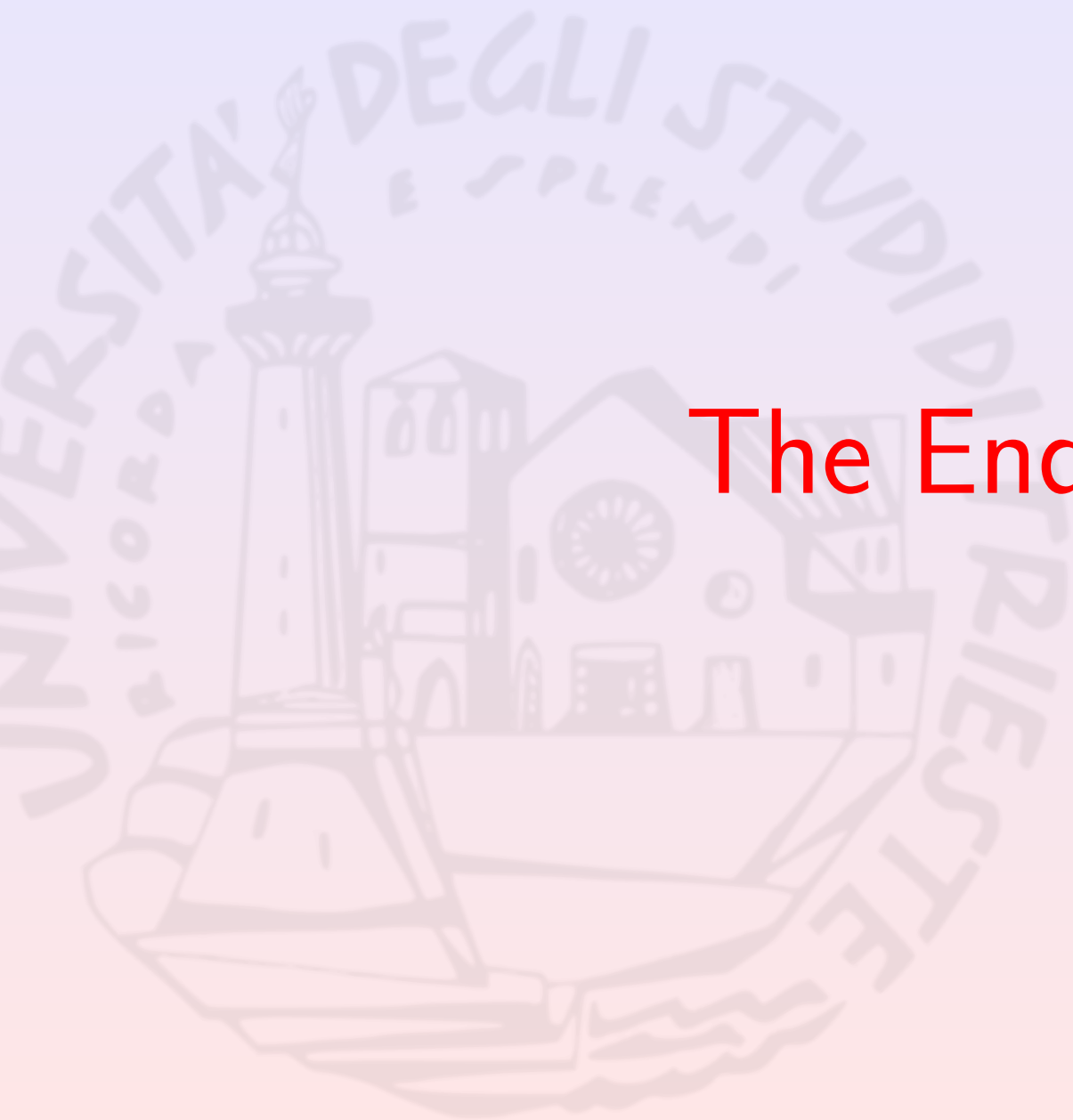
Cloning a repository

Already created repositories can be downloaded from the servers by using `git clone`

```
al@foo:~/prj$ git clone https://github.com/albertoca
sagrande/pyModelChecking.git

Cloning into 'pyModelChecking'...
remote: Enumerating objects: 217, done.
remote: Counting objects: 100% (217/217), done.
remote: Compressing objects: 100% (121/121), done.
remote: Total 483 (delta 129), reused 159 (delta 87)
Receiving objects: 100% (483/483), 177.25 KiB | 495
Resolving deltas: 100% (273/273), done.
```

Coming soon...

- The End



The End