AGH

Methodology and Programming Techniques

Department of Telecommunications, **E&T**

Jarosław Bułat, PhD

kwant@agh.edu.pl

Outline

- Classification of programming languages, syntax, semantics
- Processing of source code
- Second program in C++
  - Basic I/O,
  - variable declarations
  - Arithmetic operations
- GIT - pull, commit, push

Elements of the programming language

AGH

Elements of the programming language

- Syntax
- Semantics
- Data types
- Standard Libraries

Elements of the programming language

- » **Syntax**
- » Semantics
- » Data types
- » Standard Libraries

- – types of available symbols and principles by which we can combine it
- – syntax correct code does not have to be semantically correct
- – how to create commands and expressions
- – eg. form of control instructions
- – correct form of declaration (variable/function/ ...)

Elements of the programming language

- **Syntax**
- Semantics
- Data types
- Standard Libraries

```
// decoded, check next synchro
if (adts_head_idx_+5 < superframe_cifs_){
    if (crc_errors<num_aus)
        adts_head_idx_ += 5;
    else{
        adts_head_idx_ += 4;
    }
}else{
    adts_head_idx_ = 0;
    superframe_cifs_ = 0;
    return;
}
CircshiftBuff(data);
```

Elements of the programming language

- Syntax
- **Semantics**
- Data types
- Standard Libraries

- the precise definition of symbols and their functions in the program
- most often it is a verbal definition (formalisms are impractical)
- some of the semantic errors can be detect during the compilation, while other only in runtime
  - For example, if a name (identifier) is declared before the first use

Elements of the programming language

» Syntax

» **Semantics**

» Data types

» Standard Libraries

– the precise definition of symbols and their functions in the program

– most often it is a verbal definition (formalisms are impractical)

– some of the semantic errors can be detect during the compilation, while other only in runtime

  - For example, if a name (identifier) is declared before the first use

```
if (a = b){
   // something ...
}
```

```
int calculateArea(int width, int height){
   return width + height;
}
```

Elements of the programming language

- » Syntax
- » Semantics
- » **Data types**
- » Standard Libraries

- – types of data we can operate, their properties and allowed operations
- – built-in types (basic) usually:
  - integers (int)
  - floating point numbers (float, double)
  - text strings (char [])

Elements of the programming language

- » Syntax
- » Semantics
- » **Data types**
- » Standard Libraries

- – types of data we can operate, their properties and allowed operations
- – built-in types (basic) usually:
  - integers (int)
  - floating point numbers (float, double)
  - text strings (char [])

```
if ("1" == TRUE ){          // semantic error
}                            // no syntax error
```

Elements of the programming language

- » Syntax
- » Semantics
- » **Data types**
- » Standard Libraries

- – types of data we can operate, their properties and allowed operations
- – built-in types (basic) usually:
  - • integers (int)
  - • floating point numbers (float, double)
  - • text strings (char [])
- – **statically typed**
  - • **explicite**
  - • inference (automatic)
- – **dynamically typed**

Elements of the programming language

» Syntax

» Semantics

» **Data types**

» Standard Libraries

```
// C++
int result = 0;

// result is of the type int
// 1.3 is of the type float

result = 1.8;
// result == 1, still int
```

- types of data we can operate, their properties and allowed operations
- built-in types (basic) usually:
  - integers (int)
  - floating point numbers (float, double)
  - text strings (char [])
- **statically typed**
  - **explicite**
  - inference (automatic)
- **dynamically typed**

Elements of the programming language

- » Syntax
- » Semantics
- » **Data types**
- » Standard Libraries

```
// C++
float result;
int input = 1;

result = input;
// conversion int->float
```

- types of data we can operate, their properties and allowed operations
- built-in types (basic) usually:
  - integers (int)
  - floating point numbers (float, double)
  - text strings (char [])
- **statically typed**
  - **explicite**
  - inference (automatic)
- **dynamically typed**

Elements of the programming language

- » Syntax
- » Semantics
- » **Data types**
- » Standard Libraries

```
# python

result = 1        # int
result = 1.0      # float
result = 'abc'    # str
```

- – types of data we can operate, their properties and allowed operations
- – built-in types (basic) usually:
  - integers (int)
  - floating point numbers (float, double)
  - text strings (char [])
- – **statically typed**
  - **explicite**
  - inference (automatic)
- – **dynamically typed**

Elements of the programming language

- Syntax
- Semantics
- Data types
- **Standard Libraries (and runtime)**

- Usually the basic set of functions / procedures to operate:
  - standard input/output (console)
  - files (storage)
  - operating memory
  - multithreading
  - operations on text strings (text)
  - basic data types + operations on them
- Beginners often treat standard libraries as part of a language **implementation**

C++

C++ is “newer, better C”

C++

- General purpose programming language
- High-performance, direct access to resources ((dis)advantage)
- Multi-platform (hardware/OS)
- **Multi-paradigm** (procedural, object-oriented, generic)
- Enables data abstraction
- Compliance with "C"
- The only language supports virtually all hardware and software platforms
- Old, difficult, dangerous

C++

and
and_eq
asm
auto
bitand
bitor
bool
break
case
catch
char
class
compl
const

const_cast
continue
default
delete
do
double
dynamic_cast
else
enum
explicit
export
extern
false
float

for
friend
goto
if
inline
int
long
mutable
namespace
new
not
not_eq
operator
or

or_eq
private
protected
public
register
reinterpret_cast
return
short
signed
sizeof
static
static_cast
struct
switch

template
this
throw
true
try
typedef
typeid
typename
union
unsigned
using
virtual
void
volatile

wchar_t
while
xor
xor_eq

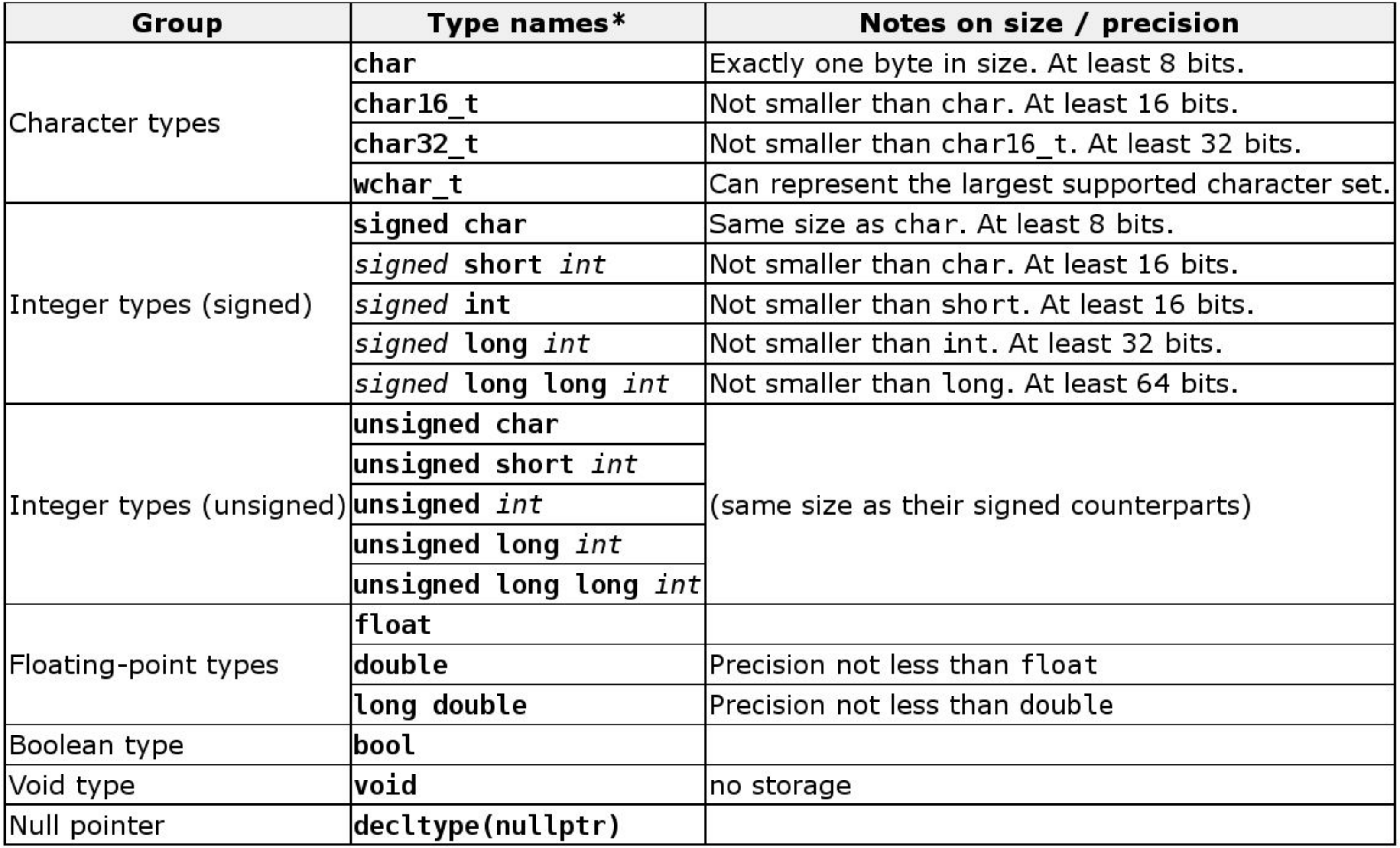
{....}
“\n”
/*....*/
//
< > <= >=
== !=
=
- + / *
<< >>
...

Variable

- **variable** is a "programming design", has:
  - name (label, ID)
  - type
  - value (state)
  - storage location (address, size)
- in the code, we can refer to a variable by name or by location (memory address)
- variable can be read, write
- **in C++ variable must be declared**

Variable types

» **character:** storage of characters, numbers, symbols...	**char**
» **integers:**	**int**
– signed, values: -128 ... 0 ... + 127	signed **int**
– unsigned, values: 0 ... 255	**unsigned int**
» **floating-point**	**float**
– single, double and quad-precision	**double**
» **logic:** bool	**bool**
» **other:** Void, null	
» **ranges of variables in the header file <climits>**	



Group	Type names*	Notes on size / precision
Character types	**char**	Exactly one byte in size. At least 8 bits.
	char16_t	Not smaller than char. At least 16 bits.
	char32_t	Not smaller than char16_t. At least 32 bits.
	wchar_t	Can represent the largest supported character set.
Integer types (signed)	**signed char**	Same size as char. At least 8 bits.
	signed **short** *int*	Not smaller than char. At least 16 bits.
	signed **int**	Not smaller than short. At least 16 bits.
	signed **long** *int*	Not smaller than int. At least 32 bits.
	signed **long long** *int*	Not smaller than long. At least 64 bits.
Integer types (unsigned)	**unsigned char**	(same size as their signed counterparts)
	unsigned short *int*	
	unsigned *int*	
	unsigned long *int*	
	unsigned long long *int*	
Floating-point types	**float**	
	double	Precision not less than float
	long double	Precision not less than double
Boolean type	**bool**	
Void type	**void**	no storage
Null pointer	**decltype(nullptr)**	

http://www.cplusplus.com/doc/tutorial/variables/

Variable declaration

```
#include <iostream>

int main(){
      int a;









}
```

Variable declaration

```
#include <iostream>

int main(){
    int a;                      // declaration
    int b, c, d;                /* declaration */
    float myNumber;







}
```

Variable declaration

```
#include <iostream>

int main(){
      int a;                          // declaration
      int b, c, d;                    /* declaration */
      float myNumber;

      a = 1;                          // assignment
      b = -3;
      c = d = 7;
      /*
       * l-wartosc symbol_przypisania wyrażenie [terminator]
       * l-value assignemnt_symbol expression [ending statement]
       */
}
```

Variable initialization

```
#include <iostream>

int main(){
    int a = 1;          // declaration and initialization
    int b(3);
    int c;
    float myNumber;





}
```

Variable initialization

```
#include <iostream>

int main(){
     int a = 1;          // declaration and initialization
     int b(3);
     int c;
     float myNumber;

     c = -3;
     int d = a + b + c;



}
```

Variable initialization

```
#include <iostream>

int main(){
    int a = 1;          // declaration and initialization
    int b(3);
    int c;
    float myNumber;

    c = -3;
    int d = a + b + c;

    std::cout << "d=" << d << std::endl;
    std::cout << "float=" << myNumber << std::endl;
}
```

cout == console output

Variable initialization

```
#include <iostream>

int main(){
    int a = 1;          // declaration and initialization
    int b(3);
    int c;
    float myNumber;

    int d = a + b + c;
    c = -3;

    std::cout << "d=" << d << std::endl;
    std::cout << "float=" << myNumber << std::endl;
}
```

Variable initialization

```
#include <iostream>

int main(){
    int a = 1;          // declaration and initialization
    int b(3);
    int c;
    float myNumber;

    int d = a + b + c;            ← wrong (very wrong!!!)
    c = -3;

    std::cout << "d=" << d << std::endl;
    std::cout << "float=" << myNumber << std::endl;
}
```

constant variable !@#$%^

```
// #define stala 10
const int stala = 10;



std::cout << stala << std::endl;
```

- » Constant == do not change
- » Need to be initialized

constant variable !@#$%^

```
// #define stala 10
const int stala;



std::cout << stala << std::endl;
```

» Constant == do not change

» Need to be initialized, if not:
error: uninitialized const ‘stala’

constant variable !@#$%^

```
// #define stala 10
const int stala = 10;

stala++;

std::cout << stala << std::endl;
```

- Constant == do not change
- Need to be initialized, if not:
  error: uninitialized const ‘stala’
- An attempt to change “stala” ends with error:
  error: increment of read-only variable ‘stala’
  stala++;

constant variable !@#$%^

```
// #define stala 10
const int stala = 10;

stala++;

std::cout << stala << std::endl;
```

Advantage of “const” over #define, is its TYPE. It means, compiler can perform some optimization and verify type during assignments.

- Constant == do not change
- Need to be initialized, if not:
  error: uninitialized const ‘stala’
- An attempt to change “stala” ends with error:
  error: increment of read-only variable ‘stala’
      stala++;

Can I solve equation: $y = 4-x$ in C++?

NO

- The equations in the mathematical sense can not be implemented in C/C++
- "Equation" will be treated as an assignment according to the scheme:

$$y = 4\text{-}x\ ;$$

l-value assignment_operator expression [statement_terminator]

- The expression is calculated and assigned to a variable (inserted, substituted)
- The expression can be complex: various arithmetic operations, fixed, variable, etc ...

Arithmetic operation

```
 5: int x = 1 + 2 + 3 + 4;          // 10      (std::cout<<x;)
 6: int y = 20 - x;             // -10
 7: y = x * 3;                      // 30(10*3)
 8: float xy = 10 * 0.73;           // 7.3
 9: xy = 10.0 / 7;              // 1.42857
10: int z = x / 3;                  // 1  (10/3)
11: z = x / 6;                  // 1  (10/6==1.6666)
12: z = x % 6;                      // 4  (reszta z dielenia 10/6)
13:                                 // 10==1*6+4
14: unsigned int u = x - y;         // 10-30==4294967276
```

- 6: assignment has no effect
- Division “x/y” on integers results in **round down value**: floor(x/y)
- **xy** is not **x*y**
- Remainder of a division “%” is not defined on float or double
- “unsigned” type do not allows for negative value

Priority of operators

```
 5: int a = 1 + 2 * 3;              // 7
 6: int b = 2 * 3 + 1;              // 7
 7: int c = 2 * (3 + 1);       // 8
 8:
 9: a = 2 - 2 - 2;                  // -2
10: b =(2 - 2) - 2;            // -2
11: c = 2 - (2 - 2);                // 2
12:
13: c = 2-(a = 1);             // 1
14: a = (b = 3, b + 2);        // b = 3
15:                                 // a = b +2
```

- Priority like in math equation, first: *, /, then: +, -
- Associative property: from left to right (lines 9-11)
- Operation from 3 i 14 are allowed but not recommended
- **Doubts what order is correct? insert brackets**

Priority of operators

```
 6: int a = 3;
 7: int b = 2;
 8: int c = 7;
 9: int d = 1;
10:
11: int x = 0;
12: x = x + a;
13: x = x + b;
14: x = x + c;
15: x = x + d;
16:
17: int y = a + b + c + d;
18:
19: int x1 = a + b;
20: int x2 = c + d;
21: int r = x1 + x2;
```

- 12-15:
  - must be done sequentially (accumulating)
  - **Fast**: One instruction
- 17: impossible to write (in sourcecode) if we accumulate a lot of numbers
- 19:21:
  - **3, instead of adding 4** but potentially any addition results in two instructions
  - the ability to parallelize!!!
- AMD Ryzen: **4xALU**, 2xload / store, **2xFPU**
- **speed arithmetic operations**:
  - int faster (and more!) of **float**
  - "**+,-** " faster than **"*"** faster than **"/"**

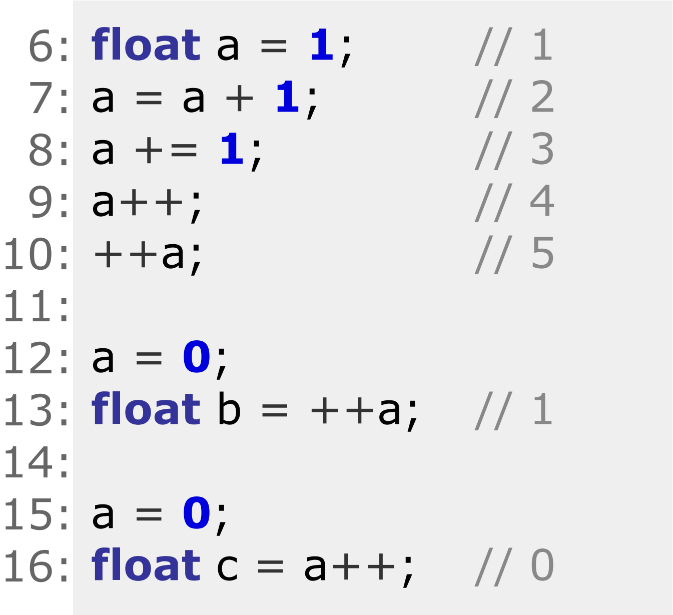
Compound assignment

```
int a = 1;
a = a + 1;    // 2

int b = 1;
b += 1;       // 2
b *= a + 1;   // 6   b=b*(a+1)
```

- Possible compound assignments: **+=, -=, *=, /=, %=, >>=, <<=, &=, |=, ^=**
- Related with hardware implementation (CPU)
- Readable
  - my_variable1=my_variable1+my_variable2;
  - my_variable1+=my_variable2;
- Readable even more when variable name is long
  - superframe_cifs_=superframe_cifs_+cifs_per_tr;
  - superframe_cifs_+=cifs_per_tr;

incrementation/decrementation



```
 6: float a = 1;       // 1
 7: a = a + 1;         // 2
 8: a += 1;            // 3
 9: a++;               // 4
10: ++a;               // 5
11:
12: a = 0;
13: float b = ++a;     // 1
14:
15: a = 0;
16: float c = a++;     // 0
```

- incrementation/decrementation:
  - very fast
  - single (short) instruction
  - different types of data (eg. float)
  - often used
  - readable code
- 7-9: modern compilers produce the same binary code
- 13: **pre-incrementation** is executed before assigning
- 16: **post-incrementation** is executed after assigning
- 9: left-handed operator
- 10: right-handed operator

Simplifications

of source code - for this presentation

Simplification

```
#include <iostream>




int main(){
     int a = 1;

     std::cout << "d=" << d << std::endl;
     std::cout << "float=" << myNumber << std::endl;



}
```

Simplification

```
#include <iostream>



int main(){
	int a = 1;

	std::cout << "d=" << d << std::endl;
	std::cout << "float=" << myNumber << std::endl;

	return 0;
}
```

Simplification

```
#include <iostream>

using namespace std;

int main(){
     int a = 1;

     std::cout << "d=" << d << std::endl;
     std::cout << "float=" << myNumber << std::endl;



}
```

Simplification

```
#include <iostream>

using namespace std;

int main(){
    int a = 1;

    std::cout << "d=" << d << std::endl;
    cout << "float=" << myNumber << endl;


}
```

Simplification

```
#include <iostream>

using namespace std;

int main(){
    int a = 1;

    cout << "d=" << d << endl;
    cout << "float=" << myNumber << endl;



}
```

Simplification

```
#include <iostream>



int main(){
	int a = 1;

	cout << "d=" << d << endl;
	cout << "float=" << myNumber << endl;


}
```

Simplification

```
int main(){
    int a = 1;

    cout << "d=" << d << endl;
    cout << "float=" << myNumber << endl;


}
```

Simplification

```
int a = 1;

cout << "d=" << d << endl;
cout << "float=" << myNumber << endl;
```

Simplification

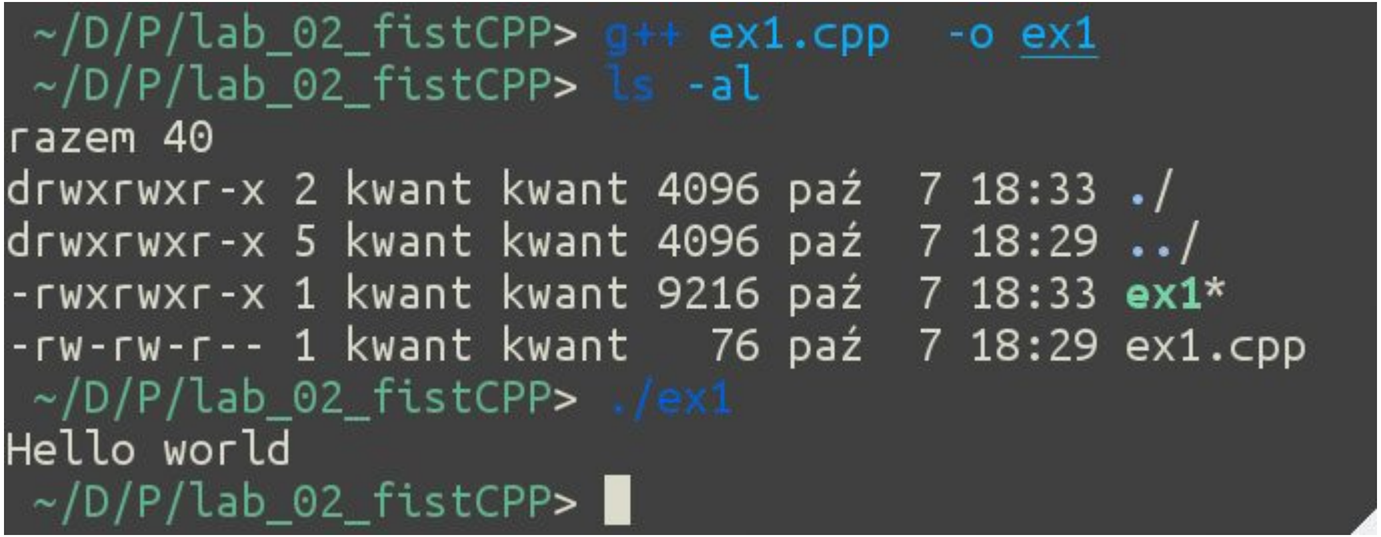
```
int a = 1;

cout << "d=" << d << endl;
cout << "float=" << myNumber << endl;
```

I wrote a program who will convert it to asm?

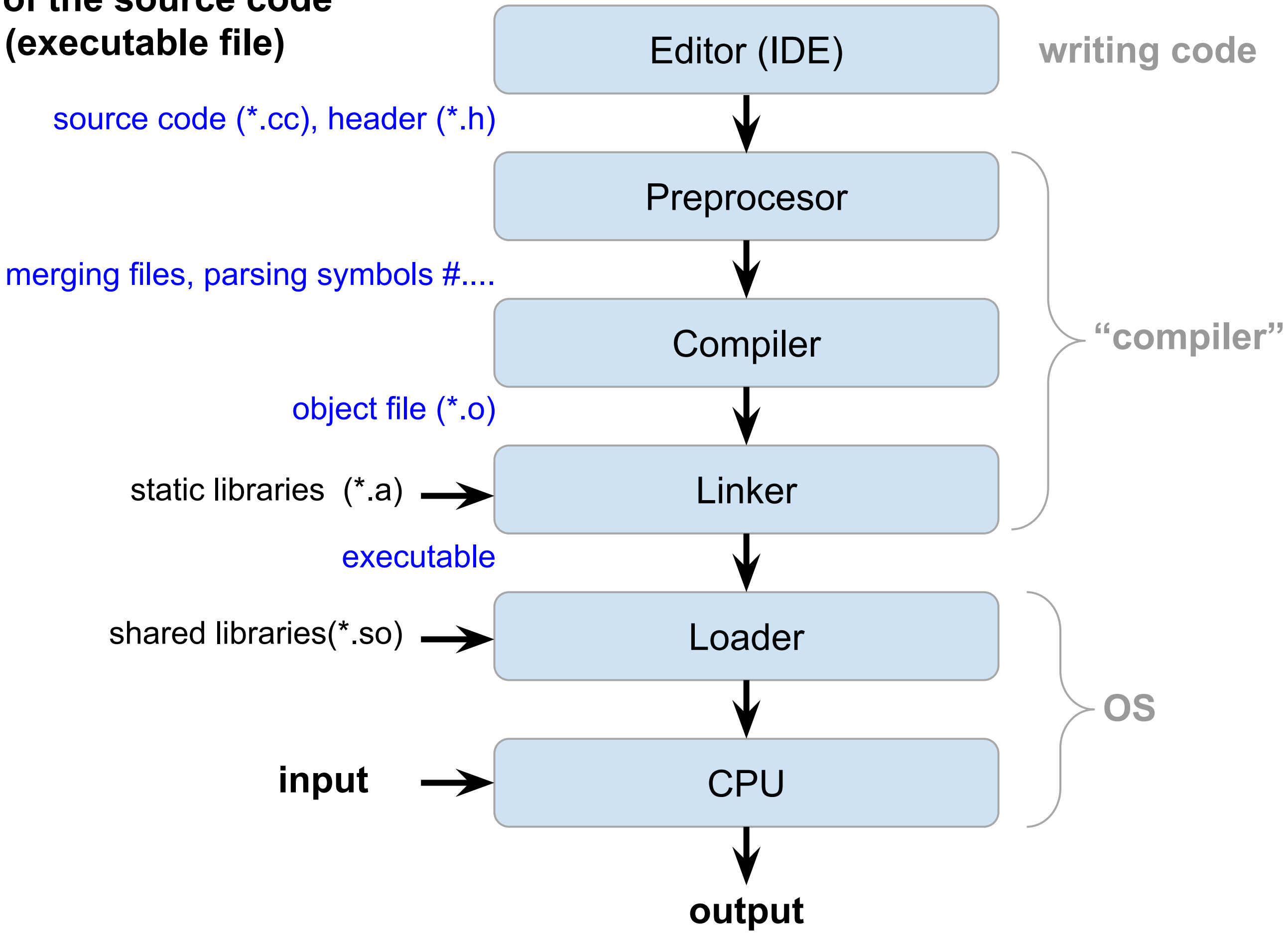
```cpp
#include <iostream>

int main(){
	std::cout << "Hello world" << std::endl;
}
```

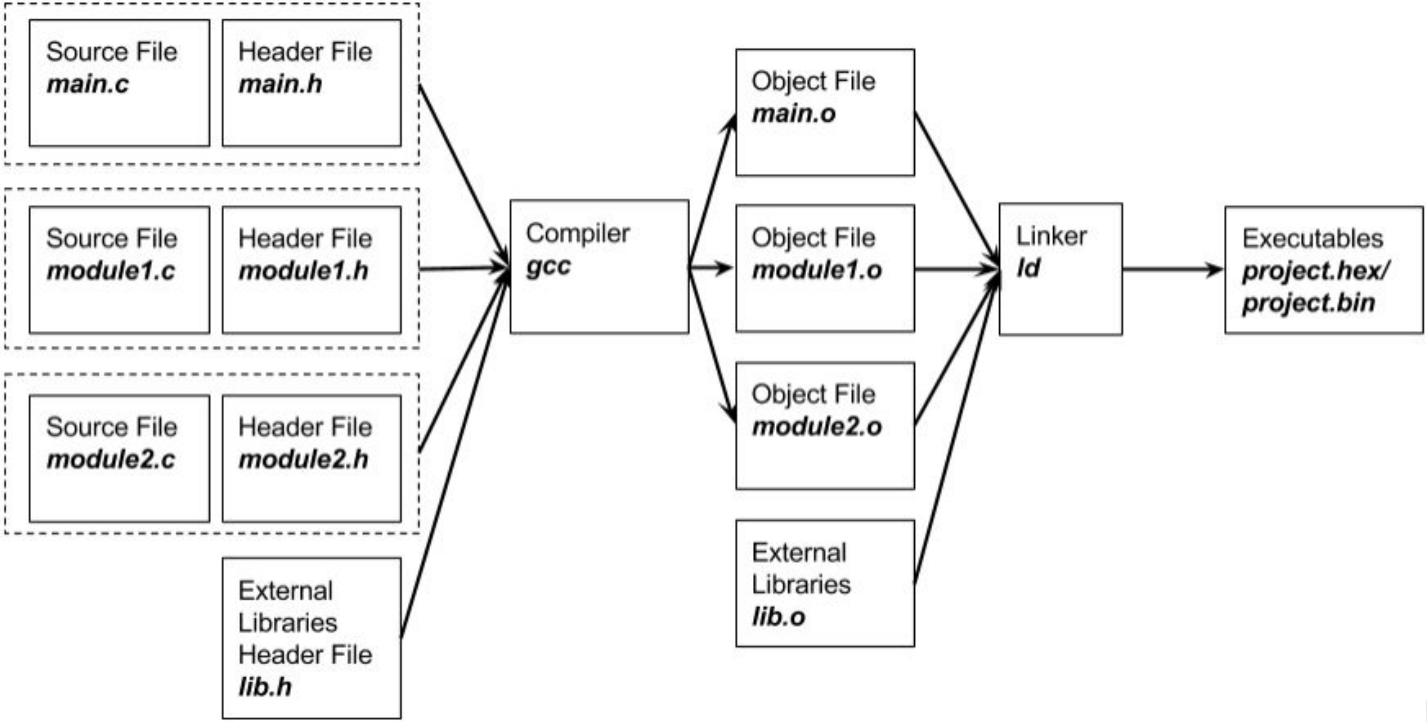


```
 ~/D/P/lab_02_fistCPP> g++ ex1.cpp  -o ex1
 ~/D/P/lab_02_fistCPP> ls -al
razem 40
drwxrwxr-x 2 kwant kwant 4096 paź  7 18:33 ./
drwxrwxr-x 5 kwant kwant 4096 paź  7 18:29 ../
-rwxrwxr-x 1 kwant kwant 9216 paź  7 18:33 ex1*
-rw-rw-r-- 1 kwant kwant   76 paź  7 18:29 ex1.cpp
 ~/D/P/lab_02_fistCPP> ./ex1
Hello world
 ~/D/P/lab_02_fistCPP> █
```

The processing of the source code for the program (executable file)



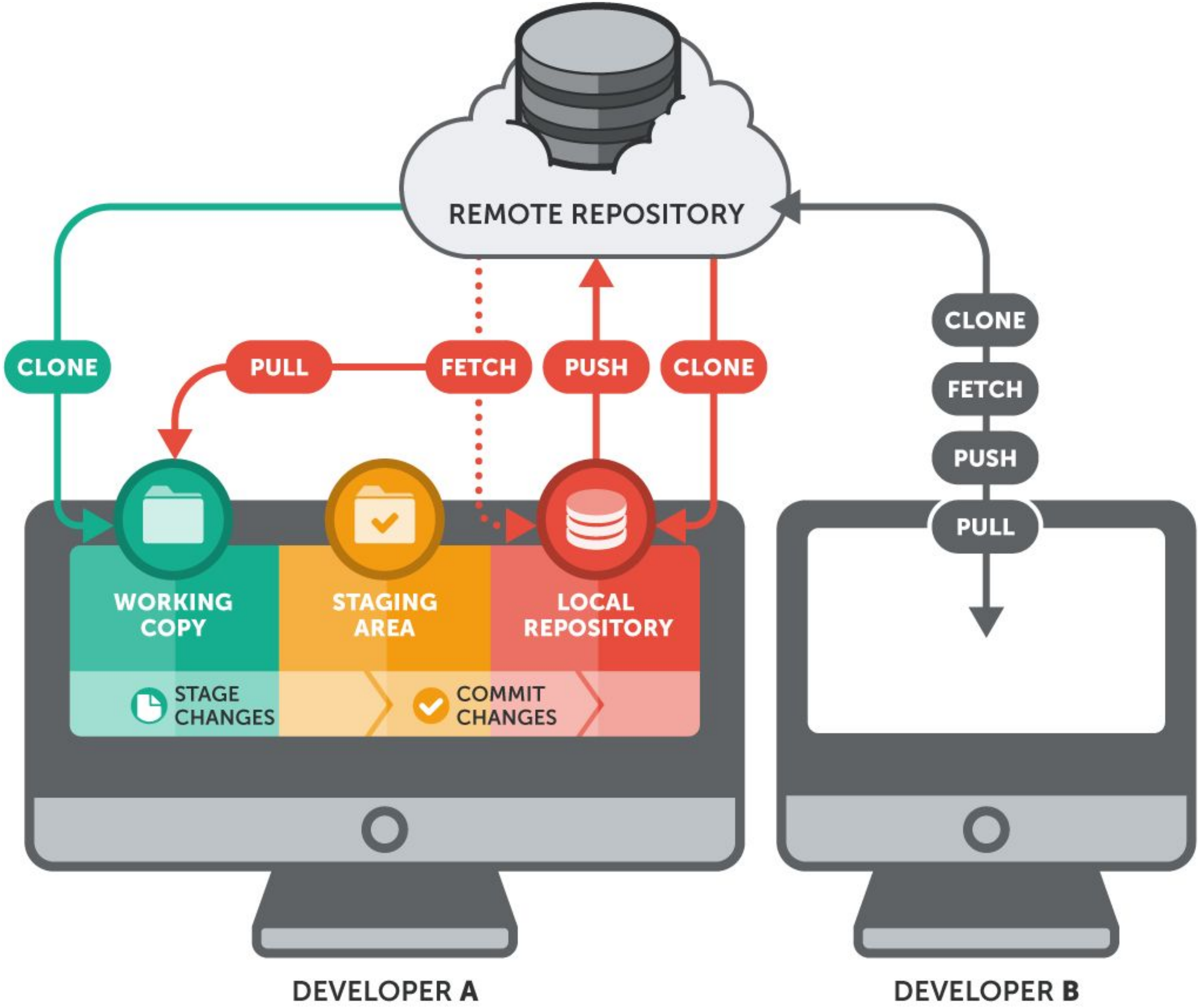
compiler - many files



https://tkcheng.files.wordpress.com/2015/05/compileandlink.png

GIT

workflow



http://gdibtv.github.io/gdi-core-git-github/images/basic-remote-workflow.png

GIT - basic workflow

» TODO: linux console (shell)

GIT - basic workflow

```
> git clone https://gitlab.com/gr/pro.git
```

- TODO: linux console (shell)
- Download (clone) external repository (remote)
  into local folder

GIT - basic workflow

```
> git clone https://gitlab.com/gr/pro.git
```

- TODO: linux console (shell)
- Download (clone) external repository (remote)
  into local folder
- “Repository” is something bigger than local folder, contains: changes history, description, metadatas, etc...
- From this moment, local folder contain part of remote repository (copy)

GIT - basic workflow

```
> git clone https://gitlab.com/gr/pro.git
```

- TODO: linux console (shell)
- Download (clone) external repository (remote)
  into local folder
- “Repository” is something bigger than local folder, contains: changes history, description, metadatas, etc...
- From this moment, local folder contain part of remote repository (copy)
- Folder pro will be created

GIT - basic workflow

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
```

- TODO: linux console (shell)
- Download (clone) external repository (remote)
  into local folder
- “Repository” is something bigger than local folder, contains: changes history, description, metadatas, etc...
- From this moment, local folder contain part of remote repository (copy)
- Folder pro will be created

GIT - basic workflow

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo “xxx” >text.txt
```

- TODO: linux console (shell)
- Download (clone) external repository (remote)
  into local folder
- “Repository” is something bigger than local folder, contains: changes history, description, metadatas, etc...
- From this moment, local folder contain part of remote repository (copy)
- Folder pro will be created
- Modify something in this directory

GIT - basic workflow

```
> git clone https://gitlab.com/gr/pro.git

> cd pro/

> echo “xxx” >text.txt

> git add text.txt
```

- TODO: linux console (shell)
- Download (clone) external repository (remote)
  into local folder
- “Repository” is something bigger than local folder, contains: changes history, description, metadatas, etc...
- From this moment, local folder contain part of remote repository (copy)
- Folder pro will be created
- Modify something in this directory
- Register new file == start tracking of this file (add to the local repository)

GIT - basic workflow

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo “xxx” >text.txt
> git add text.txt
> git commit -m “first version”
```

- TODO: linux console (shell)
- Download (clone) external repository (remote)
  into local folder
- “Repository” is something bigger than local folder, contains: changes history, description, metadatas, etc...
- From this moment, local folder contain part of remote repository (copy)
- Folder pro will be created
- Modify something in this directory
- Register new file == start tracking of this file (add to local repository)
- Register modification

GIT - basic workflow

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo “xxx” >text.txt
> git add text.txt
> git commit -m “first version”
> git push
```

- TODO: linux console (shell)
- Download (clone) external repository (remote) into local folder
- “Repository” is something bigger than local folder, contains: changes history, description, metadatas, etc...
- From this moment, local folder contain part of remote repository (copy)
- Folder pro will be created
- Modify something in this directory
- Register new file == start tracking of this file (add to local repository)
- Register modification
- Upload (push) modification to the remote repository

GIT - basic workflow

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo “xxx” >text.txt
> git add text.txt
> git commit -m “first version”
> echo “+yyy” >>text.txt
> git commit -m “second version”
> git push
```

» Do not need “push” after each commit

GIT - basic workflow

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo “xxx” >text1.txt
> echo “xxx” >text2.txt
> git add .
```

- » Do not need “push” after each commit
- » Can modify/create many files and register it all at once

GIT - basic workflow

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo “xxx” >text1.txt
> echo “xxx” >text2.txt
> git add .
> git commit -am “first version”
> git push
```

- Do not need “push” after each commit
- Can modify/create many files and register it all at once
- Can register new version for all changes (and all files)

GIT - basic workflow

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo “xxx” >text1.txt
> echo “xxx” >text2.txt
> git add .
> git commit -am “first version”
> git push
> rm text2.txt
> git add .
> git commit -am “temp. no longer …”
```

- Do not need “push” after each commit
- Can modify/create many files and register it all at once
- Can register new version for all changes (and all files)
- Removing file means registration of its removal !!!

GIT - basic workflow

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo “xxx” >text1.txt
> echo “xxx” >text2.txt
> git add .
> git commit -am “first version”
> git push
> rm text2.txt
> git add .
> git commit -am “temp. no longer …”
> git push
```

- » Do not need “push” after each commit
- » Can modify/create many files and register it all at once
- » Can register new version for all changes (and all files)
- » Removing file means registration of its removal !!!
- » Register all changes on server (remote)

GIT - basic workflow

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo “xxx” >text1.txt
> echo “xxx” >text2.txt
> git add .
> git commit -am “first version”
> git push
> rm text2.txt
> git add .
> git commit -am “temp. no longer …”
> git push
```

- Do not need “push” after each commit
- Can modify/create many files and register it all at once
- Can register new version for all changes (and all files)
- Removing file means registration of its removal !!!
- Register all changes on server (remote)
- From this moment, new version is available for other developers

GIT - team working

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo “xxx” >text.txt
> git add text.txt
> git commit -m “first version”
> git push
```

» Your “push” has modify server (remote), and now it contain new version of repository

GIT - team working

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo “xxx” >text.txt
> git add text.txt
> git commit -m “first version”
> git push
```

» Your “push” has modify server (remote), and now it contain new version of repository

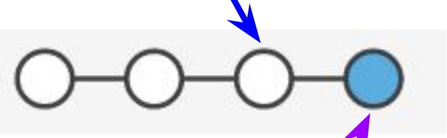
```
> git clone https://gitlab.com/gr/pro.git
> git pull
> cat text.txt
xxx
```

» Other developer…
somewhere at the other end of the world …
updates its local repository by downloading (pulling) the latest versions of files to his local directory

GIT - visualization

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo “xxx” >text.txt
> git add text.txt
> git commit -m “first version”
> echo “+yyy” >>text.txt
> git commit -m “second version”
> git push
```

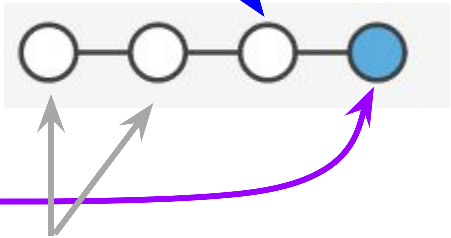
- Graphic representation of two “commits”



GIT - visualization

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo “xxx” >text.txt
> git add text.txt
> git commit -m “first version”
> echo “+yyy” >>text.txt
> git commit -m “second version”
> git push
```

- Graphic representation of two “commits”

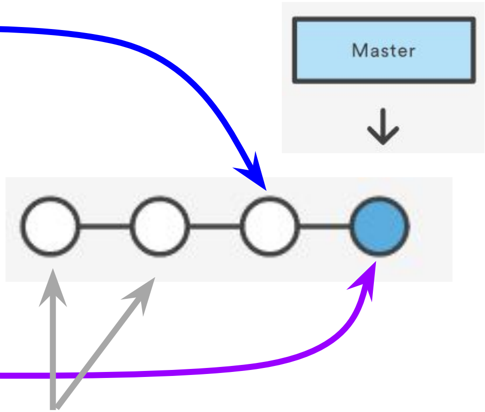


- Earlier circles represents earlier changes, not necessarily this file

GIT - visualization

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo “xxx” >text.txt
> git add text.txt
> git commit -m “first version”
> echo “+yyy” >>text.txt
> git commit -m “second version”
> git push
```

» Graphic representation of two “commits”



» Earlier circles represents earlier changes, not necessarily this file

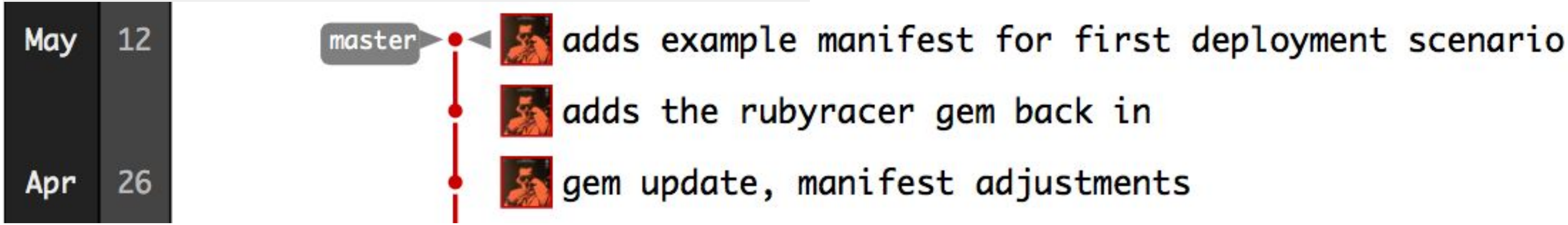
» Last circle/change/commit is the present moment (now)

GIT - wizualizacja

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo “xxx” >text.txt
> git add text.txt
> git commit -m “first version”
> echo “+yyy” >>text.txt
> git commit -m “second version”
> git push
```

» Graphic representation of two “commits”

» Visualization depends on used tools (gitlab, github, IDE, etc…)

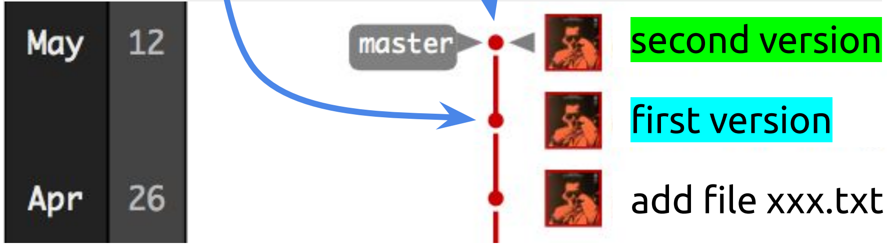


```
adds example manifest for first deployment scenario
adds the rubyracer gem back in
gem update, manifest adjustments
```

AGH

GIT - wizualizacja

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo "xxx" >text.txt
> git add text.txt
> git commit -m "first version"
> echo "+yyy" >>text.txt
> git commit -m "second version"
> git push
```



second version

first version

add file xxx.txt

- » Graphic representation of two "commits"
- » Visualization depends on used tools (gitlab, github, IDE, etc…)
- » Visualization often contains:
  - – date
  - – description (commit message)

GIT - configuration

```
> git clone https://gitlab.com/gr/pro.git
```

- Authentication, configuration
- Repository could be private, means “clone” require authentication

GIT - configuration

```
> git clone https://gitlab.com/gr/pro.git
```

- Authentication, configuration
- Repository could be private, means “clone” require authentication
- It's worth changing the configuration (**git** will not ask every time for auth…)

GIT - configuration

```
> git clone https://gitlab.com/gr/pro.git
> git config --global user.name "Your name"
```

- Authentication, configuration
- Repository could be private, means “clone” require authentication
- It's worth changing the configuration (**git** will not ask every time for auth…)
  - Name, surname

GIT - configuration

```
> git clone https://gitlab.com/gr/pro.git
> git config --global user.name "Your name"
> git config --global user.email your@email.com
```

- Authentication, configuration
- Repository could be private, means “clone” require authentication
- It's worth changing the configuration (**git** will not ask every time for auth…)
  - Name, surname
  - e-mail

GIT - configuration

```
> git clone https://gitlab.com/gr/pro.git
> git config --global user.name "Your name"
> git config --global user.email your@email.com
> git config --global push.default simple
> git config --global credential.helper "cache
    --timeout=3600"
```

- Authentication, configuration
- Repository could be private, means “clone” require authentication
- It's worth changing the configuration (**git** will not ask every time for auth…)
  - Name, surname
  - e-mail
  - push configuration (safe method)
  - remember password for 1h

GIT - configuration

```
> git clone https://gitlab.com/gr/pro.git
> git config --global user.name "Your name"
> git config --global user.email your@email.com
> git config --global push.default simple
> git config --global credential.helper "cache
    --timeout=3600"
```

- » Authentication, configuration
- » Repository could be private, means “clone” require authentication
- » It's worth changing the configuration (**git** will not ask every time for auth…)
  - – Name, surname
  - – e-mail
  - – push configuration (safe method)
  - – remember password for 1h
- » Configuration is stored in the file ~/.gitconfig

GIT - configuration

```
> git clone https://gitlab.com/gr/pro.git
> git config --global user.name "Your name"
> git config --global user.email your@email.com
> git config --global push.default simple
> git config --global credential.helper "cache
   --timeout=3600"
> cd pro/
> echo “xxx” >text.txt
> git add text.txt
> git commit -m “first version”
> echo “+yyy” >>text.txt
> git commit -m “second version”
> git push
```

- Authentication, configuration
- Repository could be private, means “clone” require authentication
- It's worth changing the configuration (**git** will not ask every time for auth…)
  - Name, surname
  - e-mail
  - push configuration (safe method)
  - remember password for 1h
- Configuration is stored in the file ~/.gitconfig

GIT - configuration

```
> git clone https://gitlab.com/gr/pro.git
> git config --global user.name "Your name"
> git config --global user.email your@email.com
> git config --global push.default simple
> git config --global credential.helper "cache
    --timeout=3600"
> cd pro/
> echo "xxx" >text.txt
> git add text.txt
> git commit -m "first version"
> echo "+yyy" >>text.txt
> git commit -m "second version"
> git push
```

- Authentication, configuration
- Repository could be private, means “clone” require authentication
- It's worth changing the configuration (**git** will not ask every time for auth…)
  - Name, surname
  - e-mail
  - push configuration (safe method)
  - remember password for 1h
- Configuration is stored in the file ~/.gitconfig
- In practice, cloning and configuration once per repository

GIT - configuration

```
> git clone https://gitlab.com/gr/pro.git
> git config --global user.name "Your name"
> git config --global user.email your@email.com
> git config --global push.default simple
> git config --global credential.helper "cache
    --timeout=3600"
> cd pro/
> echo “xxx” >text.txt
> git add text.txt
> git commit -m “first version”
> echo “+yyy” >>text.txt
> git commit -m “second version”
> git push
```

- Authentication, configuration
- Repository could be private, means “clone” require authentication
- It's worth changing the configuration (**git** will not ask every time for auth…)
  - Name, surname
  - e-mail
  - push configuration (safe method)
  - remember password for 1h
- Configuration is stored in the file ~/.gitconfig
- In practice, cloning and configuration once per repository
- In our labs, at the beginning of each classes !!!

AGH

Thank you!