

Python 00

Elias Wachmann

2024

“The magic of computing begins with 0, a simple binary digit that serves as a powerful reminder that even the smallest building blocks can create wonders.”

Content

1. Introduction
2. Binary numbers
3. Hexadecimal numbers
4. How do Computers work
5. How do Computers communicate
6. Software Survival Kit
7. Additional Resources

Introduction

Introduction

This course is designed to teach you the basics of programming in Python.

However, before we start, we need to understand how computers work. This slide deck will give you a brief introduction to the topic.

Introduction

The slide decks will include lots of examples for various concepts.

Feel free to try them out **yourself**, since this is the best way to learn programming.

By the end of this course, you'll have a solid foundation in Python programming and be ready to take on more advanced courses and real-world projects!

Binary numbers

Binary Representation

Binary representation is a method of representing numbers using only two digits, typically 0 and 1. It is commonly used in computer systems.

For example, the decimal number 5 can be represented in binary as 101, where the rightmost digit represents the 2^0 place, the next digit represents the 2^1 place, and so on.

Binary Representation of 42

To convert the decimal number 42 to binary, we can use the following method:

	42	21	10	5	2	1	0
	÷ 2	÷ 2	÷ 2	÷ 2	÷ 2	÷ 2	
	21	10	5	2	1	0	
Remainder:	0	1	0	1	0	1	

Reading the remainders from right to left, we get the binary representation of 42 as 0b00101010.

Or: $2^5 + 2^3 + 2^1 + 2^0 = 32 + 8 + 2 = 42$.

Representing Negative Numbers

Negative numbers can be represented in binary using a method called **Two's Complement**:

Step 1: Write down the binary representation of the positive number.

Step 2: Invert all the bits (change 0s to 1s and 1s to 0s).

Step 3: Add 1 to the result from step 2.

$$42 \xrightarrow{1} 0b00101010 \xrightarrow{2} 0b11010101 \xrightarrow{3} 0b11010110 = -42$$

Some Examples

0 to 15 in binary:

Decimal	Binary	Decimal	Binary
0	0b0000	8	0b1000
1	0b0001	9	0b1001
2	0b0010	10	0b1010
3	0b0011	11	0b1011
4	0b0100	12	0b1100
5	0b0101	13	0b1101
6	0b0110	14	0b1110
7	0b0111	15	0b1111

Hexadecimal numbers

Hexadecimal Representation

Hexadecimal representation is a method of representing numbers using 16 distinct symbols, typically 0-9 and A-F. It is commonly used in computer systems as a more compact representation of binary numbers.

For example, the decimal number 42 can be represented in hexadecimal as 2A, where the rightmost digit represents the 16^0 place and the next digit represents the 16^1 place.

Hexadecimal Representation of 42

To convert the decimal number 42 to hexadecimal, we can use the following method:

$$\begin{array}{r} 42 \quad 2 \quad 0 \\ \div 16 \quad \div 16 \\ \hline 2 \quad 0 \\ \text{Remainder: } 10 \quad 2 \end{array}$$

Reading the remainders from right to left, we get the hexadecimal representation of 42 as 0x2A.

$$\text{Or: } 16^1 + 10^0 = 32 + 10 = 42.$$

Converting Hexadecimal to Binary

To convert a hexadecimal number to binary, each digit can be replaced by its 4-bit binary equivalent.

For example, to convert `0x2A` to binary:

2 → 0010

A → 1010

Concatenating the binary equivalents, we get
`0b00101010`.

Some Examples

0 to 15 in hexadecimal:

Decimal	Hexadecimal	Decimal	Hexadecimal
0	0x0	8	0x8
1	0x1	9	0x9
2	0x2	10	0xA
3	0x3	11	0xB
4	0x4	12	0xC
5	0x5	13	0xD
6	0x6	14	0xE
7	0x7	15	0xF

How do Computers work

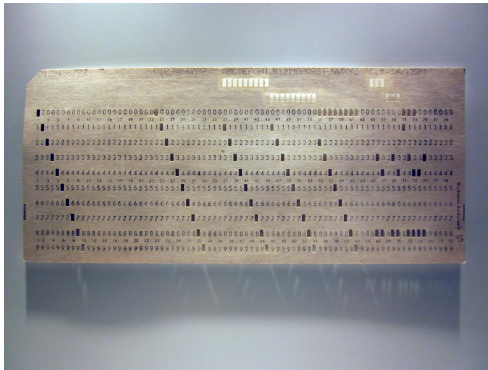
Bits and Bytes

Computers store all information as a series of 0s and 1s. These 0s and 1s are called **bits**. A group of 8 bits is called a **byte**.

For example, the decimal number 42 can be represented as the byte `0b00101010`.

Bit storage

How do computers store bits?

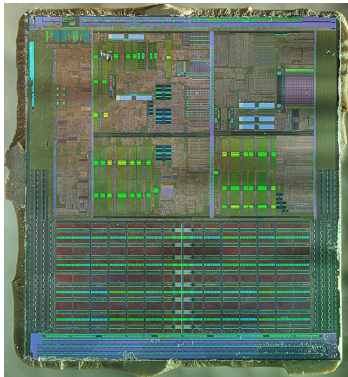


https://en.wikipedia.org/wiki/Punched_card

From punch cards to transistors

Fortunately, we don't have to use punch cards anymore. Modern computers use billions of tiny **transistors** to store bits and execute instructions.

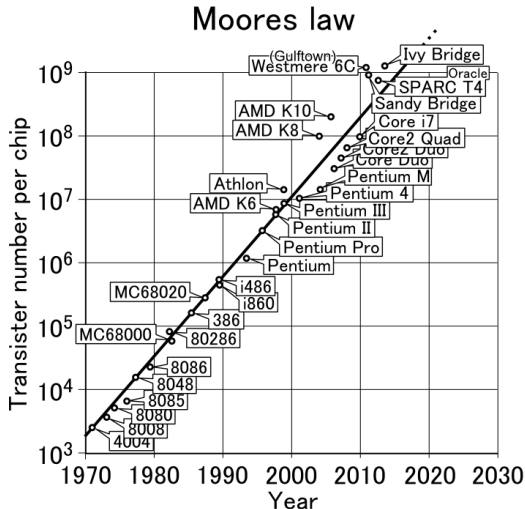
AMD Athlon 64 3200+



https://commons.wikimedia.org/wiki/File:Moore's_law_%281970-2011%29.PNG

Moore's Law

Moore's Law:
The number of transistors on a microchip doubles approximately every two years.



https://commons.wikimedia.org/wiki/File:Moore's_law_%281970-2011%29.PNG

Computer Hardware

A computer consists of the following main components:

- **Central Processing Unit (CPU):** The brain of the computer. It executes instructions.
- **Motherboard:** Main circuit board of the computer.
- **Memory:** RAM, ROM, hard disk, etc.
- **Input devices:** Keyboard, mouse, microphone, camera, etc.
- **Output devices:** Monitor, printer, speakers, etc.
- **Power supply:** Provides power to the computer.
- **Network interface:** For connecting to other networks.

Central Processing Unit (CPU)

The CPU is the brain of the computer. It executes instructions which are stored in the memory.

Instruction: A command that the CPU can execute. ADD, MOV, JMP, etc. are examples of instructions.

Instruction set: A set of instructions that the CPU can execute. Examples: x86, ARM

Registers: Small amount of memory inside the CPU. The CPU can access the registers very quickly.

Motherboard

Facilitates communication between the CPU, memory, input devices, output devices, etc. and provides power to the components.

BIOS: Program which executes when the computer is turned on. It initializes the hardware and loads the operating system.

Memory

Computer memory is used to store data and instructions executed by the CPU.

Various types of memory are used in a computer:

- **RAM:** Random Access Memory. (volatile)
Stores data & instructions that are currently being used by the CPU.
- **ROM:** Read Only Memory. (non-volatile)
Stores the BIOS and other programs that are executed when the computer is turned on.
- **Disk:** HDD/SSD/Tape etc. (non-volatile)
Used to store large amounts of data.

Input & Output devices

Input and output devices are used to communicate with the computer.

Historically: keyboard, later mouse and touch screen.

Monitors CRTs: limited to 80 characters per line.

Nowadays: LCDs, OLEDs, etc.

Power supply & Network interface

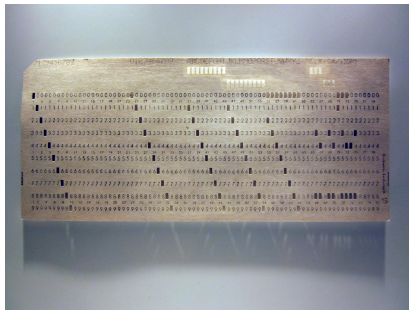
The power supply provides power to various components of the computer.

It converts the AC power from the outlet to DC power with various voltages (+3.3 V, +5 V and +12 V) required by the computer.

The network interface is used to connect the computer to other networks via various networking technologies (Ethernet, WiFi, etc.).

How to get a computer to do something?

Historically, computers were programmed using punch cards.



https://en.wikipedia.org/wiki/Punched_card

How to get a computer to do something?

Nowadays, computers can be programmed using a variety of **programming language**.

Software: set of instructions and data that is executed by the hardware of the computer.

Programming language: A language that is used to write software.

Different levels of complexity

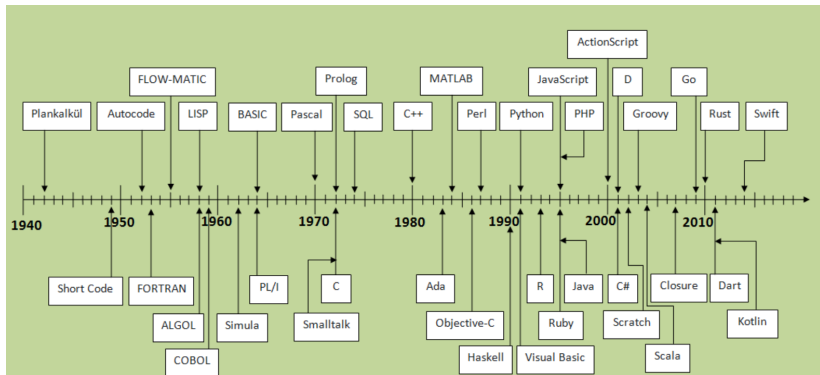
- **Machine language:** The language of the computer. Lowest level language & only language that the computer can understand. It is written in **binary**.
- **Assembly language:** It is written in a symbolic form that is easier to read and write than machine language. Yet still very close to machine language.
- **High-level languages:** A language that is closer to human languages than machine language. It is written in a symbolic form that is easier to read and write than machine language. It is portable and can be run on different machines.

High-level languages

Because Machine as well as Assembly code is cumbersome to write, **High-level languages** were first developed in the 1950s.

- **Fortran**: It was the first high-level language to be widely used. (1957)
- **C/C++**: It is the most widely used programming language. (1972 / 1985)
- **Java**: It is a general-purpose, concurrent, class-based, object-oriented language. (1995)
- **Python**: It is a general-purpose, high-level, interpreted, dynamic programming language. (1991)

High-level languages timeline



<https://i0.wp.com/javaconceptoftheday.com/wp-content/uploads/2019/07/TimelineOfProgrammingLanguages.png?ssl=1>

Who is translating?

High level languages are translated into machine code by a **compiler**.

The compiler takes the text file containing the code and translates it into 0s and 1s that the computer can understand.

The compiler then creates an executable file (.EXE) that can be run on the computer.

From compilers to interpreters

Compilers have some disadvantages. They can be slow to run, and they require the user to compile the code before it can be executed.

This led to the development of **interpreters**, which execute code **directly** without the need for compilation.

Interpreters are often used for scripting languages (like `python`) and for prototyping new code quickly.

Operating system

An operating system (Linux, Windows, Mac OS, etc.) is a software which manages the computer hardware and software resources and provides common services for computer programs.

Manages and hands out RAM-memory to programs.

Manages and hands out CPU-time to programs.

Provides abstraction of the hardware.

“Operating systems are the masterful illusionists, the fair referees, and the unifying glue that seamlessly blend the complexities of hardware and the intricacies of software, creating a harmonious dance between them” - Daniel Gruss (modified)

How do Computers communicate

How do Computers communicate?

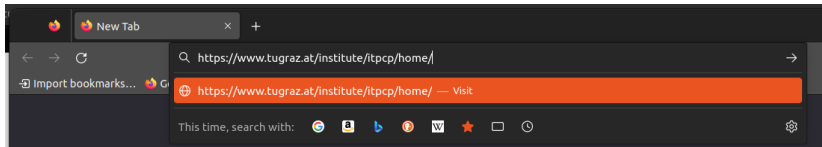
Computers can communicate with other remote computer by using:

- **Ethernet**: Used to connect computers in a local network.
- **WiFi**: Wireless Fidelity. Used to connect computers in a local network.
- **TCP/UDP**: Transmission Control Protocol / User Datagram Protocol. Used to transfer data over the internet.
- **HTTP**: HyperText Transfer Protocol. Used to transfer web pages.

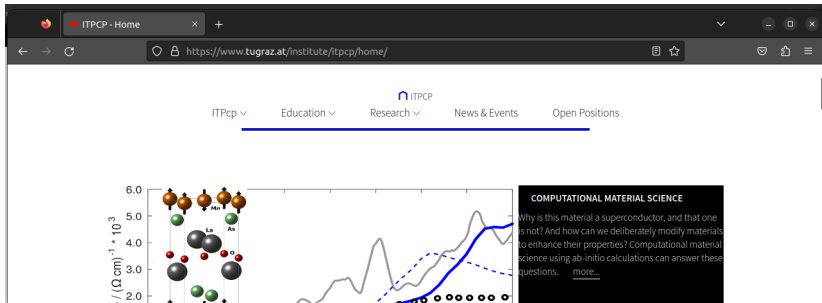
How do Computers communicate?

... they communicate locally using:

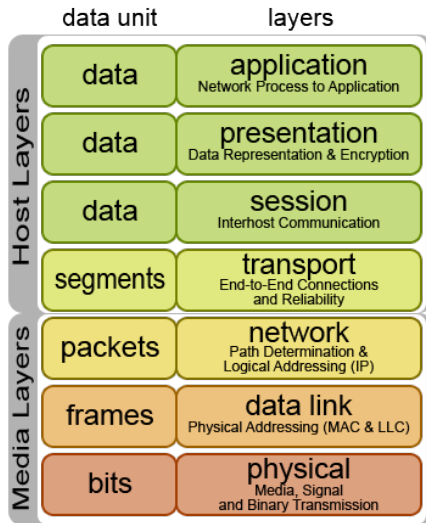
- **USB**: Universal Serial Bus. Used to connect peripherals to a computer.
- **RS-232**: Recommended Standard 232 (serial communication). Used to connect peripherals to a computer.
- **MIDI**: Musical Instrument Digital Interface. Used to connect musical instruments to a computer.



↓ How does this work? ↓



OSI Model



https://en.wikipedia.org/wiki/OSI_model

Let's look at a simplified example:

We want to access www.tugraz.at/institute/itpcp/

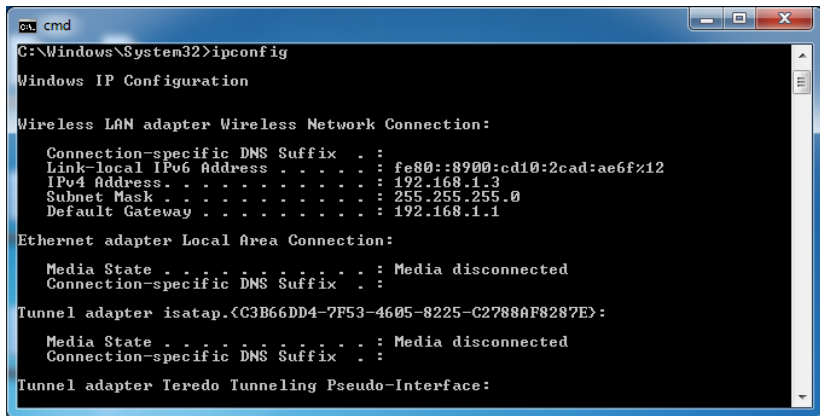
Our browser will send a **request** to the server at www.tugraz.at using the **HTTP** protocol.

But how does he know where to send the request?

Computers have addresses, just like houses. They are called **IP addresses**.¹

¹ actually it's more involved especially with IPv4 addresses → [IPv4-exhaustion](#)

IPv4 addresses



```
C:\ cmd
C:\Windows\System32>ipconfig

Windows IP Configuration

Wireless LAN adapter Wireless Network Connection:

    Connection-specific DNS Suffix  . : 
    Link-local IPv6 Address . . . . . : fe80::8900:cd10:2cad:ae6f%12
    IPv4 Address. . . . . : 192.168.1.3
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.1.1

Ethernet adapter Local Area Connection:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . : 

Tunnel adapter isatap.{C3B66DD4-7F53-4605-8225-C2788AF8287E}:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . : 

Tunnel adapter Teredo Tunneling Pseudo-Interface:
```

<https://www.flickr.com/photos/johnlsloan/5477351098>

How to get an IP address for a website?

Just ask another server for it → **DNS request** The

request will be send from your computers network card to your router.

The router will forward the request to your ISP (Internet Service Provider; A1, Magenta, etc.).

Maybe your ISP knows the IP address of the website, but if not, he will ask another server.

In case your ISP doesn't know the address, the request will be forwarded to the **root name server**.

Root name server – CERN Meyrin



Foto: Elias Wachmann

Now finally . . .

The root name server will send the IP address back to your ISP, which will send it back to your router, which will send it back to your computer.

Your computer now knows the address of the www.tugraz.at server and can send the actual request to load www.tugraz.at/institute/itpcp/

The server will send the requested website back to your computer, which will display it in your browser.

Software Survival Kit

Software Survival Kit

This guide is a collection of useful information to use a computer to its full potential based on the lecture The Missing Semester of Your CS Education by MIT.

Video lectures for the various topics can be found here.

Shell [video]

The shell is a program that takes commands from the keyboard and executes corresponding programs which are preinstalled with the OS.

```
etschgil@Deep-thought:/mnt/c/REPOS/exercises-python$
```

Empty shell with `user@hostname:directory$`

Shell – echo

A simple example is `echo`: prints its arguments to the standard output.

```
etschgi@Deep-thought:/mnt/c/REPOS/exercises-python$ echo "hello shell!"  
hello shell!  
etschgi@Deep-thought:/mnt/c/REPOS/exercises-python$
```

How can the shell know where to find the program `echo`?

PATH: A list of directories where the shell looks for programs.

Shell – \$PATH

Let's look at the PATH variable.

```
etschgil@Deep-thought: /mnt/c/REPOS/exercises-python$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/usr/lib/wsl/lib:/mnt/c/Windows/system32:/mnt/c/Windows:/mnt/c/Windows/System32/Wbem:/mnt/c/Windows/System32/WindowsPowerShell/v1.0:/mnt/c/Windows/System32/OpenSSH:/mnt/c/Program Files/dotnet:/mnt/c/Program Files/Docker/Docker/resources/bin:/mnt/c/ProgramData/DockerDesktop/version-bin:/mnt/c/Program Files/Git/cmd:/mnt/c/Users/elias/AppData/Local/Programs/Python/Python311/Script s:/mnt/c/Users/elias/AppData/Local/Programs/Python/Python311:/mnt/c/Users/elias/AppData/Local/Programs/Python/Python310/Scripts:/mnt/c/Users/elias/AppData/Local/Programs/Python/Python310:/mnt/c/Users/elias/AppData/Local/Microsoft/WindowsApps:/mnt/c/Users/elias/AppData/Local/Programs/Microsoft VS Code/bin:/mnt/c/texlive/2022/bin/win32:/mnt/c/Program Files/JetBrains/IntelliJ IDEA 2022.2.3/bin:/mnt/c/msys64/mingw64/bin:/snap/bin
```

Echo prints out the value of the \$PATH variable.
\$PATH includes the /usr/bin where echo is located.

```
etschgil@Deep-thought: /mnt/c/REPOS/exercises-python$ which echo
/usr/bin/echo
```

Shell – Directories

Directories: A directory is a file that contains a list of other files and directories.

Absolute path: The full path to a file or directory.

Relative path: The path to a file or directory relative to the current directory.

Change the current directory with `cd`.

```
etschgil@Deep-thought:/mnt/c/REPOS/exercises-python/slides/00$ cd ./fig
etschgil@Deep-thought:/mnt/c/REPOS/exercises-python/slides/00/fig$ cd ..
etschgil@Deep-thought:/mnt/c/REPOS/exercises-python/slides/00$ cd /mnt/c/REPOS/exercises-python/
etschgil@Deep-thought:/mnt/c/REPOS/exercises-python$
```

`cd ./directory` relative – `cd /mnt/directory` absolute
– `cd ..` go back

Shell – ls

ls: List directory contents.

```
etschgi1@Deep-thought:/mnt/c/REPOS/exercises-python/slides$ ls
00 01 02 03 04 05 06 common matlab_slides
etschgi1@Deep-thought:/mnt/c/REPOS/exercises-python/slides$ ls -l
total 0
drwxrwxrwx 1 etschgi1 etschgi1 4096 Apr 26 10:53 00
drwxrwxrwx 1 etschgi1 etschgi1 4096 Apr 17 10:09 01
drwxrwxrwx 1 etschgi1 etschgi1 4096 Apr 17 10:09 02
drwxrwxrwx 1 etschgi1 etschgi1 4096 Apr 17 10:09 03
drwxrwxrwx 1 etschgi1 etschgi1 4096 Apr 19 16:16 04
drwxrwxrwx 1 etschgi1 etschgi1 4096 Apr 19 14:31 05
drwxrwxrwx 1 etschgi1 etschgi1 4096 Apr 19 15:36 06
drwxrwxrwx 1 etschgi1 etschgi1 4096 Apr 19 17:21 common
drwxrwxrwx 1 etschgi1 etschgi1 4096 Mar 26 08:57 matlab_slides
etschgi1@Deep-thought:/mnt/c/REPOS/exercises-python/slides$
```

Need more information?

No problem! Just type: `man` followed by the program name:

```
LS(1)                                User Commands                                LS(1)

NAME
    ls - list directory contents

SYNOPSIS
    ls [OPTION]... [FILE]...

DESCRIPTION
    List information about the FILES (the current directory by default). Sort entries alphabetically if none of
    -cftuvSUX nor --sort is specified.

    Mandatory arguments to long options are mandatory for short options too.

    -a, --all
        do not ignore entries starting with .

    -A, --almost-all
        Manual page ls(1) line 1 (press h for help or q to quit)
```

Shell – try it out yourself

Lecture notes can be found [here](#).

Feel free to try out some exercises on the page to get familiar with the shell.

Git

Git [video]

Git is a version control system.

Version control system: A system that records changes to a file or set of files over time so that you can recall specific versions later.

But why should I use it?

No more `report_v1_final_final_final_final.pdf`!

You can download git [here](#).

Git – initializes a new repository (init)

Repository: A directory where git has been initialized to start version controlling your files.

To initialize a new repository use `git init`.

```
etschgil@Deep-thought:/mnt/c/REPOS/example_git$ git init
Initialized empty Git repository in /mnt/c/REPOS/example_git/.git/
etschgil@Deep-thought:/mnt/c/REPOS/example_git$
```

This creates a hidden folder `.git` which contains all the information about the repository.

Git – add files (add)

To add files to the repository use `git add`.
This stages the files for the next commit.

```
etschgil@Deep-thought:/mnt/c/REPOS/example_git$ echo "Hello World" > testfile.txt
etschgil@Deep-thought:/mnt/c/REPOS/example_git$ git add testfile.txt
etschgil@Deep-thought:/mnt/c/REPOS/example_git$ git add .
etschgil@Deep-thought:/mnt/c/REPOS/example_git$
```

Here "Hello World" is redirected into a file called `testfile.txt` using the `>` operator.
All files in the current folder and in subfolder below can be added using `git add .` (with a fullstop).

Git – see current status (status)

To see the current status of the repository use `git status`.

```
etschgil@Deep-thought:/mnt/c/REPOS/example_git$ git status
On branch main

No commits yet

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

etschgil@Deep-thought:/mnt/c/REPOS/example_git$
```

`testfile.txt` is staged for commit.

Git – commit changes (commit)

To commit the changes use `git commit`.

```
etschgil@Deep-thought:/mnt/c/REPOS/example_git$ git commit -m "This is my message"
Author identity unknown

*** Please tell me who you are.

Run

  git config --global user.email "you@example.com"
  git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: empty ident name (for <etschgil@Deep-thought.localdomain>) not allowed
etschgil@Deep-thought:/mnt/c/REPOS/example_git$
```

That didn't work → setup your git user name & email.

Git – setup user name & email (config)

To setup your user name & email use `git config`.
The `--global` flag sets the configuration (globally) for the current user.

```
etschgi1@Deep-thought:/mnt/c/REPOS/example_git$ git commit -m "This is my message"  
[main b71f6fb] This is my message  
1 file changed, 1 insertion(+), 1 deletion(-)
```

After setting up the user name & email you can commit the changes.
Commit messages should be short and meaningful and can be added using the `-m` flag.

Git – see commit history (log)

To see the commit history use `git log`.

```
etschgil@Deep-thought:/mnt/c/REPOS/example_git$ git log
commit b71f6fb5ecc50bbd65861a10d060fa37deffd5a9 (HEAD -> main)
Author: max huber <e.wachmann@student.tugraz.at>
Date:   Wed Apr 26 16:56:48 2023 +0200
```

```
    This is my message
```

Each commit has a unique identifier called a **SHA**.
This allows you to go back to a specific commit.

Git – Remote repositories

Remote repository: A repository that is hosted on the Internet or another network.

It is a good idea to have a remote repository to backup your work or collaborate with others.

GitLab: A website that hosts git repositories (relevant for the exercises later).

Git – Remote (remote)

To add a remote repository use `git remote` followed by an alias and `url`.

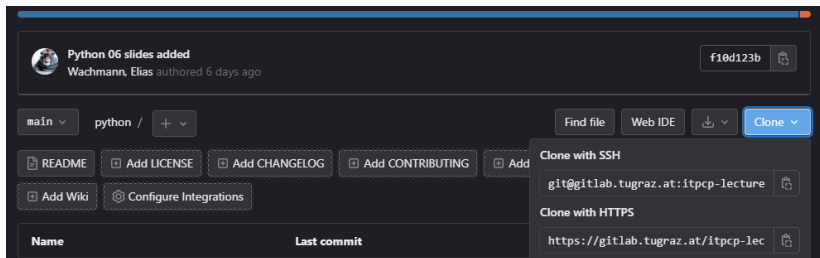
```
etschgi@Deep-thought:/mnt/c/REPOS/example_git$ git remote add origin ../remote/  
etschgi@Deep-thought:/mnt/c/REPOS/example_git$ git remote  
origin
```

Here `origin` is used as a remote alias and `../remote/` as the url.

Git – Remote (clone)

To clone a remote repository use `git clone` followed by the `url`.

The url can be found on the gitlab page of the repository.



Git – Remote (push & pull)

When cloning a repository the remote alias `origin` is automatically added.

All changes are loaded into the local repository.

After working on the project locally you can push the changes to the remote repository using `git push`.

Note: You can also pull changes from a remote repository using `git pull`.

Git – SSH keys?

SSH: Secure Shell (SSH) is a cryptographic network protocol for operating network services securely over an unsecured network.

SSH keys: An SSH key is an access credential in the SSH protocol.

You need to setup SSH keys to push changes to a remote repository.

Use [this Guide](#) to setup [link](#) your SSH keys to GitLab.

Git – Branches

Branch: A branch is a parallel version of a repository.

Main branch: The default branch when you create a repository.

You can create a new branch using `git branch` followed by the branch name.

You can switch to a branch using `git switch` followed by the branch name.

Git – Branches (merge)

To **merge** a branch into the main branch use `git merge` followed by the branch name.

This will merge the changes from the branch into the main branch.

Beware of conflicts!

If you and another person change the same line of code, git will not know which change to keep → a merge conflict will occur which must be resolved manually.

Need more information?

No problem! Just type: `git help` followed by a git command:

```
GIT-INIT(1)                                     Git Ma
nual                                           GIT-INIT(1)

NAME
    git-init - Create an empty Git repository or reinitialize an existing one

SYNOPSIS
    git init [-q | --quiet] [--bare] [--template=<template_directory>]
              [--separate-git-dir <git dir>] [--object-format=<format>]
              [-b <branch-name> | --initial-branch=<branch-name>]
              [--shared[=<permissions>]] [directory]

DESCRIPTION
    This command creates an empty Git repository - basically a .git directory with subdirectories for objects, refs
    /heads, refs/tags, and template files. An initial branch without any commits will be created (see the
    --initial-branch option below for its name).

    If the $GIT_DIR environment variable is set then it specifies a path to use instead of ./.git for the base of t
    Manual page git-init(1) line 1 (press h for help or q to quit)
```

Git – PRACTICE!!!

That was a lot of information.

However, **git** can make your life a lot easier!

Invest some time and save a lot of time in the future:

- Learn Git & GitHub
- Learn Git Branching
- Learn Git interactively
- Merging in Git

Additional Resources

Additional Resources

- Codewars
- Leetcode
- Oh my git!
- Interactive python tutorials
- Learn Python 3
- CS50