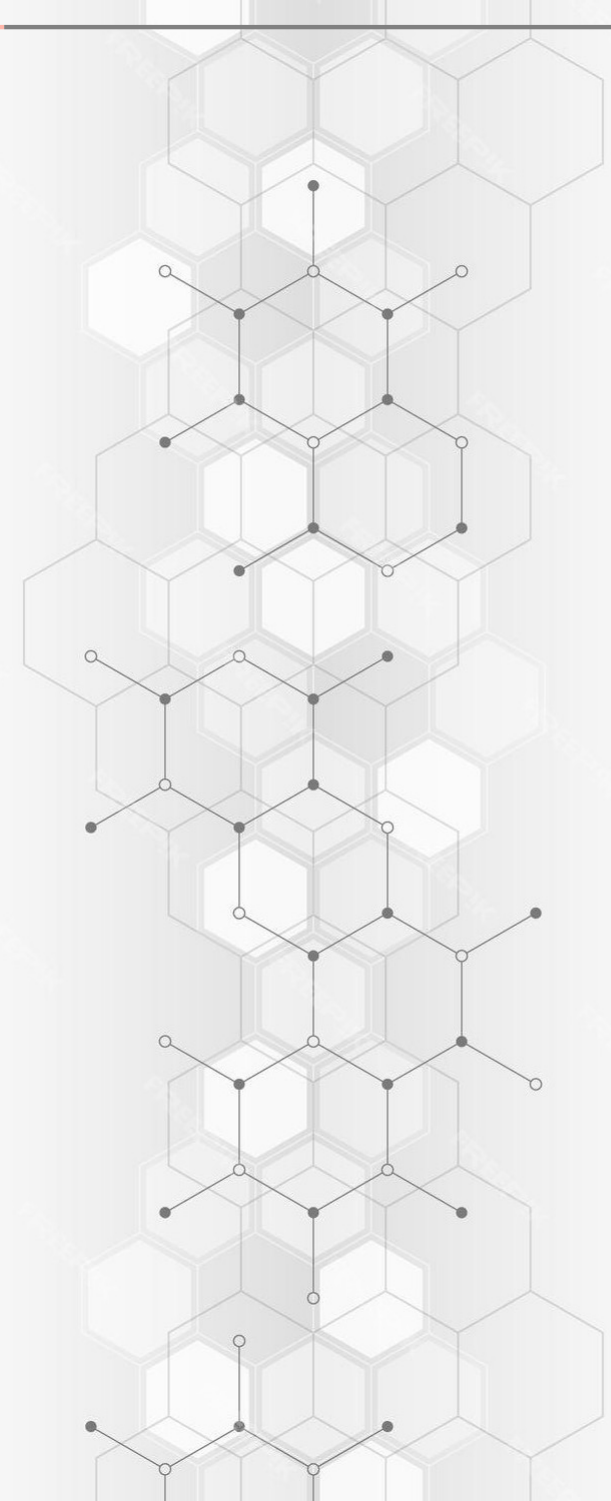
Recommended Programming Practices Practical Guide

Team Meeting - Network Dynamics and Computations

December 9, 2024



- Objectives
- Overview: Structure, Tools, Steps
- Initializing a Project

Objectives 3

Project - Definition

Cohesive set of code, resources, and related files organized to achieve a specific *goal*.

Not only code
Includes documentation, configuration files, tests...

Criteria for project quality?

Quality Criteria

Ultimate Goals

Reliability

Robustness, Reproducibility

Proximal Goals

Quality Criteria

Ultimate Goals

Reliability

Robustness, Reproducibility

Maintainability

Modifiability, Scalability

Proximal Goals

Quality Criteria

Ultimate Goals

Reliability

Robustness, Reproducibility

Maintainability

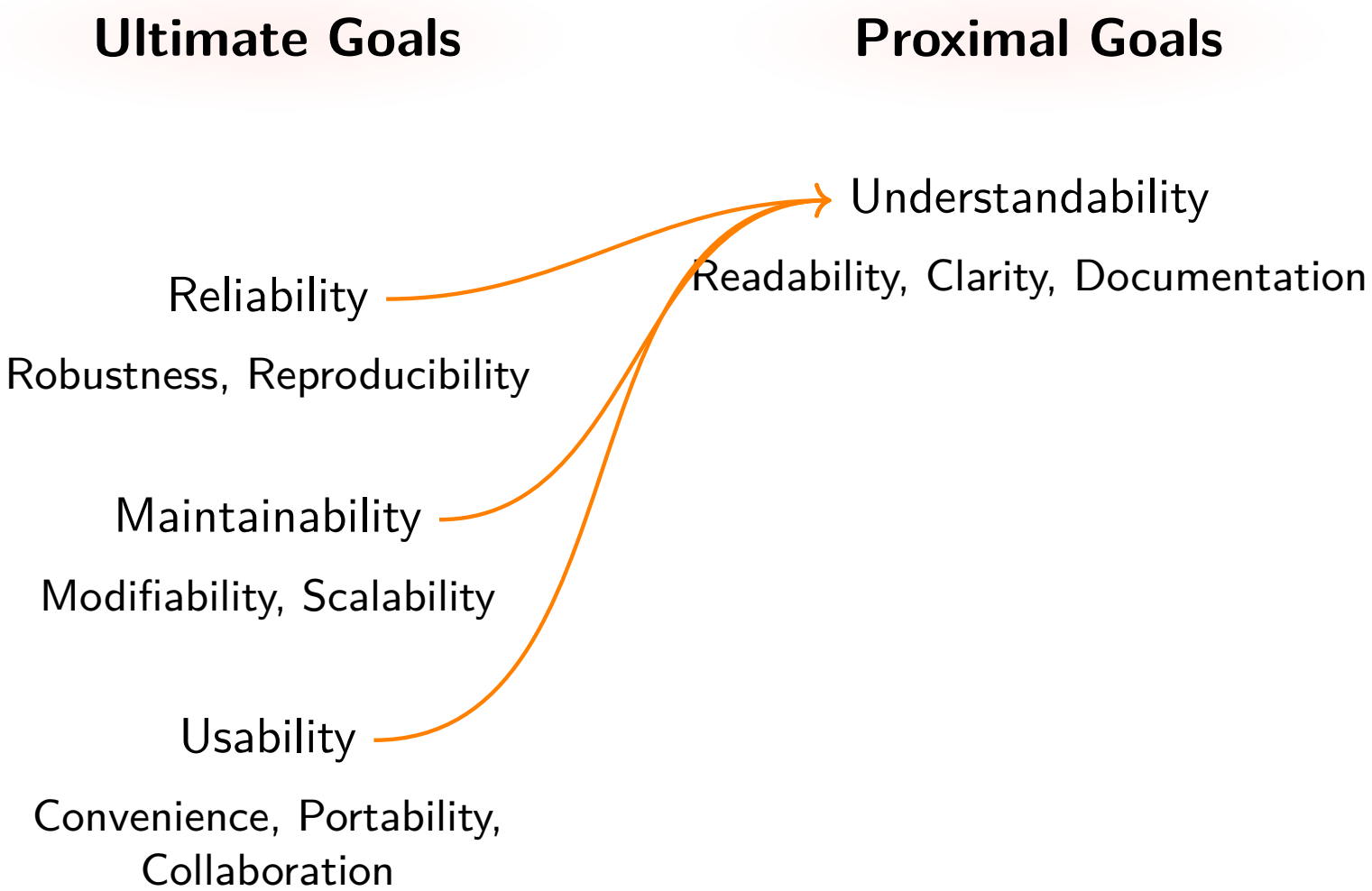
Modifiability, Scalability

Usability

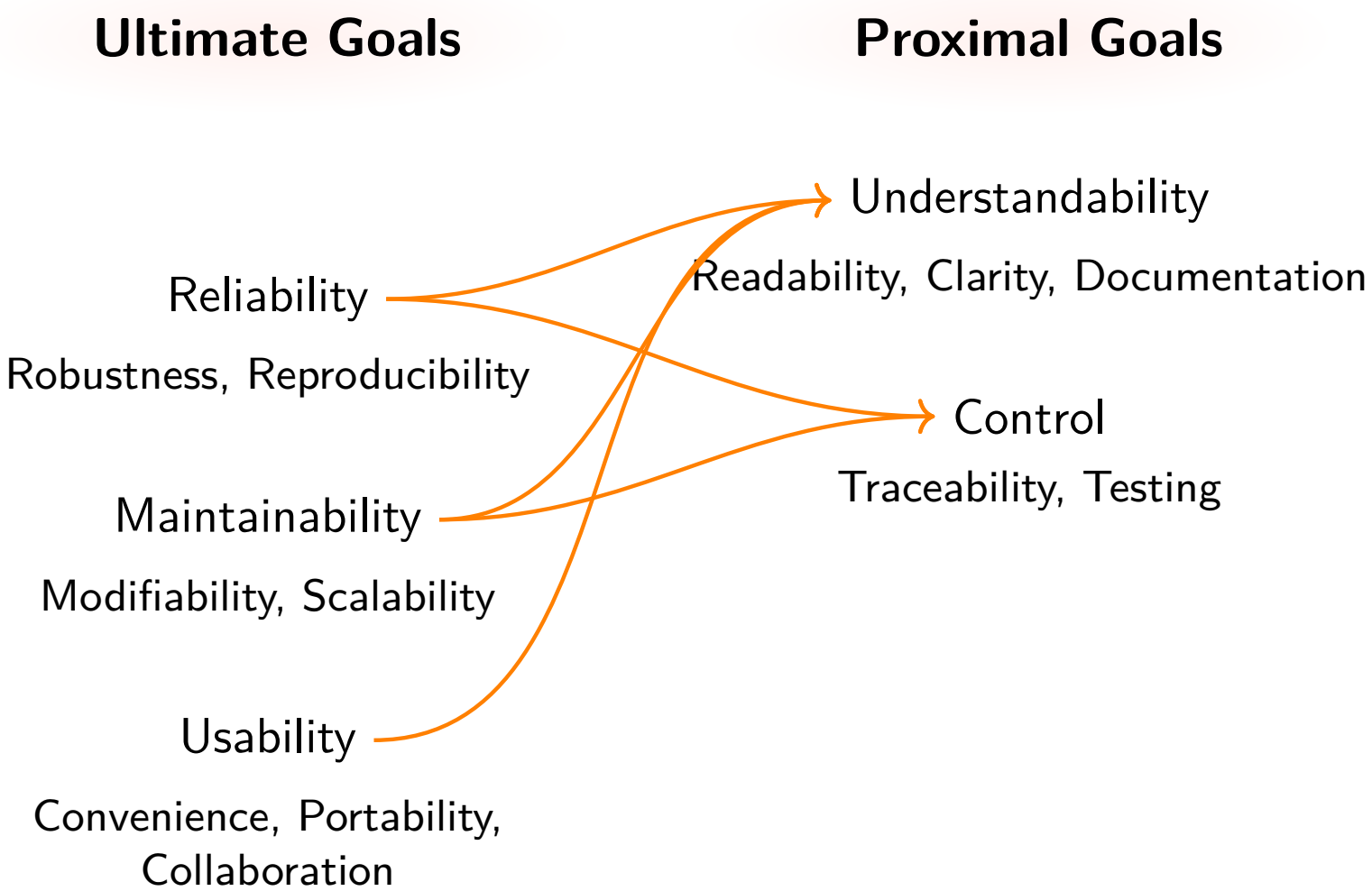
Convenience, Portability, Collaboration

Proximal Goals

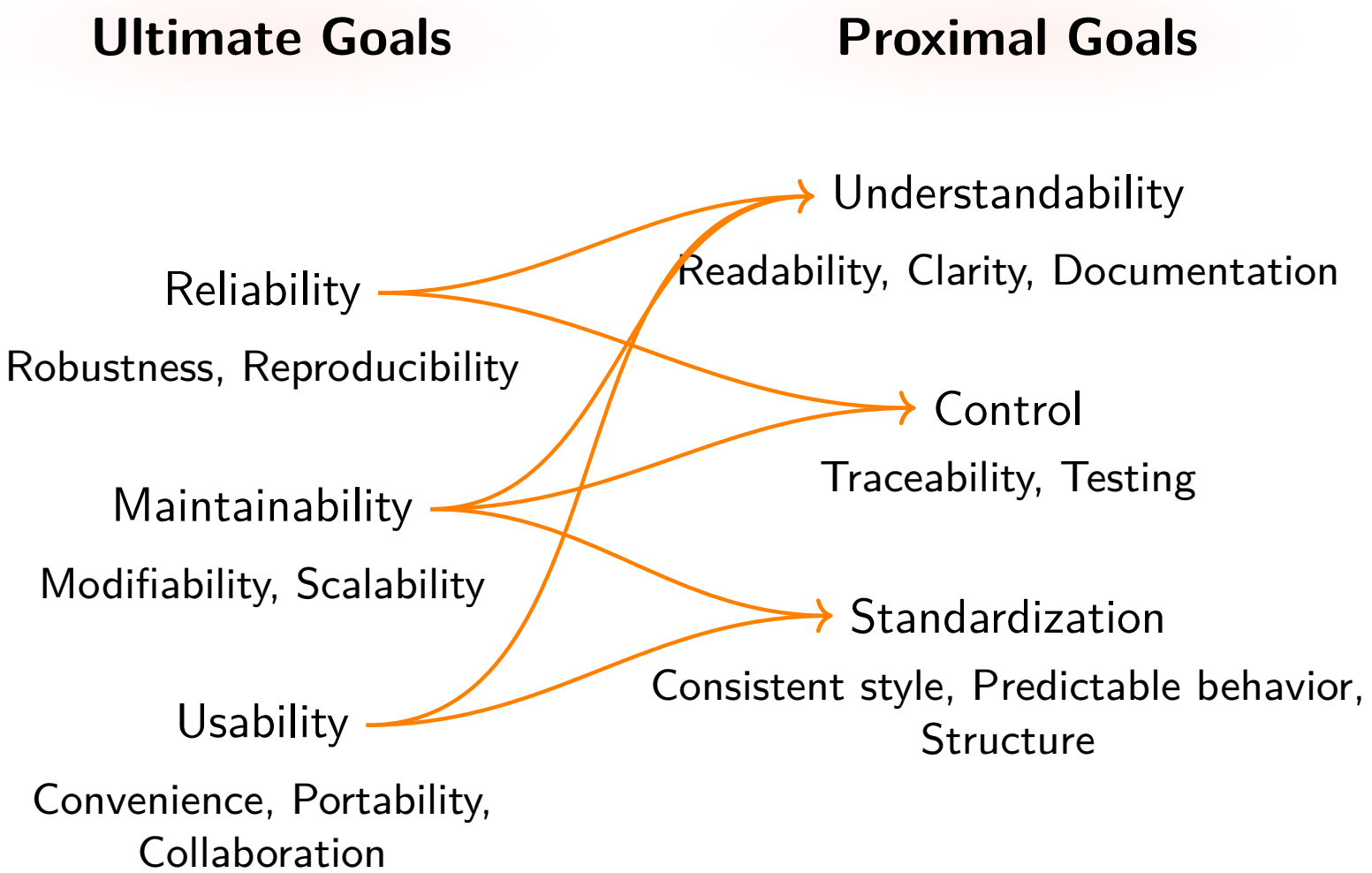
Quality Criteria



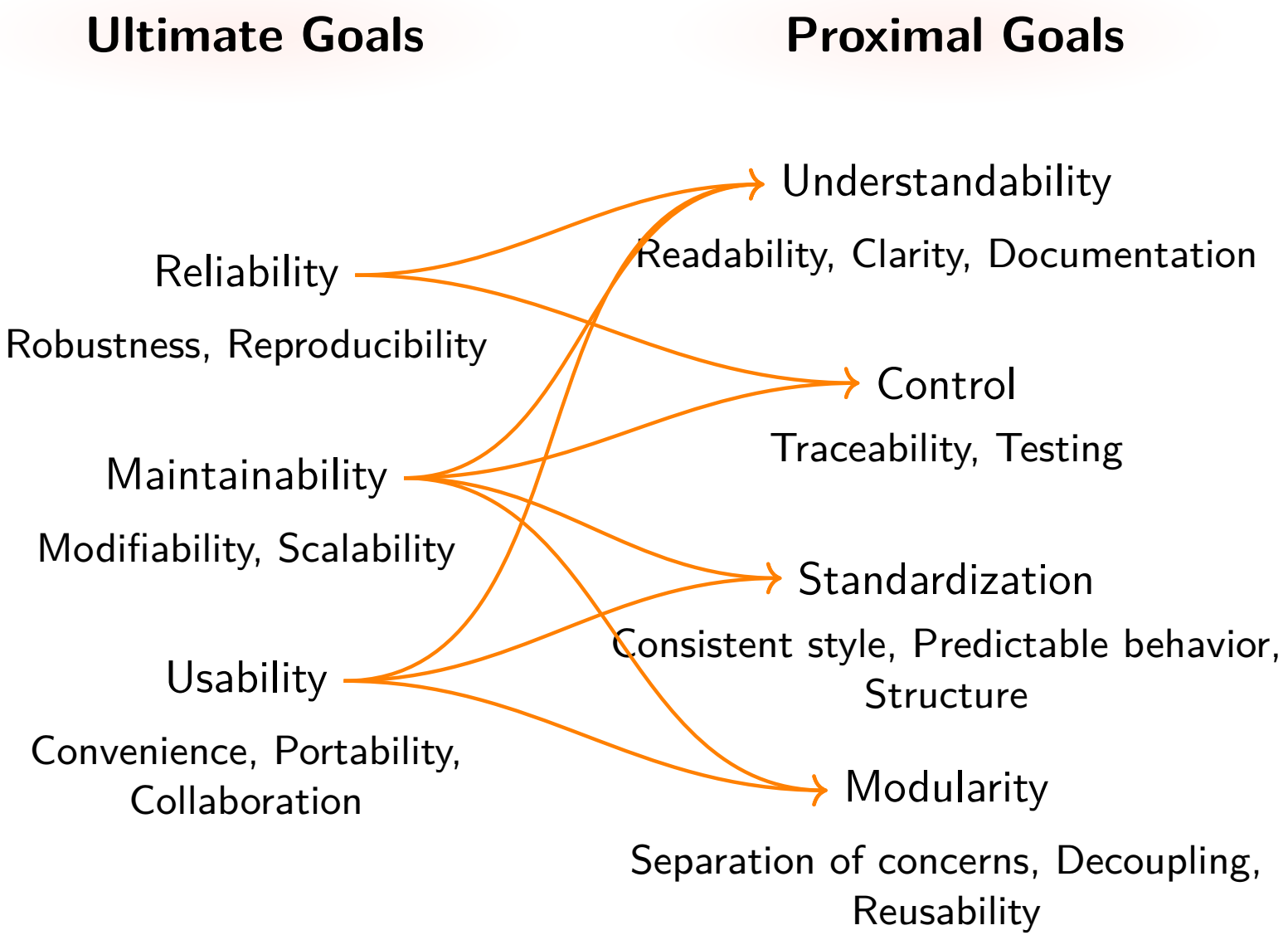
Quality Criteria



Quality Criteria



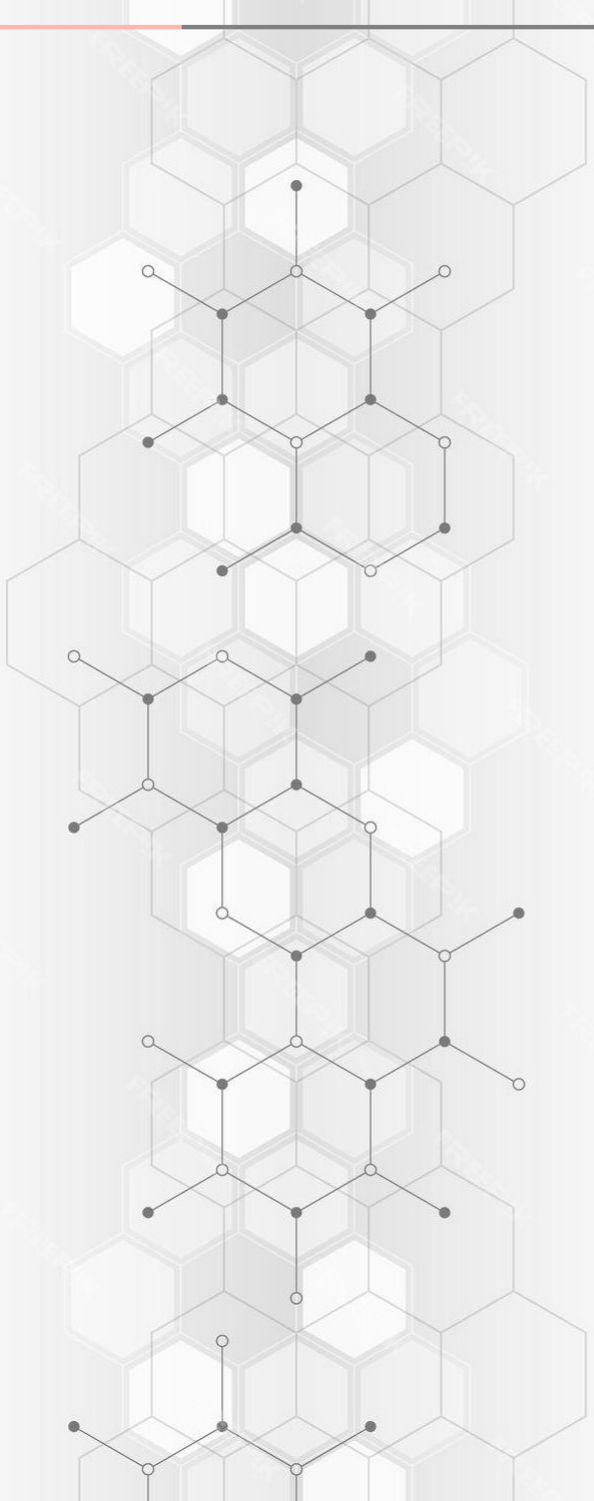
Quality Criteria



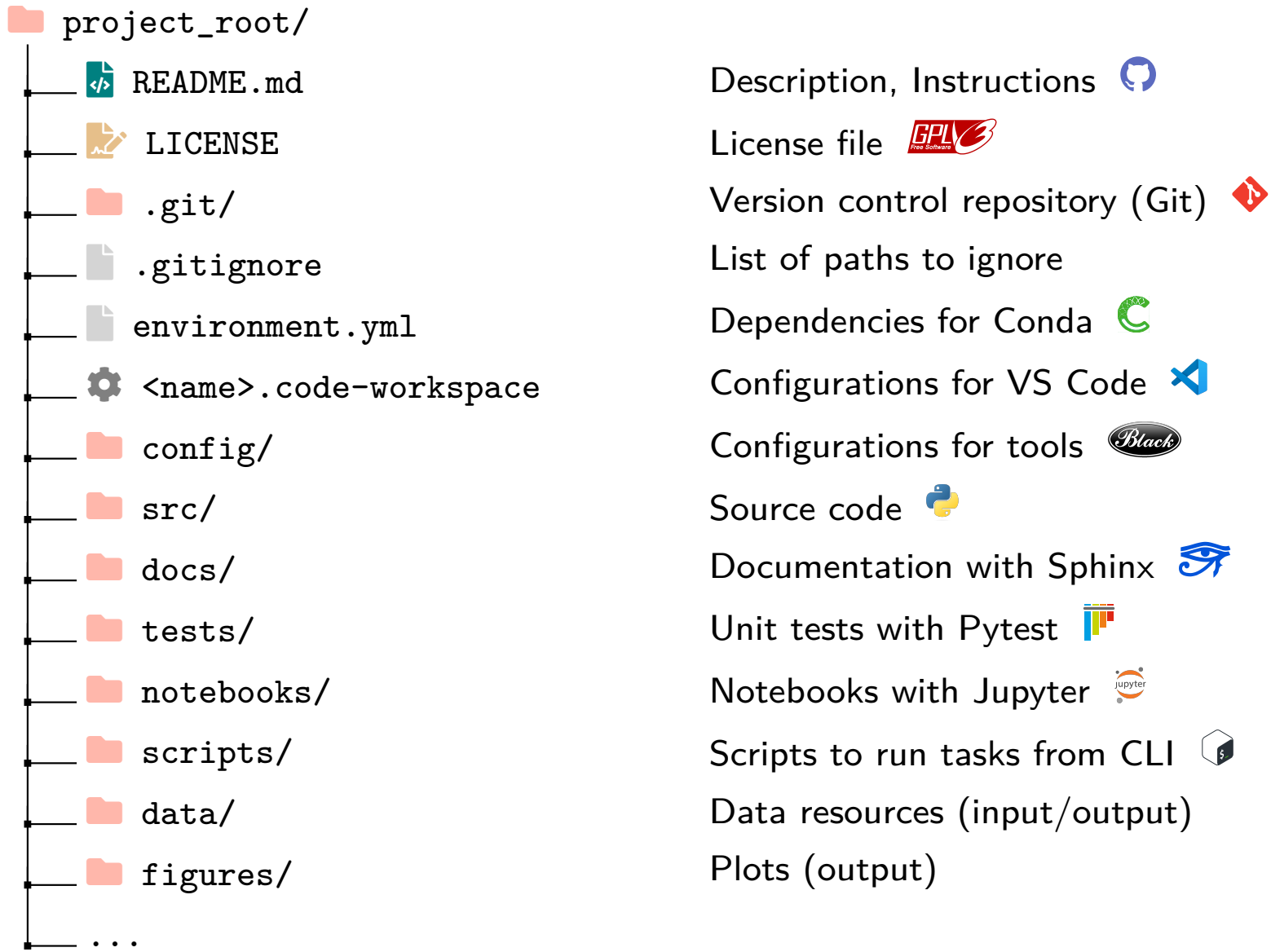
Collaboration

With others

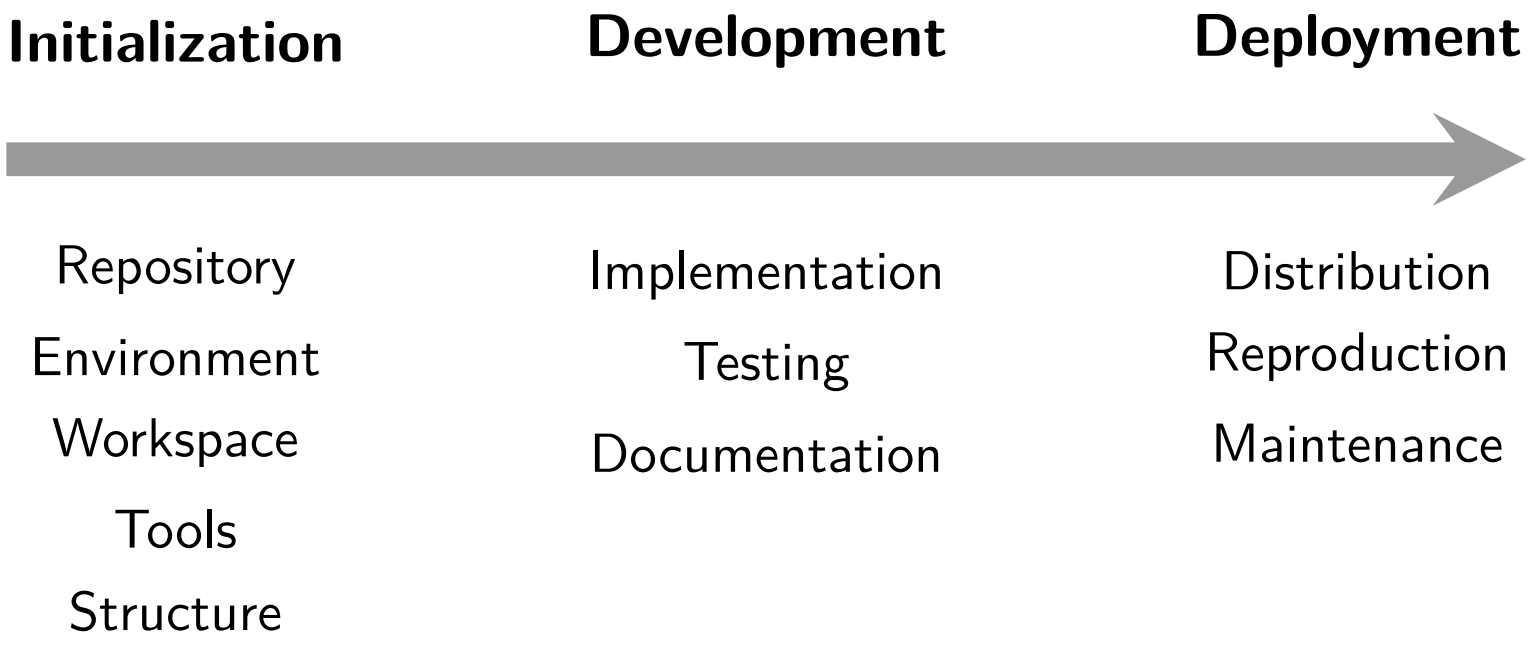
With oneself in the future

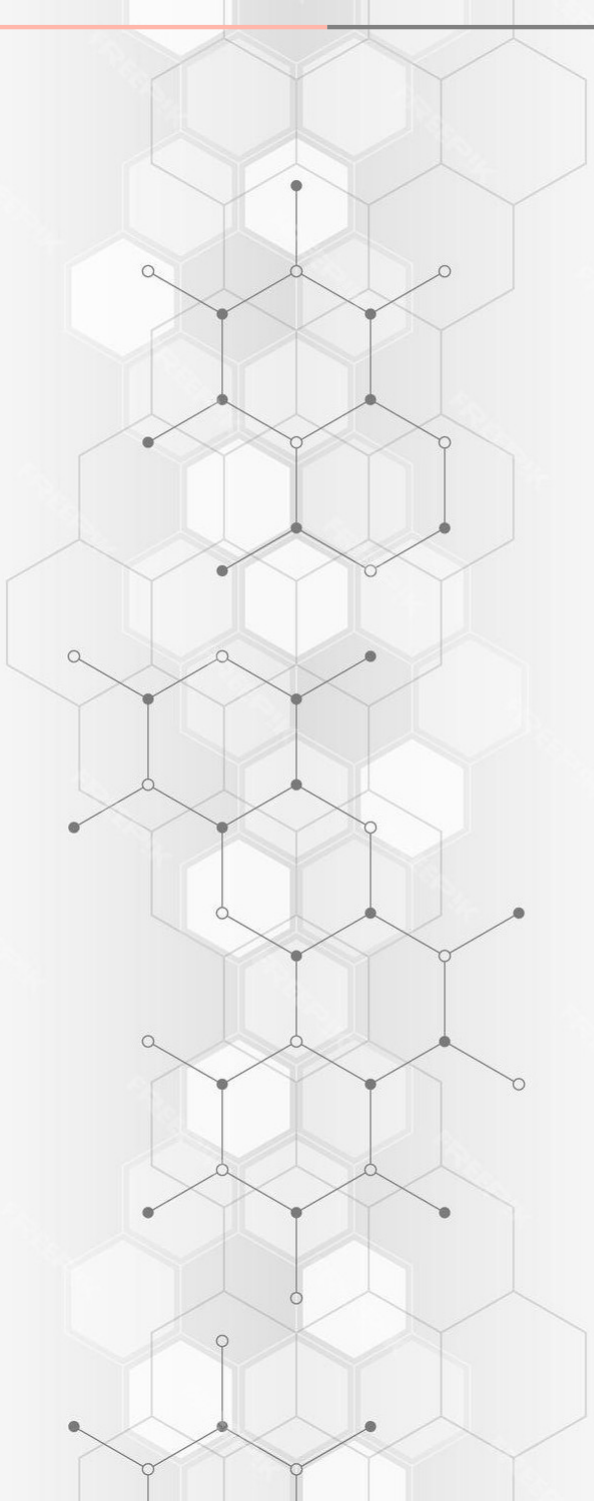


- Objectives
- Overview: Structure, Tools, Steps
- Initializing a Project


project_root/
README.md
Description, Instructions
LICENSE
License file
.git/
Version control repository (Git)
.gitignore
List of paths to ignore
environment.yml
Dependencies for Conda
<name>.code-workspace
Configurations for VS Code
config/
Configurations for tools
src/
Source code
docs/
Documentation with Sphinx
tests/
Unit tests with Pytest
notebooks/
Notebooks with Jupyter
scripts/
Scripts to run tasks from CLI
data/
Data resources (input/output)
figures/
Plots (output)
...

Project Life Cycle





- Objectives
- Overview: Structure, Tools, Steps
- Initializing a Project

Initialization Workflow

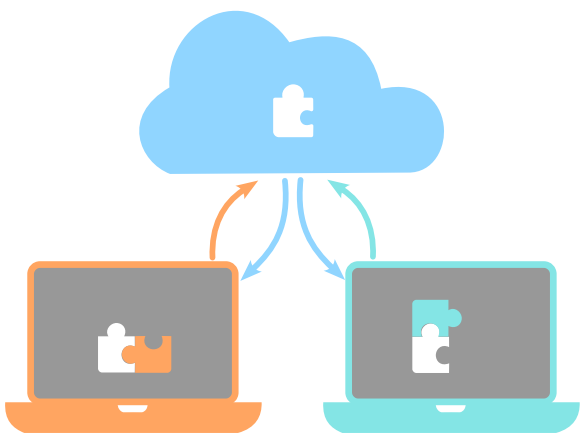
Initialization **Development** **Deployment**

- **Create Remote and Local Repositories**
- Establish Rules for the Project
- Construct the Project Structure
- Create a Virtual Environment
- Create a Code Workspace

Initialization Workflow

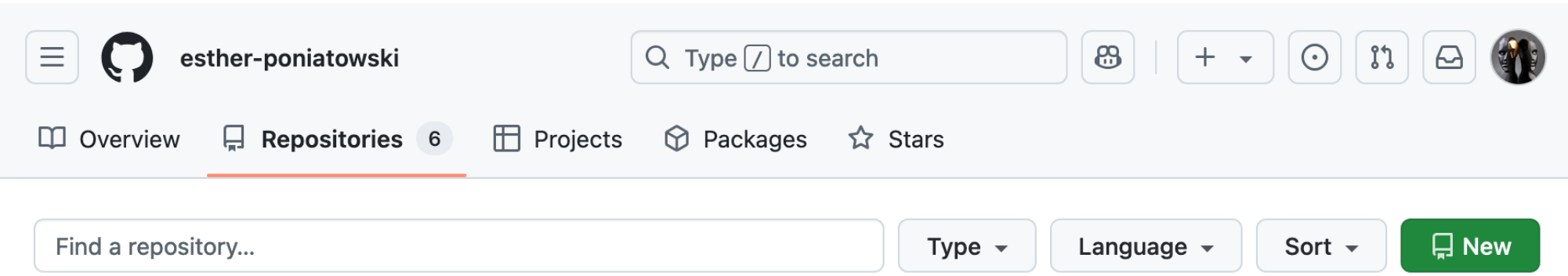
Create Remote and Local Repositories

Goal	Version-control, Tracking changes, Backup
Importance	Reliability, Traceability, Standardization
Tools	Git GitHub



Create Remote and Local Repositories

Create the remote repository on GitHub (`origin`)



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Required fields are marked with an asterisk (*).

Owner * **Repository name ***

esther-poniatowski / ProjectName

ProjectName is available.

Great repository names are short and memorable. Need inspiration? How about **literate-train** ?

Description (optional)

- Public
  Anyone on the internet can see this repository. You choose who can commit.
- Private
  You choose who can see and commit to this repository.

Initialize this repository with:

- [x] Add a README file
  This is where you can write a long description for your project. Learn more about READMEs.

Add .gitignore

.gitignore template: None

Choose which files not to track from a list of templates. Learn more about ignoring files.

Choose a license

License: GNU General Public License v3.0

A license tells others what they can and can't do with your code. Learn more about licenses.

This will set `main` as the default branch. Change the default name in your settings.

You are creating a public repository in your personal account.

Create repository

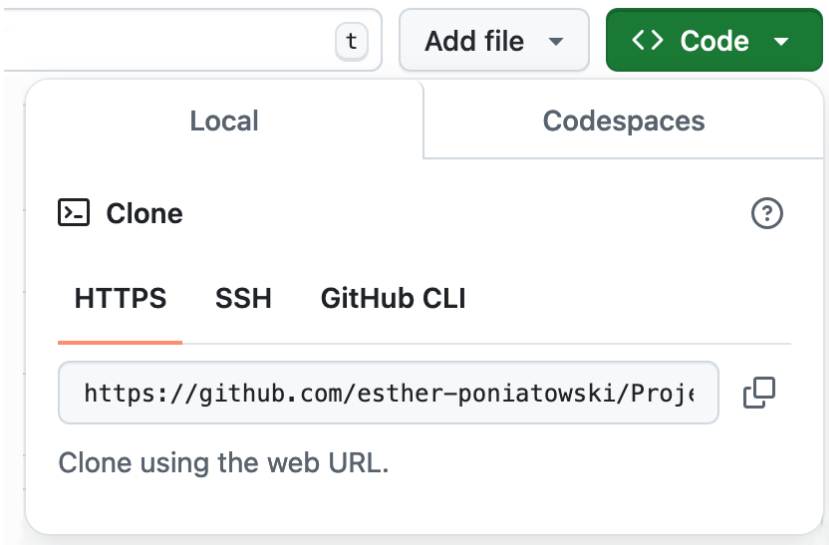
Create Remote and Local Repositories

Clone the repository locally

Navigate to the location where the project should reside
(e.g. `$HOME/Projects`)

```
$ git clone 'https://github.com/user/repo.git'
```

Copy-paste the URL from the GitHub repository page



Outcomes

- Copy of the remote repository in the current working directory.
- Name: Similar to the cloned repository (by default).
- Content: All the repository files and history (`.git/`).
- Automatic association with the remote repository (`origin`) for fetch and pull operations.

```
$ git remote -v
```

```
origin 'https://github.com/esther-poniatowski/ProjectTemplate.git' (fetch)
origin 'https://github.com/esther-poniatowski/ProjectTemplate.git' (push)
```

Notes

- Both repository become two independent *branches* of the project. They inherit the same commit history, but can now diverge.
- The `.git/` folder is not visible in GitHub.

Create Remote and Local Repositories

Configure the local Git repository

Local Settings	Global Settings
Repository-specific	Apply to all repositories, if not overridden locally
Stored in `.git/config` (project folder)	Stored in `/.gitconfig`
Commands with `--local` flag or none	Commands with `--global` flag

Create Remote and Local Repositories

Configure the Local Repository

General Syntax

```
$ git config --local <key> <value>
```

`<value>`: Either omit to view the current setting, or specify to update it.
`--local`: Optional flag for precising the scope (local by default).

Check user profile (used to attribute commits):

```
$ git config user.name
$ git config user.email
```

Update if necessary:

```
$ git config user.name "Example Name"
$ git config user.email "exampleemail@domain.com"
```

Create Remote and Local Repositories

Set the Credentials

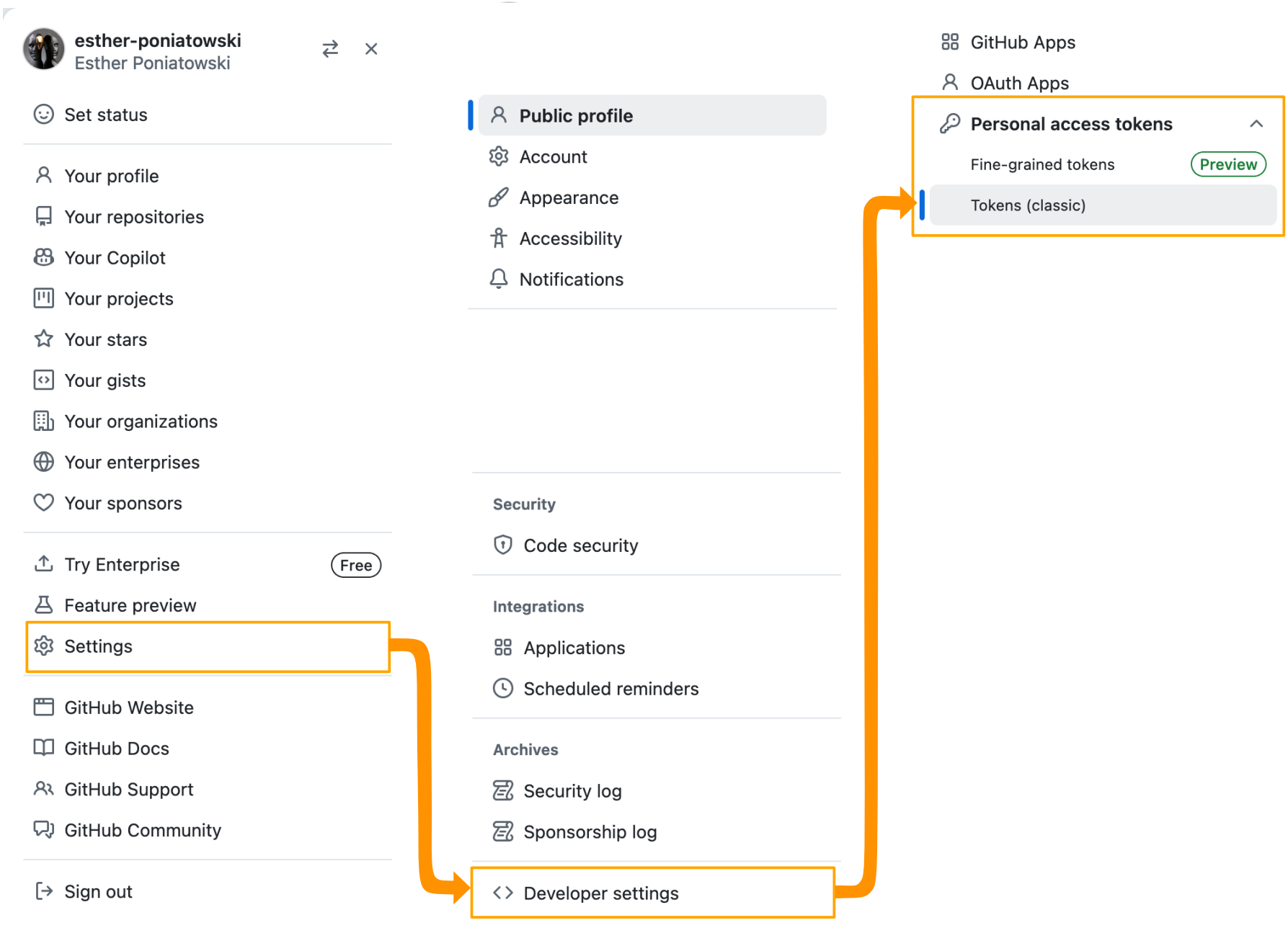
Warning

Pushing local changes to the remote repository requires authentication:

```
$ git push
Username for 'https://github.com': ...
```

Alternative approach (among others):

Personal access token


esther-poniatowski
Esther Poniatowski
Set status
Your profile
Your repositories
Your Copilot
Your projects
Your stars
Your gists
Your organizations
Your enterprises
Your sponsors
Try Enterprise
Free
Feature preview
Settings
GitHub Website
GitHub Docs
GitHub Support
GitHub Community
Sign out
Public profile
Account
Appearance
Accessibility
Notifications
Security
Code security
Integrations
Applications
Scheduled reminders
Archives
Security log
Sponsorship log
Developer settings
GitHub Apps
OAuth Apps
Personal access tokens
Fine-grained tokens
Preview
Tokens (classic)

New fine-grained personal access token Preview

Create a fine-grained, repository-scoped token suitable for personal API use and for using Git over HTTPS.

Token name *

ProjectNameToken

A unique name for this token. May be visible to resource owners or users with possession of the token.

Resource owner

esther-poniatowski

Expiration *

30 days The token will expire on Tue, Jan 7 2025

- 7 days
- ✓ 30 days
- 60 days
- 90 days
- Custom...
- No expiration

What is this token for?

Repository access

- ○ **Public Repositories (read-only)**
- ○ **All repositories**
  This applies to all current *and* future repositories you own.
  Also includes public repositories (read-only).
- ◉ **Only select repositories**
  Select at least one repository. Max 50 repositories.
  Also includes public repositories (read-only).

Select repositories

ProjectTemplate

esther-poniatowski/**ProjectTemplate**
no description

Permissions

Read our permissions documentation for information about specific permissions.

Repository permissions

Repository permissions permit access to repositories and related resources.

Actions
Workflows, workflow runs and artifacts.
Access: No access

Administration
Repository creation, deletion, settings, teams, and collaborators.
Access: No access

Attestations
Create and retrieve attestations for a repository.
Access: No access

Code scanning alerts
View and manage code scanning alerts.
Access: No access

Codespaces
Create, edit, delete and list Codespaces.
Access: No access

Codespaces lifecycle admin
Manage the lifecycle of Codespaces, including starting and stopping.
Access: No access

Codespaces metadata
Access Codespaces metadata including the devcontainers and machine type.
Access: No access

Codespaces secrets
Restrict Codespaces user secrets modifications to specific repositories.
Access: No access

Contents
Repository contents, commits, branches, downloads, releases, and merges.
Access: Read and write

Select an access level
- No access
- Read-only
- ✓ Read and write

Custom properties
View and set values for a repository's custom properties, when allowed by th

Dependabot alerts
Retrieve Dependabot alerts.

Create Remote and Local Repositories

Set the Credentials

.git-credentials

Tool	GitHub
Type	Plain Text
Location	./ (project root)
Content	Credentials for the remote repository.

```
1 https://<username>:<token>@github.com
```

Examples

```
1 https://esther-poniatowski:PERSONAL_ACCESS_TOKEN@github.com
```

Create Remote and Local Repositories

Set the Credentials

Configure the local repository to use the credentials stored in the `.git-credentials` file:

```
$ git config credential.helper 'store --file=.git-credentials'
```

Outcomes

- For push operation, authentication is performed automatically, so the user is not required to enter the personal access token manually.

Initialization Workflow

Initialization **Development** **Deployment**

- Create Remote and Local Repositories
- **Establish Rules for the Project**
- Construct the Project Structure
- Create a Virtual Environment
- Create a Code Workspace

Initialization Workflow

Establish Rules for the Project

Goal	Impose practices for using the project (instructions, guidelines, templates...)
Importance	Standardization, Collaboration
Tools	Git  GitHub 

Key Files

`README.md`, `.gitignore`, `.git-message`

Create a Repository

Give Project Information and Instructions

README.md

Tool	GitHub
Type	Markdown
Location	./ (project root)
Content	Project description Authors and contact information Installation and setup instructions Usage Contribution guidelines ...

````
# Project Title

## Description

Purpose, Background, Features...

## Authors

Names and contact information: Author Name <email>

## Installation

Instructions to create a local environment.

1. Step 1
```bash
$ command [options] arguments
```

## Usage

Main commands, functions, or features.

Code example with expected results:
```python
print("Example Command")
```

## Contributing

Guidelines for submitting well-formed changes to the project.
````

Create a Repository

Specify the Ignored Patterns

.gitignore

Tool	Git
Type	Plain Text
Location	`./` (project root)
Content	List of files and directory patterns to exclude from version control: private information (credentials), data, temporary files, build artifacts...

Syntax Rules

Warning

Patterns are **relative** to the location of the `.gitignore` file.

Plain text Match exact file or directory names, in any directory.

```
# Ignore credential files in any location
credentials.txt
```

Slash (/) At the end, match directories and their contents. At the beginning, match files in the *root directory only*.

```
# Ignore the "data" directory and its contents
data/
```

Single asterisk (*) Match any character (0 or more).

```
# Ignore any file with a specific extension
*.tmp
```

Double asterisk (**) Match any number of directories, recursively.

```
# Ignore all PDF files in the "docs" directory and its
    subdirectories
docs/**/*.pdf
```

Question mark (?) Match any *single* character.

```
# Ignore files with a single-character suffix
file?.txt
```

Brackets ([]) Match any character inside the brackets, or a range of characters.

```
# Ignore files with specific-letter suffixes (A, B, or C)
file[ABC].txt

# Ignore files with a single-digit number suffix
file[1-9].txt
```

Exclamation mark (!) Negate a pattern, i.e. include a pattern that would otherwise be ignored.

```
# Ignore all data files except CSV files
data/*
!data/*.csv
```

Create a Repository

Establish a Commit Message Template

.gitmessage

Tool	Git
Type	Plain Text
Location	./ (project root)
Content	Template for commit messages. Standardized format across contributors.

Create a Repository

Establish a Commit Message Template

General Structure

```
prefix(scope): Title (< 50 characters) ---------

Description: What and Why (not How) (< 72 characters per line)

Ref-Keyword: #ID

Signed-off-by: User Name <user.name@example.com>
```

Proposed Template

```
_():

Signed-off-by: User Name <user.name@example.com>
```

Prefixes for Commit Types

Conventional keywords to categorize the changes introduced by a commit. They can be used as filters to search the commit history.

`fix`	Correct a bug.
`feat`	Introduce a new feature or functionality.
`refact`	Refactor the code without changing its external behavior.
`test`	Add or update tests.
`docs`	Add or update documentation.
`style`	Improve formatting.
`perf`	Enhance performance.
`build`	Affect build components.
`chore`	Miscellaneous: manage dependencies, configure tools...
`revert`	Reverts a previous commit.
`merge`	Merge a branch.

Ref Keywords

Keywords recognized by the issue tracker on GitHub, to link commits to issues and to trigger actions (if applicable).

Keywords for Closing Issues

Automatically close an issue upon merging into the default branch.

`close`, `closes`, `closed` `fix`, `fixes`, `fixed`
`resolve`, `resolves`, `resolved`

Keywords for Referencing Issues

Provides context about the task addressed by the commit.

`references`, `refs` `related to` `see`
`duplicate of`

Examples

```
fix(compute_sum): Correct calculation of `chechsum`

Handle edge case where `a=0`.

Fixes: #123

Signed-off-by: Esther Poniatowski <eponiatowski@ens.psl.eu>
```

```
docs(README): Correct typos

- Indentation in lists in "Notes" section
- Name change: `batchLogbatchLog` -> `batchLog`

Signed-off-by: Esther Poniatowski <eponiatowski@ens.psl.eu>
```

```
feat(parser): Add support for .m files (Matlab)

Adapt the pipeline from the entry point to the `parse` function.

Signed-off-by: Esther Poniatowski <eponiatowski@ens.psl.eu>
```

Create a Repository

Establish a Commit Message Template

Retrieve the template from the remote repository and set it up locally:

```
$ git pull origin main
$ git config --local commit.template .gitmessage
```

Outcomes

- When committing, an editor opens and displays the template to edit (when running `git commit` without the `-m` option).

Notes

- When committing directly in the *remote repository* (i.e. from the GitHub page), the template is not automatically pre-filled