Genomes Workflow - LBCM

01 - Introduction

General guidelines

Wednesday 9th July, 2025

1 Overview

1.1 Module Objectives

Learning goals

This module covers:

- Repository organization.
- Available scripts.
- General pipeline structure.
- Modules organization.

2 Background

2.1 General Context

Why this matters

Key concepts:

- The integration between biological knowledge and IT or bioinformatics tools can be harsh.
- The presence and interference of multiple key concepts of each area creates a big wall on newcomers.
- Explain key features and provide support in a organized way can help to circumvent this initial barrier.

3 Workflow & Methods

3.1 The repository

Key parameters

The repository is structured aiming to facilitate user operation and base codes and modules maintenance. The main separation happens in the global folder, that stores templates, bibliography files and scripts for project management.

The base skeleton of the repository can be seen below:

```
.
|-- README.md
|-- global
|   |-- bibliography
|   |   `-- master.bib
|   |-- scripts
|   |   |-- main.py
|   |   `-- tools
|   |       |-- __init__.py
|   |       |-- create_module.py
|   |       `-- set_vim_config.py
|   `-- templates
|       |-- README_template.md
|       |-- enhanced_vimtex_config.vim
|       |-- gilles_snippets.snippets
|       |-- main_template.tex
|       `-- preamble_template.tex
|-- modules
```

```
|   `-- 01_introduction
|       |-- README.md
|       |-- images
|       |-- main.tex
|       |-- preamble.tex
|       `-- references.bib
|-- requirements.txt
|-- scripts
|   |-- README.md
|   |-- build_course.py
|   |-- config.yaml
|   |-- interactive_builder.py
|   |-- main.py
|   |-- requirements.txt
|   |-- tools
|   |   `-- __init__.py
|   `-- utils
|       |-- __init__.py
|       |-- file_manager.py
|       |-- latex_compiler.py
|       |-- module_validator.py
|       `-- pdf_merger.py
`-- setup.py
```

3.2 Available scripts

Currently, there are two distinct scripts folders. The first, located inside the global folder, contains py scripts focused on project maintenance, while the second receives all py scripts user-oriented.

3.2.1 Global - Scripts

main.py Presents a selection menu to all available scripts inside the tools folder.

create_module.py Allows the automatic creation of a new Module folder. Includes the basic structure, with a README file, main and preamble tex files, images folder and a local references.bib file.

set_vim_config.py Change the user .vimrc file to a custom pre-made one, that can be found inside the templates folder. Allows VimTeX and UltiSnips integration, alongside keyboard shortcuts for easier navigation.

3.2.2 Main folder - Scripts

Example 3.1. Practical example with real genomic data.

4 Practical Examples

4.1 Example 1: Basic Analysis

Input/output files

- **Input:** Sample data description
- **Command:** Based on approach from **example2024**

- **Output:** Expected results and file formats

```
# Example command with typical genomic data
tool_name -i input_file.fasta -o output_file.txt --parameter
    value
```

Listing 1: Basic command example

4.2 Example 2: Advanced Usage

Complex parameters

```
# Multi-step analysis pipeline
step1_tool input.fasta | step2_tool --param1 value1 >
    intermediate.txt
step3_tool intermediate.txt --param2 value2 -o final_result.txt
```

Listing 2: Advanced analysis pipeline

5 Results & Interpretation

5.1 Output Files

Common output formats and their interpretation:

File formats

- **Format 1:** Description and typical contents
- **Format 2:** When and how to use this output
- **Quality metrics:** How to assess result quality

Remark 5.1. Important note about result interpretation following **author2024**.

6 Scripts & Code

6.1 Helper Scripts

```
#!/usr/bin/env python3
"""
Helper script for genomic data processing
Usage: python script.py input.fasta output.txt
"""

def process_sequences(input_file, output_file):
    """Process genomic sequences"""
    with open(input_file, 'r') as f:
        sequences = f.read()

    # Processing logic here
    processed = sequences.upper()

    with open(output_file, 'w') as f:
        f.write(processed)

if __name__ == "__main__":
    import sys
    process_sequences(sys.argv[1], sys.argv[2])
```

Listing 3: Data processing script

6.2 Quality Control

```
#!/bin/bash
# Quality control pipeline for genomic data

# Check file format
file_format_check.py $INPUT_FILE

# Basic statistics
sequence_stats.py $INPUT_FILE > stats.txt

# Quality assessment
quality_assessment_tool $INPUT_FILE --output qc_report.html
```

Listing 4: QC pipeline

7 Troubleshooting & Best Practices

7.1 Common Issues

Error solutions

- **Memory errors:** Reduce dataset size or increase available RAM
- **Format issues:** Check input file formatting and encoding
- **Parameter tuning:** Guidelines for optimization

7.2 Best Practices

- **Data backup:** Always keep original data copies
- **Version control:** Track analysis versions and parameters
- **Documentation:** Record all analysis steps and decisions
- **Reproducibility:** Use consistent environments and seeds

8 References

- Key papers: **example2024**; **author2024**; **smith2024**
- Software documentation: [Tool official docs]
- Related modules: [Other workflow modules]

9 Exercises & Next Steps

- **[TODO]: Practice with provided sample data**
- **[TODO]: Try different parameter settings**
- **Apply to your own genomic dataset**
- **[TODO]: Explore advanced features**

10 Research Notes

Additional observations and module-specific notes...

Key insight: Connection between this tool and genome annotation pipeline

[IDEA]: Extension: Integration with other bioinformatics tools in the workflow