Bioinformatics Quality Management System

Ben Rambo-Martin, PhD
Lead Bioinformatics Scientist
US CDC – Influenza Division

Disclaimer

The findings and conclusions in this presentation are those of the authors and do not necessarily represent the official position of the Centers for Disease Control and Prevention.
Use of trade names and commercial sources is for identification only and does not imply endorsement by the U.S. Department of Health and Human Services.
References to non-CDC sites on the Internet do not constitute or imply endorsement of these organizations or their programs by CDC or the U.S. Department of Health and Human Services. CDC is not responsible for the content of pages found at these sites.

Objectives

- Describe quality management system (QMS) components
  - Reproducibility
  - Version control
  - Documentation
- Apply QMS principles to NGS workflows
  - Pre-analytical
  - Analytical
- Common troubleshooting

Reproducibility

- Same input + same methods = same results
  - Core principle of computational (all) science

- Record software versions
  - Aligners, samtools, variant callers, workflow managers
- Track changes with version control
  - Git for scripts and pipeline development
- Document parameters and references
  - Primer schemes
  - Quality thresholds

- Identify test data to run through new versions and pipelines

- Use workflow managers
  - Snakemake, Nextflow for structured, repeatable runs
- Leverage containers
  - Docker or Singularity ensure consistent environments

Reproducibility

- Same input + same methods = same results
  - Core principle of computational (all) science

- Record software versions
  - Aligners, samtools, variant callers, workflow managers
- Track changes with version control
  - Git for scripts and pipeline development
- Document parameters and references
  - Primer schemes
  - Quality thresholds

- Identify test data to run through new versions and pipelines

- Use workflow managers
  - Snakemake, Nextflow for structured, repeatable runs
- Leverage containers
  - Docker or Singularity ensure consistent environments

Reproducibility

- Same input + same methods = same results
  - Core principle of computational (all) science

- Record software versions
  - Aligners, samtools, variant callers, workflow managers
- Track changes with version control
  - Git for scripts and pipeline development
- Document parameters and references
  - Primer schemes
  - Quality thresholds

- Identify test data to run through new versions and pipelines

- Use workflow managers
  - Snakemake, Nextflow for structured, repeatable runs
- Leverage containers
  - Docker or Singularity ensure consistent environments

Reproducibility

- Same input + same methods = same results
  - Core principle of computational (all) science

- Record software versions
  - Aligners, samtools, variant callers, workflow managers
- Track changes with version control
  - Git for scripts and pipeline development
- Document parameters and references
  - Primer schemes
  - Quality thresholds

- Identify test data to run through new versions and pipelines

- Use workflow managers
  - Snakemake, Nextflow for structured, repeatable runs
- Leverage containers
  - Docker or Singularity ensure consistent environments

Documentation

- Explains how tools and pipelines work
  - Inputs, outputs, parameters, and assumptions
- Enables reproducibility
  - Others (and future you) can rerun the same analysis
- Reduces errors and misuse
  - Clear defaults and examples prevent incorrect runs
- Essential for collaboration
  - Shared understanding across labs, teams, and institutions

Documentation

- Explains how tools and pipelines work
  - Inputs, outputs, parameters, and assumptions
- Enables reproducibility
  - Others (and future you) can rerun the same analysis
- Reduces errors and misuse
  - Clear defaults and examples prevent incorrect runs
- Essential for collaboration
  - Shared understanding across labs, teams, and institutions

Documentation

- Explains how tools and pipelines work
  - Inputs, outputs, parameters, and assumptions
- Enables reproducibility
  - Others (and future you) can rerun the same analysis
- Reduces errors and misuse
  - Clear defaults and examples prevent incorrect runs
- Essential for collaboration
  - Shared understanding across labs, teams, and institutions

Documentation

- Explains how tools and pipelines work
  - Inputs, outputs, parameters, and assumptions
- Enables reproducibility
  - Others (and future you) can rerun the same analysis
- Reduces errors and misuse
  - Clear defaults and examples prevent incorrect runs
- Essential for collaboration
  - Shared understanding across labs, teams, and institutions

Best Practices in writing documentation

- README.md files
  - Overview, requirements, and basic usage
- Usage examples & command snippets
  - Copy-pasteable starting points
- Parameter descriptions
  - What flags do, expected values, and defaults
- Workflow diagrams
  - Visualize steps, inputs, and outputs

Best Practices in writing documentation

- README.md files
  - Overview, requirements, and basic usage
- Usage examples & command snippets
  - Copy-pasteable starting points
- Parameter descriptions
  - What flags do, expected values, and defaults
- Workflow diagrams
  - Visualize steps, inputs, and outputs

Best Practices in writing documentation

- README.md files
  - Overview, requirements, and basic usage
- Usage examples & command snippets
  - Copy-pasteable starting points
- Parameter descriptions
  - What flags do, expected values, and defaults
- Workflow diagrams
  - Visualize steps, inputs, and outputs

Best Practices in writing documentation

- README.md files
  - Overview, requirements, and basic usage
- Usage examples & command snippets
  - Copy-pasteable starting points
- Parameter descriptions
  - What flags do, expected values, and defaults
- Workflow diagrams
  - Visualize steps, inputs, and outputs

Best Practices in writing documentation

- Document data formats
    - FASTQ, FASTA, BAM, reference versions
- Record software versions
    - Tools, containers, workflow managers
- Include example datasets
    - Small test cases to validate setup

Best Practices in using documentation

- Start with the README
  - Understand the purpose, inputs, outputs, and limitations
- Check versions and dates
  - Ensure the documentation matches the tool or pipeline version you're using
- Follow the example first
  - Run the provided test or example before real data
- Read parameter descriptions carefully
  - Defaults may not match your experiment
- Verify input formats
  - FASTQ, FASTA, BAM, sample sheets, reference builds
- Look for expected outputs
  - Confirm files and directories match the documented results

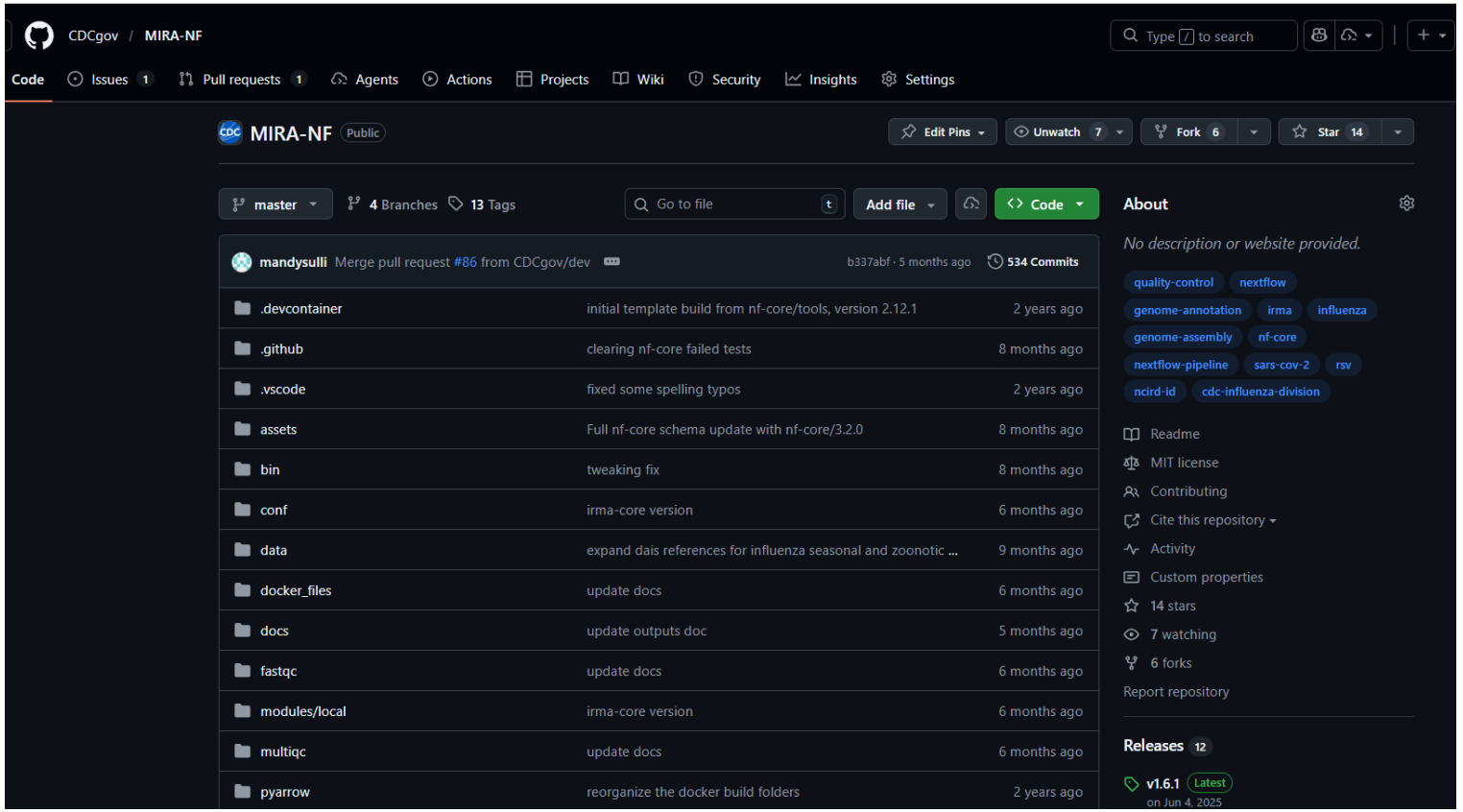
Git

- Git is a version control system
  - Tracks changes to files over time
- Designed for collaboration
  - Multiple people can work on the same codebase
  - Think of it like a shared document (Microsoft office, Google docs)
- Keeps a history of your work
  - See what changed, when, why, and by whom
- Common in bioinformatics
  - Pipelines, scripts, documentation, and workflows

- Gitlab and Github most used

Core Git concepts

- Repository (repo)
  - A project tracked by Git
- Commit
  - A saved snapshot of changes with a message
- Branch
  - A parallel version of the code for development or testing
- Remote repository
  - Shared copy on platforms like GitHub or GitLab




```
qgx6@rosalind02:MIRA-NF$ git status
On branch dev
Your branch is up to date with 'origin/dev'.

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

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

Git

- Common Git Commands (User Level)
- Create or get a repository
  - git init — start a repo
  - git clone <repo> — copy an existing repo
- Track and save changes
  - git status — see changes
  - git add <file> — stage changes
  - git commit -m "message" — save snapshot
- Sync with others
  - git pull — get updates
  - git push — share your changes