**WORKFLOW NOTES**

## REITER ET AL., 2021

### INTRO

* Biological research community committed to FAIR practices (all life sciences research should be):
  + F: findable
  + A: accessible
  + I: interoperable
  + R: reusable
* Workflow systems needs to specify this info at each step:
  + Input (or types of inupt) needed for that step
  + Output (or types of output) being produced

### GETTING STARTED WITH WORKFLOWS

* Research workflows: under iterative development to answer novel scientific questions
* Production workflows: have reached maturity and are used to run standard analysis on new samples
* USING WORKFLOW WITHOUT LEARNING WORKFLOW SYNTAX
  + Existing websites like Galaxy, Cavatica, etc can allow workflow operation without using command line systems
  + There are also ways to use workflow systems in command line applications
    - Examples: RHNA-seq, PiGX, ATLAS, Sunbeam, dammit eukaryotic transcriptome annotation pipeline, elvers de novo transcriptome pipeline

Diagram

Description automatically generated

* CHOOSING A WORKFLOW SYSTEM
  + Four of the most widely-used bioinformative workflow systems
    - Snakemake
    - Nextflow
    - Common Workflow Language (CWL)
    - Workflow Description Language (WDL)

### WRANGLING SCIENTIFIC SOFTWARE

* Module system
  + An organized approach to installing, managing, keeping track of software and software versions
  + Often overseen by system administrators, available to all users (this has limitations)
* Most workflow managers integrate with software management systems to handle this for the user
  + Ex: conda package manager
    - Not necessarily perfect reproducibility over time
  + Exs: singularity, docker
    - Heavier weight options
    - Wrap software environments that can capture and reproduce runtime environment information
    - Can be limited by things like changes in dependency installations over time
  + Ex: functional package managers (GNU Guix, Nix, etc):
    - Strictly require all dependency and configuration details be encoded within each software package
    - Limited by steep learning curve; also requires system-wide permissions from administrators usually

### GETTING STARTED WITH SOFTWARE MANAGEMENT

* USING SOFTWARE WITHOUT LEARNING SOFTWARE MANAGEMENT SYSTEMS
  + Integrated development environments (IDE)
    - Examples: PyCharm, RStudio: language-specific
    - Useful for testing a tool before integrating into an analysis
* CHOOSING A SOFTWARE MANAGEMENT SYSTEM
  + A balance between reproducibility capabilities and learning curve
  + Conda may be a good balance between the two for active projects
* INTEGRATING SOFTWARE MANAGEMENT WITHIN WORKFLOWS
  + IDEs/jupyter with conda are useful for developing an analysis strategy and creating an initial workflow
  + Next steps with workflow-integrated software management via conda/singularity/nixOS for executing the resulting workflow on many samples

### WORKFLOW-BASED PROJECT MANAGEMENT

* SYSTEMATICALLY DOCUMENT YOUR WORKFLOWS
  + Good documentation includes EVERYTHING, including:
    - Organization of files and results
    - Clear and commented code
    - Accompanying explanatory documents for design decisions and metadata
* USE CONSISTENT, SELF-DOCUMENTING NAMES
  + For files, scripts, variables, workflows, projects, and manuscripts
* STORE WORKFLOW METADATE WITH THE WORKFLOW
  + Hundreds of small decisions: what parameters used and why, how you got something to work, etc
  + Keep this information in an intuitive and easily accessible place
    - Plain text files usually
    - Code itself can also contain comments, links, etc
* DOCUMENT DATA AND ANALYSIS EXPLORTAION USING COMPUTATIONAL NOTEBOOKS
  + Combination of narrative, code, and code output in a single location
* VISUALIZE YOUR WORKFLOW
  + Highest level: flowcharts detailing the relationships between steps
  + Individual steps: graphical representation of the output
* VERSION CONTROL YOUR PROJECT
  + Without version control software: frequent hard drive backups, manually saving different versions of the same file by appending version number or the date onto the end of the file name
  + Version control software (eg: git, mercurial)
  + Online repositories (eg: github, gitlab, bitbucket)
* SHARE YOUR WORKFLOW AND ANALYSIS CODE
  + Can package code with binder, reprozip, whole tale, etc
  + Can create interactive visualization tools with shiny apps, plotly
  + All of these allow others to the run the code in a web-based environment in environments identical to the environment that the original computation was performed

### GETTING STARTED DEVELOPING WORKFLOWS

* Best method:
  + include as much of the analysis as possible within an automated workflow framework
  + use self-documenting names
  + include analysis visualizations
  + keep rigorous documentation alongside your workflow
* START WITH WORKING CODE
  + Use working examples provided as part of the tool documentation or online
  + To model syntax after
* TEST WITH SUBSAMPLED DATA
  + Test your working workflow syntax on a subset of data
  + For example: first X number of lines of a dataset
* DOCUMENT YOUR PROCESS
  + Track changes, explorations, errors
  + Suggested: markdown language
* DEVELOP YOUR WORKFLOW
  + From the working code, iteratively modify and add steps
  + Periodically clear output directors and rerun the entire workflow
* ASSESS YOUR RESULTS
  + Evaluate as you go
  + Consider which aspects can be evaluated to assess each step for expected behavior
* BACK UP EARLY AND OFTEN
* SCALE UP YOUR WORKFLOW
  + May need extensive computing capability or storage
* FIND A COMMUNITY AND ASK FOR HELP AS NECESSARY
  + Many free and accessible tutorials and code out there

### DATA AND RESOURCE MANAGEMENT FOR WORKFLOW-ENABLES BIOLOGY

* MANAGING LARGE-SCALE DATASETS
  + No substitute for good design, etc
* LOOK FOR APPROPRIATE PUBLICLY AVAILABLE DATA
  + Sometimes you don’t even need to collect your own data, your question can be answered with existing datasets
  + Largest repository for raw sequencing data: International Nucleotide Sequence Database Collaboration (INSDC)
* CONSIDER ANALYSIS WHEN GENERATING YOUR OWN DATA
  + Experimental design considerations
    - Sequencing type
    - Sequencing depths per sample
    - Biological replication
  + Buffer
    - Once generated, keep multiple independent backups of the raw data
  + Keep track of as much information as possible
    - Dates and times of sample collection, storage, and extraction
    - Sample names
    - Aberrations that occurred during sampling
    - Kit lot used for extraction
    - Any other sample/sequencing measures possible (helps control for batch effects)
      * Temperature
      * Location
      * Metabolite concentration
      * Name of collector
      * Well number
      * Plate number
      * Machine used

### GETTING STARTED WITH SEQUENCING DATA

* PROTECT VALUABLE DATA
  + Keep a read-only version alongside a workflow
  + Keep multiple backups
  + Data version control
* ENSURE DATA INTEGRITY DURING TRANSFERS
  + Zipping files saves time and space during transfer
  + Checksums can be used to ensure file integrity after transfer
* PERFORM QUALITY CONTROL AT EVERY STEP
  + Look at your files
    - First few and first last lines
    - Checking file size
  + Visualize your data
    - FastQC for raw data
    - Integrative Genome Viewer for processed sequence data
    - Plotting using python or R
  + Pay attention to warning and log files
    - Can give useful information about why a run may have failed
    - Can use tools like MultiQC
  + Look for common biases in sequencing data
    - Can come from experimental design, methodology, sequencing chemistry, workflows
  + Check for contamination
    - Can happen during sample collection, extraction, library preparation, sequencing spikeins like PhiX
  + Consider the costs and benefits of stringent quality control for your data
    - Look for publications using similar experimental to help establish a proper balance
    - Talk to experts

### SECURING AND MANAGING APPROPRIATE COMPUTATIONAL RESOURCES

* Less complex datasets ay only need a laptop
* Others may need research-focused high-performance computing systems or research-integrated commercial analysis platforms
* Clusters!

### GETTING STARTED WITH RESOURCE MANAGEMENT

* APPLY FOR RESEARCH UNITS IF ELIGIBLE
  + Some cloud computing services offer grants to data-intensive researchers
* DEVELOP ON A LOCAL COMPUTER WHEN POSSIBLE
  + Test with subsampled data on local computer
* GAIN QUICK INSIGHTS USING SKETCHING ALGORITHMS
  + Quicker but less exact
  + Can be useful for generating hypotheses to guide the next analysis steps
* USE THE RIGHT TOOL FOR YOUR QUESTION
* SEEK HELP WHEN YOU NEED IT
  + Especially when available computing systems cannot handle what you are trying to do

### STRATEGIES FOR TROUBLESHOOTING

* Workshops and materials from: Carpentries, R-Ladies, RStudio
* HOW TO HELP YOURSELF: TRY TO PINPOINT YOUR ISSUE OR ERROR
  + Syntax errors, dependency issues, OS conflicts, bugs in the software, problems with input data, etc
  + Try running software on provided test data
    - If successful: it’s likely not a syntax error, the source code is likely functioning, unlikely an OS or dependency error
    - Check out the error message, might be useful
      * Likely a common issue, try google
    - If no error message: software help command
  + Semantic versioning
    - Arguments and functionality maintained for minor releases (1.1 to 1.2)
    - Functions change only with major releases (1.X to 2.0)
* HOW TO SEEK HELP: INCLUDE THE RIGHT DETAILS WITH YOUR QUESTION
  + Include (at minimum)
    - Name and version of the program
    - Method used to install it
    - Whether the test data ran
    - Exact code that produced the error
    - The error message
    - Full output text from the run (if applicable)
  + Also include
    - Type and version of the OS used to run it
    - Providing data that can reproduce the error (if applicable)
* WHERE TO SEEK HELP: ONLINE AND LOCAL COMMUNITIES OF PRACTICE
  + Errors for specific programs: the developers preferred location for answering questions and solving errors
  + Open source programs: “issues” tan within the software repository
  + General questions; forums like Stack Overflow, BioStars, SEQanswers
  + Also try local groups and meetups

Further applicable reading:

* FAIR practices[3]: Gruning B, Chilton J, K ¨ oster J, et al. Practical computational reproducibility in the life sciences. Cell Syst 2018;6(6):631–5.
* Snakemake documentation [30]: https://snakemake.readthedocs.io/
* Snakemake example workflow [31]: https://github.com/snakemake-workflows/chipseq
* Snakemake tutorial [32]: https://snakemake.readthedocs.io/en/stable/tutorial/tutorial.html
* Nextflow documentation [33]:
* Nextflow example workflow [34]:
* Nextflow tutorial [35]:
* CWL documentation [36]:
* CWL example workflow [37]:
* CWL tutorial [38]:
* WDL documentation [39]:
* WDL example workflow [40]:
* WDL tutorial [41]:

## SNAKEMAKE (MOLDER ET AL., 2022)

### ABSTRACT

* Reproducibility is very important!
* But reproducibility is not the only thing important for sustainability in the field
* Also important: adaptability, transparency

### INTRODUCTION

* Automating analyses can help achieve in silico reproducibility
* Existing workflow management systems partitioned into five niches
  + [1] GUIs: Galaxy, KNIME, Watchdog
    - Pros: shallow learning curve
  + [2] Specified using a set of classes and functions for generic languages: Anduril, Balsam, Hyperloom, Jug, Pwrake, Ruffus, SciPipe, SCOOP, COMPs
    - Pros: can be used without a graphical interface (eg: in a server), workflows can be managed by version control sysmtes like git
  + [3] Specified using a domain specific language (DSL): Nextflow, Snakemake, BioQueue, Bpipe, ClusterFlow, Cylc, BigDataScript
    - Pros: same as 2, but with improved readability
    - For snakemake specifically: it’s an extension of python, so a pro is maintained access to the full power of the underlying language
  + Workflow specification in a purely declarative way via config files: Popper
    - Pros: similar to 3, but are also particularly readable to non-developers
    - Cons: disallow imperative of functional programming (more restrictive)
  + System independent languages (CWL, WDL)
    - Pros: similar to number 4, can be executed on various specialized execution backends, some other workflows can be automatically exported into these
    - Cons: imperative or functional programming is not or less integrated into the specification language (limited expressive power)
* Support full in silico reproducibility of data analysis: Galaxy, Nextflow, Snakemake, WDL, CWL
* To gain full in silico reproducibility, data analysis must be automated, scalable to various computing platforms, and portable
* Equally important is transparency (parameters, software, custom code of each analysis step is fully accessible)
  + Must be readable and well-documented
  + Must be possible to trace parameters, code, components
* Adaptability will make it most useful to the scientific community (which means it must also be scalable and portable and readable)

### METHODS AND RESULTS

* AUTOMATION
  + Workflows specified through decomposition into steps represented as rules
    - Each rule described how to obtain a set of output files from a set of input files
    - Script integration instead of shell: Snakemake automatically inserts an object giving access to all properties of the job
    - Dependencies between jobs are implicit and inferred like this:
      * For each input file, snakemake determines a rule that can generate it yielding another job
      * Continues recursively
      * From this, snakemake obtains directed acyclic graph of jobs
  + Automated unit test generation
    - Important to test each contained step (unit test)
    - Each unit test consists of the execution of one rule, using input data taken from the source workflow, compared byte-by-byte against results given in source workflow
      * Can be overwritten by the user
* READABILITY
  + Dale-Chall readability formula: scores from the fraction of potentially unknown words among all words in a text
  + Seven categories that a statement can fall into (subsection 3.3)
  + In addition, each statement may:
    - Need domain knowledge (from the field analyzed in given workflow)
    - Need technical knowledge (eg about unix-style shell commands, python, etc)
    - Needs snakemake knowledge
    - Is trivial (understandable for everyone)
  + Modularization
    - Various levels of modularization that help design a workflow ensuring the reader is not distracted from aspects relevant to their interest
    - Snakefile inclusion
      * Snakefile: snakemake workflow definition
      * Can include others with an “include” statement defining path or url
    - Workflow composition
      * Snakemake can call workflow modules, compose multiple external workflows together
    - Step-wise modularization
      * For workflow steps that can be common to the scientific field and use widely used tools and libraries
        + Snakemake allows to deposit and use tool wrappers in central repository
    - Script integration
      * Works via a special “script” directive
      * Referred script can use all properties of the job (input files, output files, wildcard values, parameters, etc) automatically inserted as global snakemake object
    - Jupyter notebook integration
      * Analogous to script integration
      * “notebook” directive allows rule to specify path to jupyter notebook
    - Tool wrappers
      * Each wrapper contains:
        + Python or R script that:

Uses libraries of the respective scripting language, or

Calls a shell command

* + - * Also meta-wrappers, like a package of wrappers that work together
  + Standardized code linting and formatting
    - Readability can be influences by adhering to common style and best practices
    - Snakemake has automatic code formatting (snakefmt tool)
    - Built in code linter: detects code violting best practices and suggest solutions
    - There are also preconfigured github actions available for snakefmt and code linter
* PORTABILITY
  + Deploying a workflow to an unprepared system depends on
    - Ability to install workflow management system itself
    - Ability to obtain and use required software stack for each step
  + Conda integration
    - Can specify a software environment to be automatically deployed via conda package manager
  + Container integration
    - Snakemake pulls requested container image and run a job inside it using Singularity
  + Automatic containerization
    - Containers: requires additional energy and space
    - Recommended to use conda during workflow development as software needs may evolve
    - Once the workflow is ready, snakemake can automatically generate a dockerfile that defines the conda environment into a container
* TRACEABILITY AND DOCUMENTATION
  + Snakemake tracks input/output files, parameters, software, code for each job
    - This is then made into an interactive HTML based report
    - Visualizing dependencies, click on nodes to see details
    - Includes runtime statistics
* SCALABILITY
  + Snakemake allows workflow execution to scale to various computational platforms (single workstations, large compute servers)
  + Job scheduling
    - Jobs are either done, scheduled but not done, or not yet scheduled (open jobs)
    - Pending jobs: open job where prerequisites met, ready to run
    - Snakemake schedules based on priority, so that execution is as fast as possible, temporary outputs deleted quickly
    - A bunch of math I don’t care about
  + Caching between workflows
    - Reference management systems (for commonly accessed data like reference genomes)
    - Snakemake overcomes this with a hash value that allows you to store and look up outputs
    - More math that I don’t care about
  + Graph partitioning
    - Instead of forming one queue for all jobs, they can be partitioned into subgraphs that will be submitted together
    - Happens by assigning rules to a group
  + Streaming
    - Can tell the program not to save large data files on a disk, but instead directly streaming the output from a job into the consuming job

### FURTHER CONSIDERATIONS

* WORKFLOW COMPOSITION
  + Beneficial to keep the main workflow to a certain data type or not extend beyond a common scope
    - This keeps the workflow useful for people with different datasets and goals (not too specific)
    - Can add on additional stuff by declaring external workflows as modules
* ADVANCED WORKFLOW DESIGN PATTERNS
  + An example (I’m skipping this)
* READABILITY
  + Statements in snakemake workflow definitions fall into seven categories
    - 1.) a natural language word, followed by a colon (eg input: and output:)
    - 2.) the word “rule” followed by a name and a colon (eg rule convert\_to\_pdf:)
    - 3.) a quoted filename pattern (eg “{prefix}.pdf”)
    - A quoted shell command
    - A quoted wrapper identifier
    - A quoted container URL
    - A python statement
  + Justifications for each (skipping)
* SCHEDULING
  + Solved using a mixed integer linear program (MILP)
    - Fast removal of temporary files
    - You can also identify bottleneck jobs and prioritize them automatically
* PERFORMANCE
  + Processes dependencies between jobs beforehand, incurs some startup time
  + They are working on it

### CONCLUSION

* Interplay of automation, scalability, portability, readability, traceability, and documentation