Snakemake

Snakemake is a powerful workflow management system designed to create reproducible and scalable data analyses. Developed primarily for bioinformatics applications, it has gained widespread use across various fields due to its flexibility and ease of use. Here are some key features and characteristics of Snakemake:

Key Features
Rule-Based Workflow Definition:

Snakemake workflows are defined using rules, which specify how to transform input files into output files.
Each rule includes directives for the required input, the expected output, and the shell commands or scripts needed to perform the transformation.

Python Integration:

Workflows are written in a Python-based domain-specific language (DSL), allowing users to leverage Python's full capabilities.
Users can embed Python code directly within the workflow to handle complex logic and data manipulation.

Automatic Dependency Resolution:

Snakemake automatically determines the dependencies between tasks based on the input and output files specified in the rules.
It constructs a directed acyclic graph (DAG) to represent the workflow, ensuring tasks are executed in the correct order.

Scalability:

Snakemake can handle workflows ranging from small-scale analyses on a single machine to large-scale computations distributed across clusters or cloud platforms.
It supports parallel execution and can distribute tasks across multiple cores or nodes.

Reproducibility:

By explicitly defining the steps and dependencies in a workflow, Snakemake ensures that analyses can be reproduced precisely.
It includes features for version control of workflows and tracking of software environments using conda or container technologies like Docker and Singularity.

Flexible Execution:

Users can execute workflows locally, on high-performance computing (HPC) clusters, or in cloud environments.
Snakemake supports various job schedulers such as SLURM, PBS, and Grid Engine.

Error Handling and Checkpointing:

Snakemake provides mechanisms to handle errors gracefully and allows workflows to resume from the last successful step.
Checkpoints can be used for dynamic workflows where the completion of a step determines the execution of subsequent steps.

Resource Management:

Users can specify resource requirements for each rule, such as CPU cores, memory, and disk space.
Snakemake optimizes resource allocation and can manage resource constraints to prevent overloading the system.

Integrated Reporting and Visualization:

Snakemake can generate detailed reports and visualizations of the workflow, including the DAG of jobs and execution statistics.
These reports aid in debugging and documenting the analysis.

Community and Extensibility:

Snakemake has an active community and a wealth of resources, including tutorials, example workflows, and a collection of reusable workflow modules.
Users can extend Snakemake's functionality through custom scripts and third-party tools.

Now, we will create a snakemake pipeline as an example (Check the file in the google drive).

1) Before the use of the pipeline, copy or move the needed files (reads, genome created with the assembly) in a new folder called pipeline.
2) Create a new environment where you install bwa and samtools (tool for mapping, which is not a splicing-aware mapper). Call the environment: **alignment**.
3) Create a config file.
4) Run the following to create the index of your genome:
   bwa index contigs.fasta

Now, you can ran the pipeline:
co
snakemake -s snakefile.txt --use-conda --cores 2 # real run

Exercise for the final report:
Develop a Snakemake pipeline that maps the downloaded reads against the assembly previously generated with SPAdes. Extract the unmapped reads and assemble them. Finally, perform a manual BLAST search to identify the sequences.