Timeline of Workflow languages

- Bedford Lab Meeting-

Jennifer Chang, Ph.D.
Bioinformatic Analyst III
Fred Hutchinson Cancer Center

Some terms

- **Programming language**: bash, R, Python, Perl, C++, Java, JavaScript, Rust
- **Workflow language**: Snakemake, Nextflow, CWL, WDL
- **Runtime:**
  - Dependencies - Docker, Singularity, Conda, ambient installs
  - Specialized hardware - HPC (Slurm or SGE job schedulers)
  - Cloud - AWS, Google Cloud Compute, Microsoft Azure
- Possible comparisons:
  - syntax
  - features
    - data types (inputs/outputs)
    - control structures
    - parallelism
    - modularization

Why use a workflow language?

Step by step

For computational biologists, pipelines are methods; much like wet-lab protocols, they must be documented. But pipelines often comprise dozens of steps, so it's not trivial to do. Bioinformatician Titus Brown at the University of California, Davis, calculated that passing six samples through his *de novo* transcriptome assembly pipeline – involving data download, quality control, normalization, assembly, annotation and analysis – requires "well over 100 steps". Researchers must document precisely how each step is performed if they have any hope of reproducing them at a later date.

Typically, researchers codify workflows using general scripting languages such as Python or Bash. But these often lack the necessary flexibility. Workflows can involve hundreds to thousands of data files; a pipeline must be able to monitor their progress and exit gracefully if any step fails. And pipelines must be smart enough to work out which tasks need to be re-executed and which do not.

Bioinformatician Davis McCarthy at St Vincent's Institute of Medical Research in Fitzroy, Australia, says Python and R were more than enough for the relatively simple workflows he used as a PhD student. But today, McCarthy, who works with single-cell data sets, processes orders of magnitude more samples, some of which inevitably fail owing to problems such as network issues and memory shortages. "It was just way beyond my capabilities to figure that out from scratch for an analysis of this size," he says. He adopted the command-line-driven Snakemake, instead (see 'Anatomy of a workflow').

Perkel 2019 "Workflow systems turn raw data into scientific knowledge"

The early days

Before my involvement with Nextflow, I was a research engineer at the Cedric Notredame Lab for Comparative Bioinformatics. My job at the time was to help researchers run their workloads more efficiently on in-house computing clusters. While tools existed for managing bioinformatics workflows, most of our pipelines were developed in house using Bash and other scripting languages. There were challenges with this approach:

- Scripts were complex and usually understood only by their authors, making enhancing and maintaining workflows challenging.

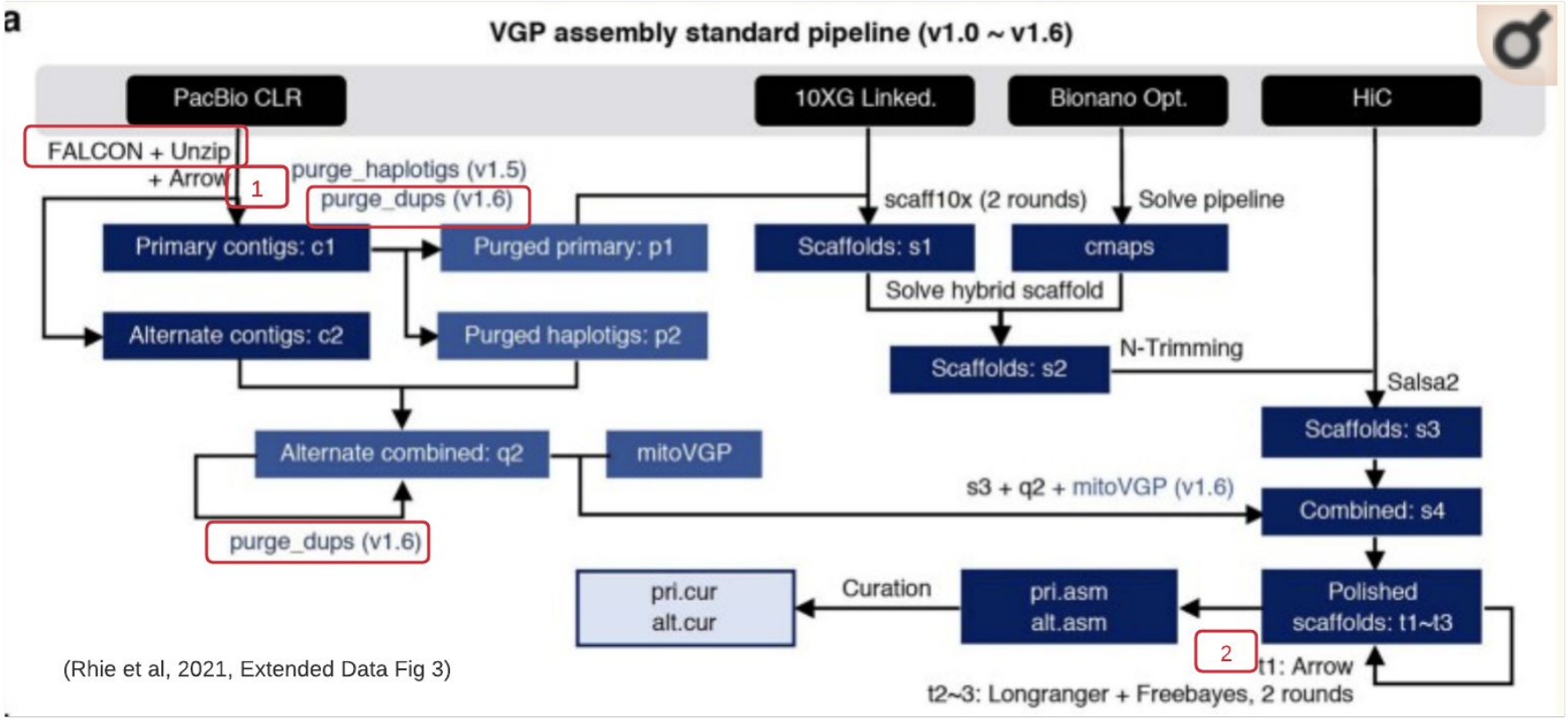
- Workflows were buggy and error-prone: imagine kicking off a long-running pipeline, launching thousands of jobs, only to have it fail after 10 hours of execution and needing to restart it from scratch.

- When workflows ran, it was hard to track progress. Without monitoring tools, we found ourselves constantly using the Linux command line, 'tailing' files and 'grepping' jobs to get a sense of where we were.

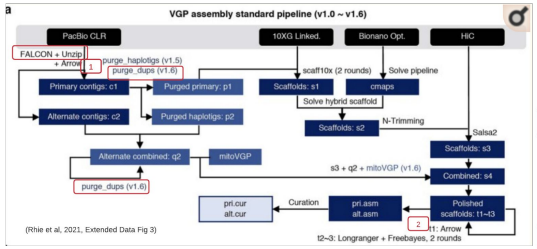
- Finally, the workflows were tightly tied to the compute environments. Even small changes to the environment could cause pipelines to break.

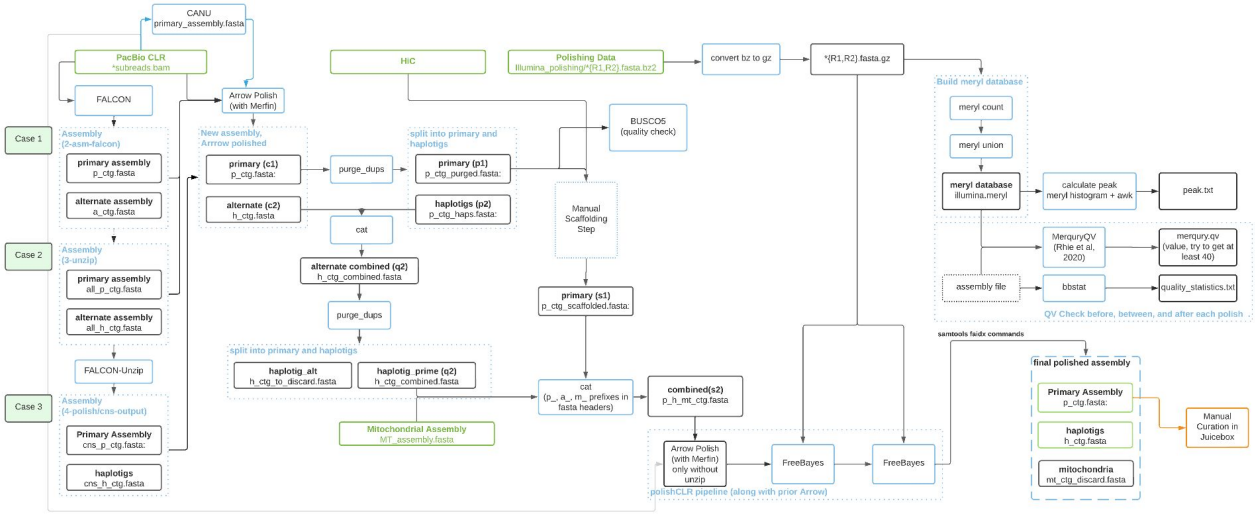
In other words, early pipeline processing was an utter mess. Installing a pipeline could take weeks of effort, requiring the configuration of obscure pieces of software, the use of bizarre programming languages and compilers, and troubleshooting missing libraries and components. One needed to know arcane environment variables and command line options passed among PhD students as a matter of ritual.

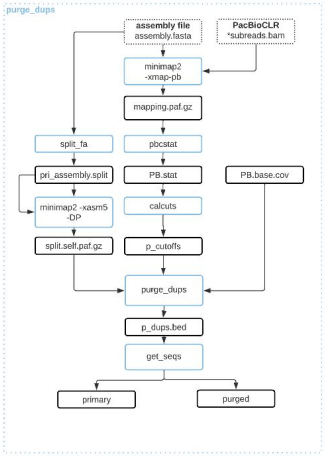
Di Tommaso, 2021 "The story of Nextflow: Building a modern pipeline orchestrator"

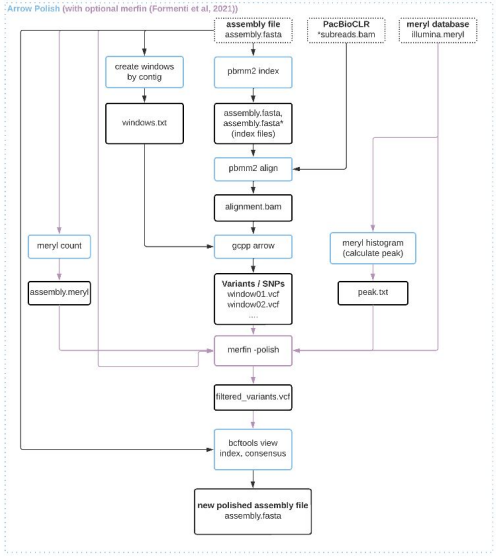


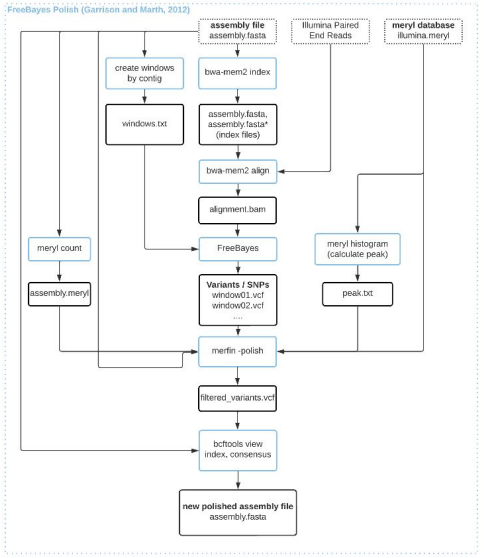
(Rhie et al, 2021, Extended Data Fig 3)











https://github.com/isugifNF/polishCLR/wiki

Why use a workflow language?

Step by step

For computational biologists, pipelines are methods; much like wet-lab protocols, they must be documented. But pipelines often comprise dozens of steps, so it's not trivial to do. Bioinformatician Titus Brown at the University of California, Davis, calculated that passing six samples through his *de novo* transcriptome assembly pipeline – involving data download, quality control, normalization, assembly, annotation and analysis – requires "well over 100 steps". Researchers must [illegible] is [illegible] hope of reproducing them at a later date.

Typically, researchers [illegible] Bash. But these often lack the necessary flexibility. Workflows can involve hundreds to thousands of data files, a pipeline must be able to monitor their progress and exit gracefully if any step fails. And pipelines must be smart enough to work out which tasks need to be re-executed and which do not.

Bioinformatician Davis McCarthy at St Vincent's Institute of Medical Research in Fitzroy, Australia, says Python and R were more than enough for the relatively simple workflows he used as a PhD student. But today, McCarthy, who works with single-cell data sets, processes orders of magnitude more samples, some of which inevitably fail owing to problems such as network issues and memory shortages. "It was just way beyond my capabilities to figure that out from scratch for an analysis of this size," he says. He adopted the command-line-driven Snakemake, instead (see 'Anatomy of a workflow').

Perkel 2019 "Workflow systems turn raw data into scientific knowledge"

The early days

Before my involvement with Nextflow, I was a research engineer at the Cedric Notredame Lab for Comparative Bioinformatics. My job at the time was to help researchers run their workloads more efficiently on in-house computing clusters. While tools existed for managing bioinformatics workflows, most of our pipelines were developed in house using Bash and other scripting languages. There were challenges with this approach:

[illegible] making enhancing and maintaining workflows challenging.

[illegible] running pipeline, launching thousands of jobs, only to have it fail after 10 hours of execution and needing to restart it from scratch.

- When workflows ran, it was hard to track progress. Without monitoring tools, we found ourselves constantly using the Linux command line, 'tailing' files and 'grepping' jobs to get a sense of where we were.

- Finally, the workflows were tightly tied to the compute environments. Even small changes to the environment could cause pipelines to break.

In other words, early pipeline processing was an utter mess. Installing a pipeline could take weeks of effort, requiring the configuration of obscure pieces of software, the use of bizarre programming languages and compilers, and troubleshooting missing libraries and components. One needed to know arcane environment variables and command line options passed among PhD students as a matter of ritual.

Di Tommaso, 2021 "The story of Nextflow: Building a modern pipeline orchestrator"

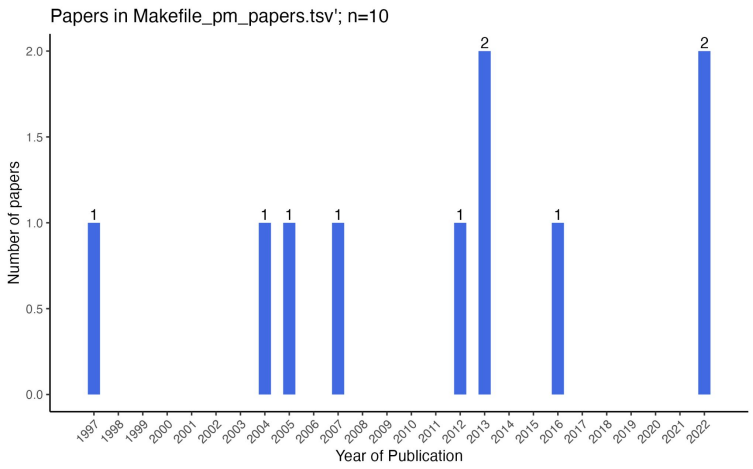
The common aim is to make computational methods reproducible, portable, maintainable, and shareable

Agenda

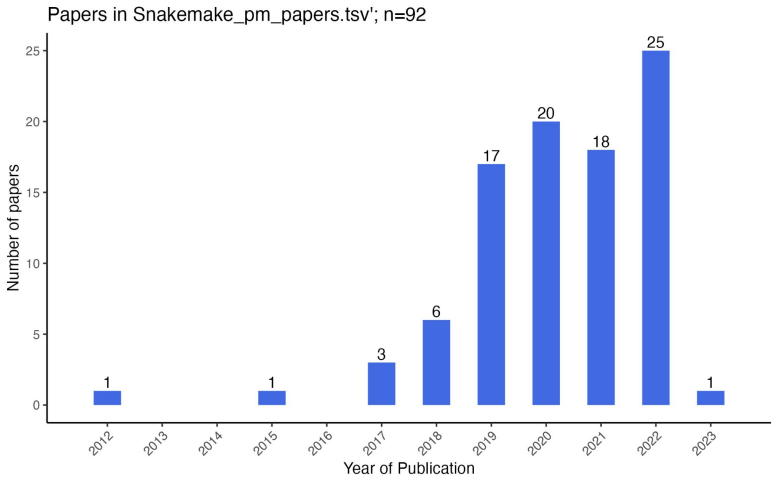
- Makefiles
- Snakemake
- Nextflow
- CWL and WDL

Agenda

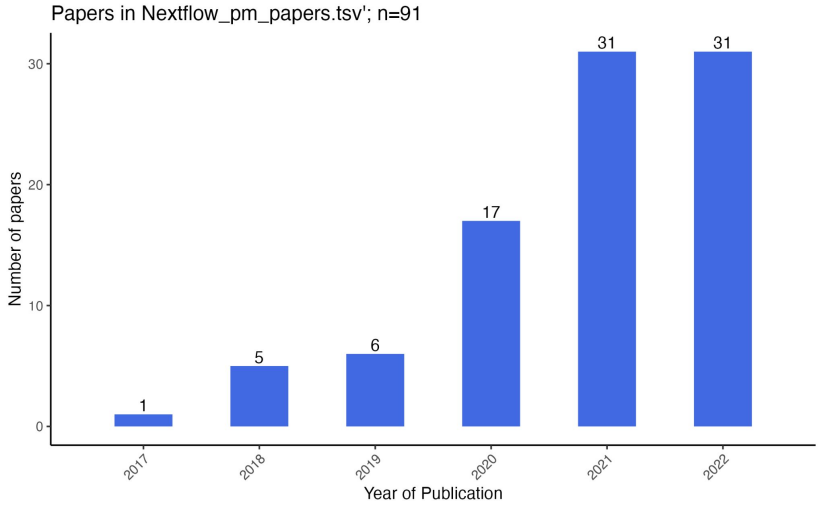
Makefiles - pubmed check (3)



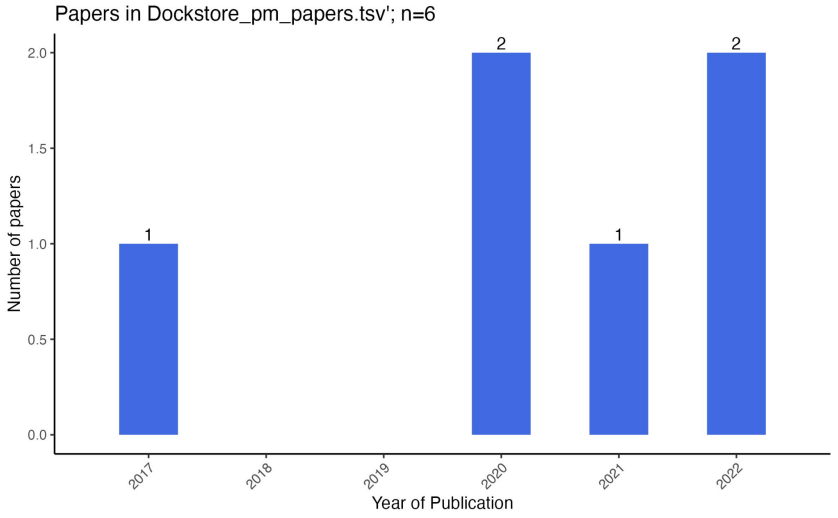
Snakemake - pubmed_check (93)



Nextflow - pubmed_check (92)

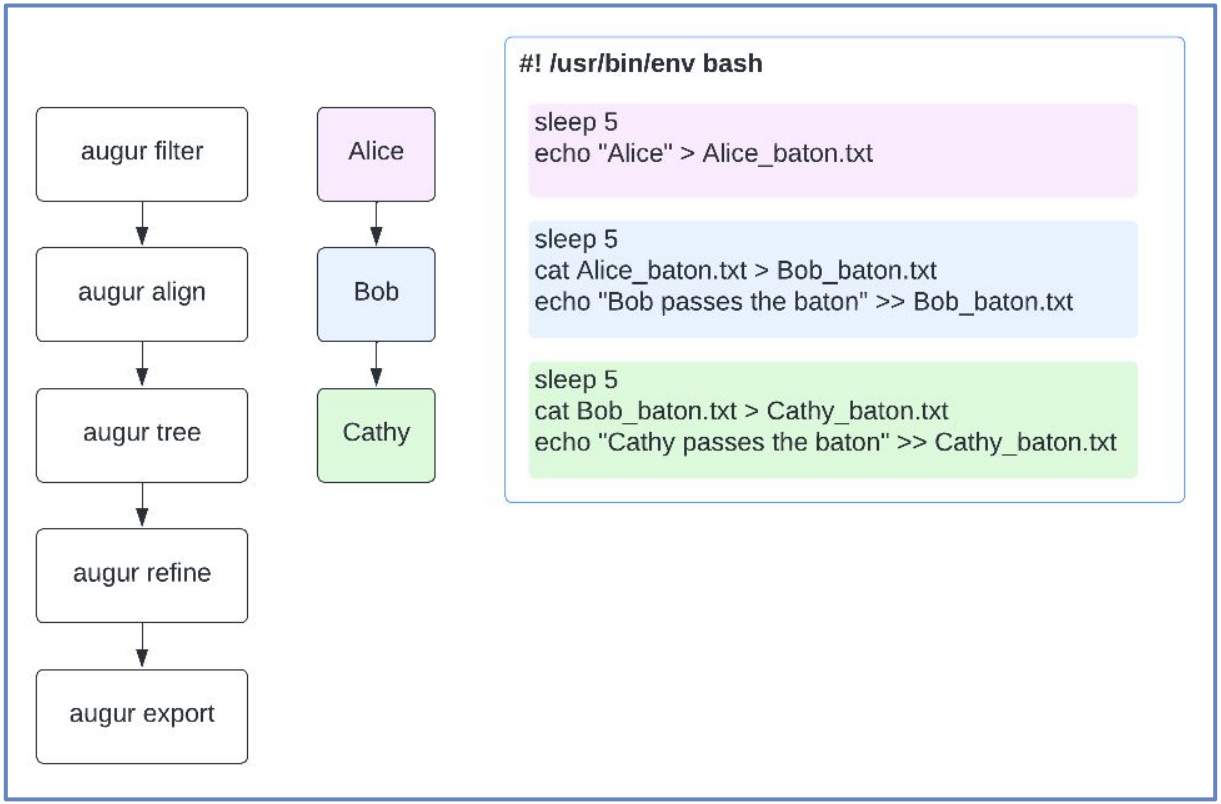


CWL and WDL - "Dockstore" pubmed_check (7)



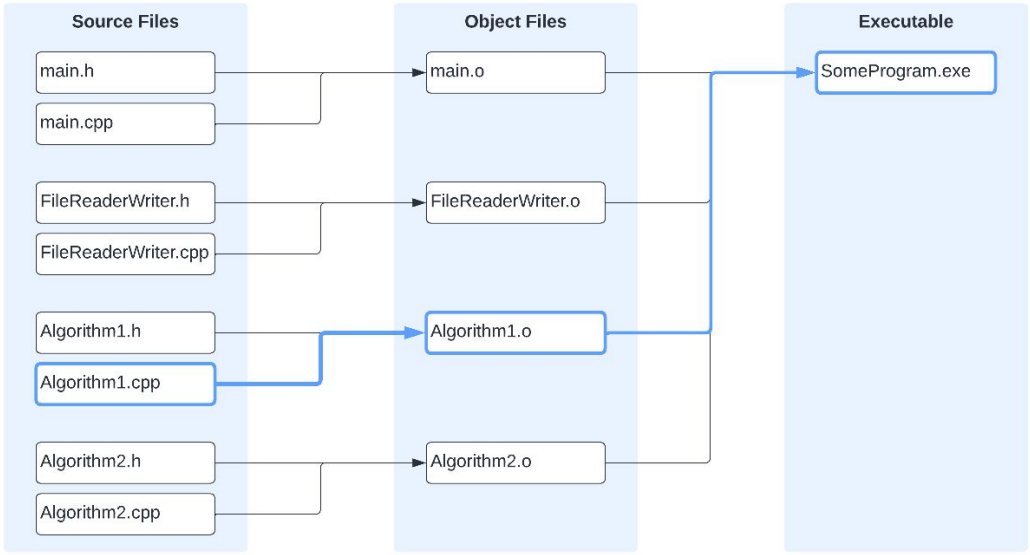
Agenda

- **Makefiles**
- Snakemake
- Nextflow
- CWL and WDL



1976 - Makefiles

> Make originated with a visit from Steve Johnson (author of yacc, etc.), storming into my office, cursing the Fates that had caused him to waste a morning debugging a correct program (bug had been fixed, file hadn't been compiled, `cc *.o` was therefore unaffected). As I had spent a part of the previous evening coping with the same disaster on a project I was working on, the idea of a tool to solve it came up. It began with an elaborate idea of a dependency analyzer, boiled down to something much simpler, and turned into Make that weekend. Use of tools that were still wet was part of the culture. Makefiles were text files, not magically encoded binaries, because that was the Unix ethos: printable, debuggable, understandable stuff.
>
> — Stuart Feldman, *The Art of Unix Programming*, Eric S. Raymond 2003



- Designed by Stuart Feldman and first released April 1976
- Makefiles defined **rules**
- **target**: name of the file generated, usually object files or executables
- **dependency**: input file that is used to create a target
- **command**: the action that takes dependencies to create the target

1976 - Makefiles

#! /usr/bin/env bash

```
sleep 5
echo "Alice" > Alice_baton.txt
```

```
sleep 5
cat Alice_baton.txt > Bob_baton.txt
echo "Bob passes the baton" >> Bob_baton.txt
```

```
sleep 5
cat Bob_baton.txt > Cathy_baton.txt
echo "Cathy passes the baton" >> Cathy_baton.txt
```

Writing a set of Makefile Rules

target : dependency

command

Makefile

```
all :  Cathy_baton.txt
	cat Cathy_baton.txt
```

```
Alice_baton.txt :
	sleep 5
	echo "Alice" > Alice_baton.txt
```

```
Bob_baton.txt : Alice_baton.txt
	sleep 5
	cat Alice_baton.txt > Bob_baton.txt
	echo "Bob passes the baton" >> Bob_baton.txt
```

```
Cathy_baton.txt  : Bob_baton.txt
	sleep 5
	cat Bob_baton.txt > Cathy_baton.txt
	echo "Cathy passes the baton" >> Cathy_baton.txt
```

1976 - Makefiles

```
#! /usr/bin/env bash

sleep 5
echo "Alice" > Alice_baton.txt

sleep 5
cat Alice_baton.txt > Bob_baton.txt
echo "Bob passes the baton" >> Bob_baton.txt

sleep 5
cat Bob_baton.txt > Cathy_baton.txt
echo "Cathy passes the baton" >> Cathy_baton.txt
```

Writing a set of Makefile Rules

```
target : dependency
    command
```

Makefile

```
all :  Cathy_baton.txt
    cat Cathy_baton.txt

Alice_baton.txt :
    sleep 5
    echo "Alice" > Alice_baton.txt

Bob_baton.txt : Alice_baton.txt
    sleep 5
    cat Alice_baton.txt > Bob_baton.txt
    echo "Bob passes the baton" >> Bob_baton.txt

Cathy_baton.txt  : Bob_baton.txt
    sleep 5
    cat Bob_baton.txt > Cathy_baton.txt
    echo "Cathy passes the baton" >> Cathy_baton.txt
```

- Caching is file-based —tracks the existence of files (dependencies and targets)
- Use rules to pull a final "target"
- To be run locally
- PubMed: https://pubmed.ncbi.nlm.nih.gov/?term=makefile&sort=date
- Example: mgalardini/reads2snps/Makefile

pubmed check

Agenda

- Makefiles - 1976, rule, target, dependency, command
- **Snakemake**
- Nextflow
- CWL and WDL

2012 Snakemake

Makefile

```
all :  Cathy_baton.txt
        cat Cathy_baton.txt

Alice_baton.txt :
        sleep 5
        echo "Alice" > Alice_baton.txt

Bob_baton.txt : Alice_baton.txt
        sleep 5
        cat Alice_baton.txt > Bob_baton.txt
        echo "Bob passes the baton" >> Bob_baton.txt

Cathy_baton.txt  : Bob_baton.txt
        sleep 5
        cat Bob_baton.txt > Cathy_baton.txt
        echo "Cathy passes the baton" >> Cathy_baton.txt
```

Writing a set of Snakemake Rules

```
rule NAME:
    input: dependency files
    output: target files
    shell:
        """
        command
        """
    script:
    log:
    params:
    thread:
    ....
```

Snakemake

```
rule all :
  input: "Cathy_baton.txt "

rule Alice:
  output: "Alice_baton.txt"
  shell:
      """
      sleep 5
      echo 'Alice passes the baton' >> {output}
      """

rule Bob:
  input: "Alice_baton.txt"
  output: "Bob_baton.txt"
  shell:
      """
      sleep 5
      cat {input} > {output}
      echo 'Bob passes the baton' >> {output}
      """

rule Cathy:
  input: "Bob_baton.txt"
  output: "Cathy_baton.txt"
  shell:
      """
      sleep 5
      cat {input} > {output}
      echo 'Cathy passes the baton' >> {output}
      """
```

2012 - Snakemake

- Designed by Johannes Koster and team and first published in 2012
- Snakemake was designed to be a readable python-based workflow definition language and powerful execution environment
- first system to support the use of automatically inferred multiple named widcards (or variables)
- newer features: module, paramspace, piped output for streamed output, job grouping
- https://github.com/nextstrain/zika/blob/refactor/modularize/Snakefile

Listing 1. Example Snakefile for mapping paired-end reads with BWA.

```
(1)  SAMPLES = "100 101 102 103".split()
(2)  REF = "hg19.fa"
(3)  rule all:
(4)   input: "{sample}.coverage.pdf".format(sample = sample)
(5)     for sample in SAMPLES
(6)  rule fastq_to_sai:
(7)    input: ref = REF, reads = "{sample}.{group}.fastq"
(8)    output: temp("{sample}.{group}.sai")
(9)    shell: "bwa aln {input.ref} {input.reads} > {output}"
(10) rule sai_to_bam:
(11)   input: REF, "{sample}.1.sai", "{sample}.2.sai",
(12)     "{sample}.1.fastq", "{sample}.2.fastq"
(13)   output: protected("{sample}.bam")
(14)   shell: "bwa sampe {input} | samtools view -Sbh - > {output}"
(15) rule remove_duplicates:
(16)   input: "{sample}.bam"
(17)   output: "{sample}.nodup.bam"
(18)   shell: "samtools rmdup {input} {output}"
(19) rule plot_coverage_histogram:
(20)   input: "{sample}.nodup.bam"
(21)   output: hist = "{sample}.coverage.pdf"
(22)   run:
(23)     from matplotlib.pyplot import hist, savefig
(24)     hist(list(map(int,
(25)       shell("samtools mpileup {input} | cut -f4",
(26)       iterable=True))))
(27)     savefig(output.hist)
```

Koster & Rahmann, 2012; Molder et al, 2021

2012 Snakemake

Makefile

```
all :  Cathy_baton.txt
        cat Cathy_baton.txt

Alice_baton.txt :
        sleep 5
        echo "Alice" > Alice_baton.txt

Bob_baton.txt : Alice_baton.txt
        sleep 5
        cat Alice_baton.txt > Bob_baton.txt
        echo "Bob passes the baton" >> Bob_baton.txt

Cathy_baton.txt  : Bob_baton.txt
        sleep 5
        cat Bob_baton.txt > Cathy_baton.txt
        echo "Cathy passes the baton" >> Cathy_baton.txt
```

Writing a set of Snakemake Rules

```
rule NAME:
    input: dependency files
    output: target files
    shell:
        """
        command
        """
    script:
    log:
    params:
    thread:
    ....
```

Snakemake

```
rule all :
  input: "Cathy_baton.txt "

rule Alice:
  output: "Alice_baton.txt"
  shell:
      """
      sleep 5
      echo 'Alice passes the baton' >> {output}
      """

rule Bob:
  input: "Alice_baton.txt"
  output: "Bob_baton.txt"
  shell:
      """
      sleep 5
      cat {input} > {output}
      echo 'Bob passes the baton' >> {output}
      """

rule Cathy:
  input: "Bob_baton.txt"
  output: "Cathy_baton.txt"
  shell:
      """
      sleep 5
      cat {input} > {output}
      echo 'Cathy passes the baton' >> {output}
      """
```

- file-based caching, but can have temp/pipe intermediates
- pass in values via params
- pulls a final "target"
- Command using {input} {output} variables pubmed_check

https://github.com/nextstrain/ncov-ingest/blob/master/workflow/snakemake_rules

Agenda

- Makefiles - 1976, rule, target, dependency, command
- Snakemake - 2012 published, rule, wildcards, params, modules
- **Nextflow**
- CWL and WDL

2013 Nextflow

Snakemake

```
rule all :
  input: "Cathy_baton.txt "
```

```
rule Alice:
    output: "Alice_baton.txt"
    shell:
        """
        sleep 5
        echo 'Alice passes the baton' >> {output}
        """
```

```
rule Bob:
    input: "Alice_baton.txt"
    output: "Bob_baton.txt"
    shell:
        """
        sleep 5
        cat {input} > {output}
        echo 'Bob passes the baton' >> {output}
        """
```

```
rule Cathy:
    input: "Bob_baton.txt"
    output: "Cathy_baton.txt"
    shell:
        """
        sleep 5
        cat {input} > {output}
        echo 'Cathy passes the baton' >> {output}
        """
```

Writing a set of Nextflow Processes

```
process NAME {
      input: tuple, path, value
      output: tuple, path, value
      shell:
          """
          command
          """
}

workflow WK_NAME {
  Name()
  | view
}
```

Nextflow

```
workflow all {
   Alice
   | Bob
   | Cathy
   | view
}
```

```
process Alice:
    output: path("Alice_baton.txt" )
    shell:
        """
        sleep 5
        echo 'Alice passes the baton' >> "Alice_baton.txt"
        """
```

```
process Bob:
    input: path(inile)
    output: path("Bob_baton.txt" )
    shell:
        """
        sleep 5
        cat $infile > Bob_baton.txt
        echo 'Bob passes the baton' >> "Bob_baton.txt"
        """
```

```
process Cathy:
    input: path(infile)
    output: path("Cathy_baton.txt" )
    shell:
        """
        sleep 5
        cat ${infile} > "Cathy_baton.txt"
        echo 'Cathy passes the baton' >> "Cathy_baton.txt"
        """
```

nextflow.config

```
process {
  publishDir "results", mode "copy"
}

profile {
```

2013 - Nextflow

- Designed by **Paolo Di Tommaso** and first released in March 2013
- **Nextflow** designed with the goal that researchers can continue to use their favorite programming language and tools, and swap out the compute environment
- **DataFlow** is a programming model that allows the definition of tasks that execute in parallel in a declarative manner. Imagine tasks in Nextflow workflow as cells in a spreadsheet. When a cell is modified the change is propagated automatically to all dependent cells.

Di Tommaso, 2021 "The story of Nextflow: Building a modern pipeline orchestrator"

2013 Nextflow

Snakemake

```
rule all :
  input: "Cathy_baton.txt "

rule Alice:
   output: "Alice_baton.txt"
   shell:
        """
        sleep 5
        echo 'Alice passes the baton' >> {output}
        """

rule Bob:
   input: "Alice_baton.txt"
   output: "Bob_baton.txt"
   shell:
        """
        sleep 5
        cat {input} > {output}
        echo 'Bob passes the baton' >> {output}
        """

rule Cathy:
   input: "Bob_baton.txt"
   output: "Cathy_baton.txt"
   shell:
        """
        sleep 5
        cat {input} > {output}
        echo 'Cathy passes the baton' >> {output}
        """
```

Writing a set of Nextflow Processes

```
process NAME {
      input: tuple, path, value
      output: tuple, path, value
      shell:
          """
          command
          """
}

workflow WK_NAME {
  Name()
  | view
}
```

Nextflow

```
workflow all {
   Alice
   | Bob
   | Cathy
   | view
}

process Alice:
   output: path("Alice_baton.txt" )
   shell:
        """
        sleep 5
        echo 'Alice passes the baton' >> "Alice_baton.txt"
        """

process Bob:
   input: path(inile)
   output: path("Bob_baton.txt" )
   shell:
        """
        sleep 5
        cat $infile > Bob_baton.txt
        echo 'Bob passes the baton' >> "Bob_baton.txt"
        """

process Cathy:
   input: path(infile)
   output: path("Cathy_baton.txt" )
   shell:
        """
        sleep 5
        cat ${infile} > "Cathy_baton.txt"
        echo 'Cathy passes the baton' >> "Cathy_baton.txt"
        """
```

- Input accepts both files and paths
- Instead of wiring input and outputs within each rule, connect rule (process) in a workflow section
- isolated runs in a work directory

nextflow.config

```
process {
  publishDir "results", mode "copy"
}

profile {
```

2013 Nextflow - isolated folders

- isolated runs in a [work directory]

```
[2017_Nextflow % tree -a work
work
├── 1b
│   └── 415d18adab625f579c622db59e6184
│       ├── .command.begin
│       ├── .command.err
│       ├── .command.log
│       ├── .command.out
│       ├── .command.run
│       ├── .command.sh
│       ├── .exitcode
│       ├── Cathy_baton.txt -> /Users/jchang3/github/j23414/compare_workflows/2017_Nextflow/work/c1/dbc37a7011a9b2c3120b1f788e1375/Cathy_baton.txt
│       └── Dave_baton.txt
├── 8d
│   └── d951c35cc8ebede31c2f5ae4c7aec5
│       ├── .command.begin
│       ├── .command.err
│       ├── .command.log
│       ├── .command.out
│       ├── .command.run
│       ├── .command.sh
│       ├── .exitcode
│       ├── Dave_baton.txt -> /Users/jchang3/github/j23414/compare_workflows/2017_Nextflow/work/1b/415d18adab625f579c622db59e6184/Dave_baton.txt
│       └── Eve_baton.txt
├── a0
│   └── 6d627760375285d1fb919c2213567c
│       ├── .command.begin
│       ├── .command.err
│       ├── .command.log
│       ├── .command.out
│       ├── .command.run
│       ├── .command.sh
│       ├── .exitcode
│       └── Alice_baton.txt
├── c1
│   └── dbc37a7011a9b2c3120b1f788e1375
│       ├── .command.begin
│       ├── .command.err
│       ├── .command.log
│       ├── .command.out
│       ├── .command.run
│       ├── .command.sh
│       ├── .exitcode
│       ├── Bob_baton.txt -> /Users/jchang3/github/j23414/compare_workflows/2017_Nextflow/work/ed/a34e08e4ecf606e7730ba5518dd62c/Bob_baton.txt
│       └── Cathy_baton.txt
└── ed
    └── a34e08e4ecf606e7730ba5518dd62c
        ├── .command.begin
        ├── .command.err
        ├── .command.log
        ├── .command.out
        ├── .command.run
        ├── .command.sh
        ├── .exitcode
        ├── Alice_baton.txt -> /Users/jchang3/github/j23414/compare_workflows/2017_Nextflow/work/a0/6d627760375285d1fb919c2213567c/Alice_baton.txt
        └── Bob_baton.txt
```

```
[2017_Nextflow % nextflow run next_baton.nf
N E X T F L O W  ~  version 22.10.0
Launching `next_baton.nf` [happy_roentgen] DSL2 - revision: 4e1f6f8ca7

Pipeline = Alice -> Bob -> Cathy -> Dave -> Eve
 where each person runs 5 seconds to pass the baton to next person

executor >  local (5)
[a0/6d6277] process > Alice [100%] 1 of 1 ✔
[ed/a34e08] process > Bob   [100%] 1 of 1 ✔
[c1/dbc37a] process > Cathy [100%] 1 of 1 ✔
[1b/415d18] process > Dave  [100%] 1 of 1 ✔
[8d/d951c3] process > Eve   [100%] 1 of 1 ✔
/Users/jchang3/github/j23414/compare_workflows/2017_Nextflow/work/8d/d951c35cc8ebede31c2f5ae4c7aec5/Eve_baton.txt
```

Agenda

- Makefiles - 1976, rule, target, dependency, command
- Snakemake - 2012 published, rule, wildcards, params, modules
- Nextflow - 2013 (?) 2017 published, DataFlow, channel, isolated work dir
- **CWL** and WDL

2014 - CWL

- cat.cwl - https://github.com/ncbi/cwl-ngs-workflows-cbb/blob/master/tools/basic/cat.cwl
- wc.cwl - https://github.com/ncbi/cwl-ngs-workflows-cbb/blob/master/tools/basic/wc.cwl
- CWL originated from discussions between Peter Amstutz, John Chilton, Nebojsa Tijanic, and Michael R. Crusoe
  "The Common Workflow Languagen (CWL) is a language specification designed by the bioinformatics community to unify the style, principles and standards of coding pipelines, in a way that is agnostic of the hardware. It prioritizes reproducibility and portability of workflows and hence requires explicit/pedantic parameters definitions, making it very verbose. In contrast, Workflow Description Language (WDL) is a **language specification that emphasizes human readability of the code and an easy learning curve**, at the cost of being restrictive in its expressiveness (fig Supplementary 4)."
  https://www.nature.com/articles/s41598-021-99288-8
  OpenWDL was developed at the broad institute

WfMS	First commit	Contributors	Closed	Open	License
Swift-t	2011-05-11	16	109	81	apache-2.0
Nextflow	2013-03-22	81	1770	159	apache-2.0
CWL	2014-09-25	62	667	249	apache-2.0
WDL	2012-08-01	44	376	50	bsd-3-clause

Table 3. GitHub activities from each WfMS (March 4th, 2021). *Contributors* is the number of contributors

Agenda

- Makefiles - 1976, rule, target, dependency, command
- Snakemake - 2012 published, rule, wildcards, params, modules
- Nextflow - 2013 (?) 2017 published, DataFlow, channel, isolated work dir
- CWL (verbose) and **WDL**

Nextflow

```
workflow all {
   Alice
   | Bob
   | Cathy
   | view
}
```

```
process Alice:
    output: path("Alice_baton.txt" )
    shell:
        """
        sleep 5
        echo 'Alice passes the baton' >> "Alice_baton.txt"
        """
```

```
process Bob:
    input: path(inile)
    output: path("Bob_baton.txt" )
    shell:
        """
        sleep 5
        cat $infile > Bob_baton.txt
        echo 'Bob passes the baton' >> "Bob_baton.txt"
        """
```

```
process Cathy:
    input: path(infile)
    output: path("Cathy_baton.txt" )
    shell:
        """
        sleep 5
        cat ${infile} > "Cathy_baton.txt"
        echo 'Cathy passes the baton' >> "Cathy_baton.txt"
        """
```

nextflow.config

```
process {
  publishDir "results", mode "copy"
}

profile {
  standard {}
  docker {}
  singularity {}
  aws {}
  conda {}
   slurm {}
}
```

Writing a set of WDL Tasks

```
task NAME {
      input {
        File infile
        # String, Int, Double, Boolean, Array, Object
      }
      command <<<
           command


          >>>
      output {
         File outfile = "outfile.txt"
      }
      runtime {
        docker: "ubuntu"
      }
}

workflow WK_NAME {
   input {
     File textfile
   }
   call NAME {
     input: infile=textfile
   }
  output {
     File final_out = NAME.outfile
  }
}
```

WDL

```
workflow all {
   call Alice
   call Bob { input: infile=Alice.outfile }
   call Cathy { input: infile=Bob.outfile }
   output {
     File final_out = Cathy.outfile
}
```

```
task Alice {
   command <<<
        sleep 5
        echo 'Alice passes the baton' >> "Alice_baton.txt"

   >>>
   output {
     File outfile = "Alice_baton.txt"
   }
   runtime {
      docker: "ubuntu"
   }
}
```

```
task Bob {
   input {
     File infile
   }
   command <<<
    sleep 5
    cat {input} > {output}
    echo 'Bob passes the baton' >> "Bob_baton.txt"
   >>>
   output {
     File outfile = "Bob_baton.txt"
   }
   runtime {
      docker: "ubuntu"
   }
}
```

```
task Cathy {
   input {
     File infile
   }
   command <<<
    sleep 5
    cat ${input_file} > "Cathy_baton.txt"
    echo 'Cathy passes the baton' >> "Cathy_baton.txt"
   >>>
   output {
     File outfile = "Bob_baton.txt"
   }
   runtime {
      docker: "ubuntu"
   }
}
```

Compute isolation - File localization

script

```
#!/bin/bash

cd /cromwell-executions/All_Workflow/4de95676-46bf-4181-8452-ece5ff3bab29/call-Bob/execution
tmpDir=$(mkdir -p "/cromwell-executions/All_Workflow/4de95676-46bf-4181-8452-ece5ff3bab29/call-Bob/tmp.55e4ea9b" && echo "/cromwell-executions/All_Workflow/4de95676-46bf-4181-8452-ece5ff3bab29/call-Bob/tmp.55e4ea9b")
chmod 777 "$tmpDir"
export _JAVA_OPTIONS=-Djava.io.tmpdir="$tmpDir"
export TMPDIR="$tmpDir"
export HOME="$HOME"
(
cd /cromwell-executions/All_Workflow/4de95676-46bf-4181-8452-ece5ff3bab29/call-Bob/execution

)
out4de95676="${tmpDir}/out.$$" err4de95676="${tmpDir}/err.$$"
mkfifo "$out4de95676" "$err4de95676"
trap 'rm "$out4de95676" "$err4de95676"' EXIT
tee '/cromwell-executions/All_Workflow/4de95676-46bf-4181-8452-ece5ff3bab29/call-Bob/execution/stdout' < "$out4de95676" &
tee '/cromwell-executions/All_Workflow/4de95676-46bf-4181-8452-ece5ff3bab29/call-Bob/execution/stderr' < "$err4de95676" >&2 &
(
cd /cromwell-executions/All_Workflow/4de95676-46bf-4181-8452-ece5ff3bab29/call-Bob/execution


#! /usr/bin/env bash
sleep 5
cat /cromwell-executions/All_Workflow/4de95676-46bf-4181-8452-ece5ff3bab29/call-Bob/inputs/1050289724/Alice_baton.txt > Bob_baton.txt
echo 'Bob passes baton' >> Bob_baton.txt
) > "$out4de95676" 2> "$err4de95676"
echo $? > /cromwell-executions/All_Workflow/4de95676-46bf-4181-8452-ece5ff3bab29/call-Bob/execution/rc.tmp
(
# add a .file in every empty directory to facilitate directory delocalization on the cloud
cd /cromwell-executions/All_Workflow/4de95676-46bf-4181-8452-ece5ff3bab29/call-Bob/execution
find . -type d -exec sh -c '[ -z "$(ls -A '"'"'{}'"'"')" ] && touch '"'"'{}'"'"'/.file' \;
)
(
cd /cromwell-executions/All_Workflow/4de95676-46bf-4181-8452-ece5ff3bab29/call-Bob/execution
sync


)
mv /cromwell-executions/All_Workflow/4de95676-46bf-4181-8452-ece5ff3bab29/call-Bob/execution/rc.tmp /cromwell-executions/All_Workflow/4de95676-46bf-4181-8452-ece5ff3bab29/call-Bob/execution/rc

```

Nextflow

```
workflow all {
   Alice
   | Bob
   | Cathy
   | view
}
```

```
process Alice:
    output: path("Alice_baton.txt" )
    shell:
        """
        sleep 5
        echo 'Alice passes the baton' >> "Alice_baton.txt"
        """
```

```
process Bob:
    input: path(inile)
    output: path("Bob_baton.txt" )
    shell:
        """
        sleep 5
        cat $infile > Bob_baton.txt
        echo 'Bob passes the baton' >> "Bob_baton.txt"
        """
```

```
process Cathy:
    input: path(infile)
    output: path("Cathy_baton.txt" )
    shell:
        """
        sleep 5
        cat ${infile} > "Cathy_baton.txt"
        echo 'Cathy passes the baton' >> "Cathy_baton.txt"
        """
```

nextflow.config

```
process {
  publishDir "results", mode "copy"
}

profile {
  standard {}
  docker {}
  singularity {}
  aws {}
  conda {}
   slurm {}
}
```

Writing a set of WDL Tasks

```
task NAME {
      input {
        File infile
        # String, Int, Double, Boolean, Array, Object
      }
      command <<<
           command


          >>>
      output {
         File outfile = "outfile.txt"
      }
      runtime {
        docker: "ubuntu"
      }
}

workflow WK_NAME {
   input {
     File textfile
   }
   call NAME {
     input: infile=textfile
   }
  output {
     File final_out = NAME.outfile
  }
}
```

real-world example:
[TheiaCov_Augur_Run](#)
[Terra Dashboard](#)
[remote modules](#)

WDL

```
workflow all {
   call Alice
   call Bob { input: infile=Alice.outfile }
   call Cathy { input: infile=Bob.outfile }
   output {
      File final_out = Cathy.outfile
}
```

```
task Alice {
   command <<<
        sleep 5
        echo 'Alice passes the baton' >> "Alice_baton.txt"

   >>>
   output {
     File outfile = "Alice_baton.txt"
   }
   runtime {
      docker: "ubuntu"
   }
}
```

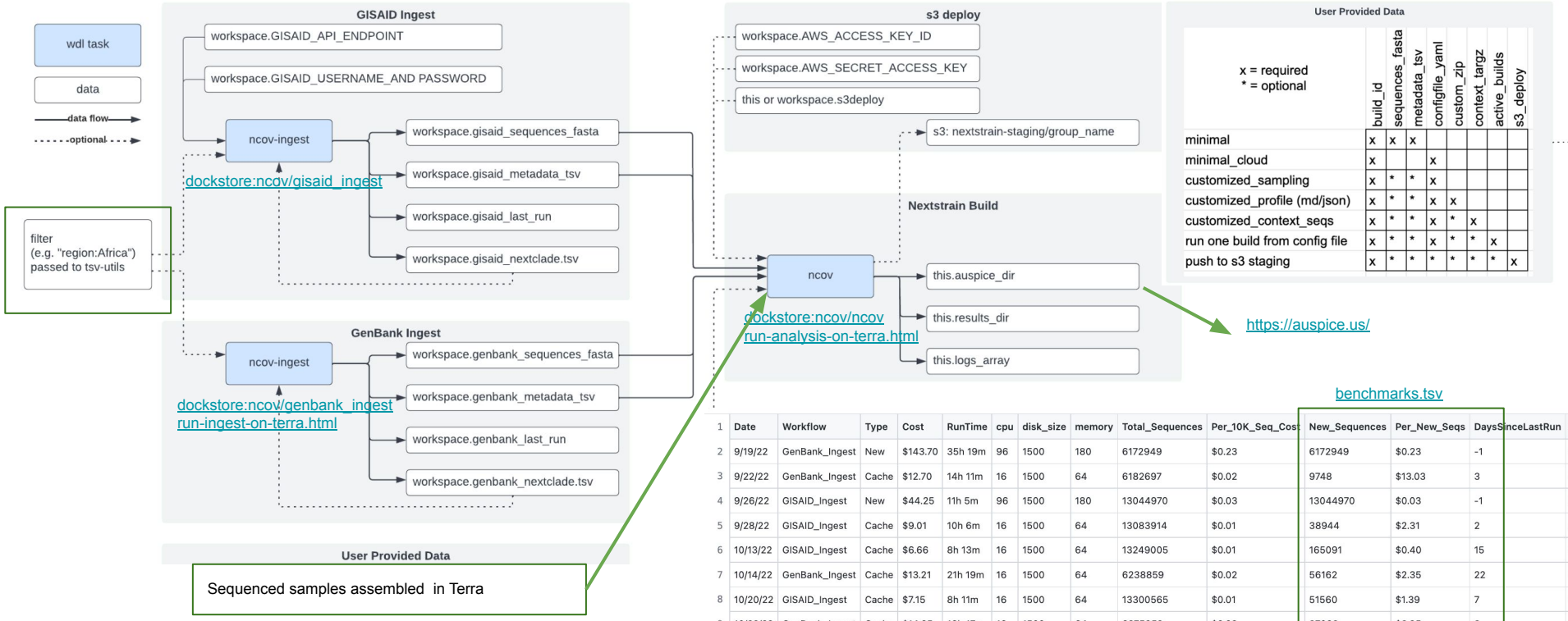
```
task Bob {
   input {
     File infile
   }
   command <<<
    sleep 5
    cat {input} > {output}
    echo 'Bob passes the baton' >> "Bob_baton.txt"
   >>>
   output {
     File outfile = "Bob_baton.txt"
   }
   runtime {
      docker: "ubuntu"
   }
}
```

```
task Cathy {
   input {
     File infile
   }
   command <<<
    sleep 5
    cat ${input_file} > "Cathy_baton.txt"
    echo 'Cathy passes the baton' >> "Cathy_baton.txt"
   >>>
   output {
     File outfile = "Bob_baton.txt"
   }
   runtime {
      docker: "ubuntu"
   }
}
```

2012 or 2014? - WDL

Cloud-based Nextstrain analyses (Aim A3) with Terra

- Work has continued to document ncov-ingest and ncov workflows on Terra



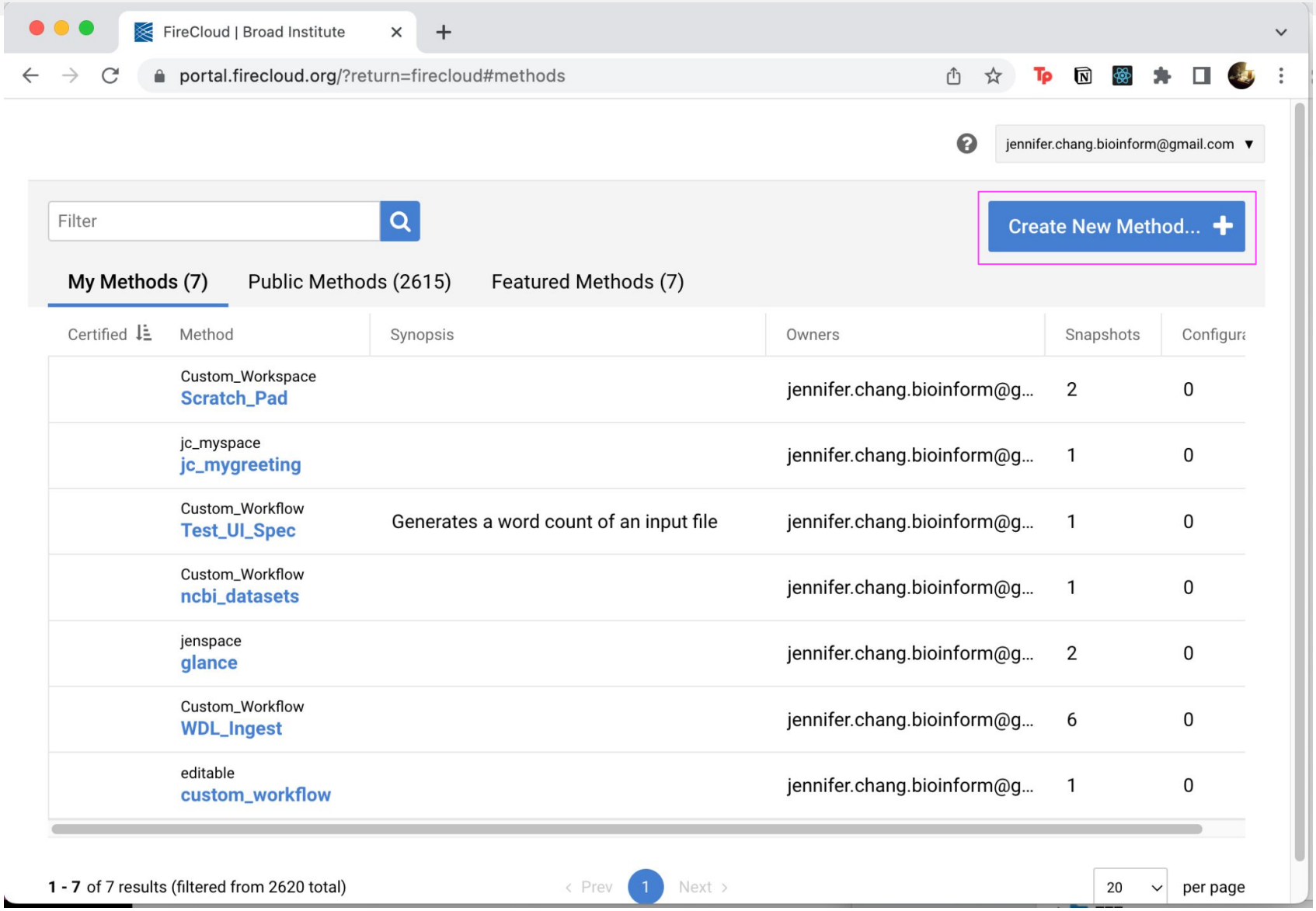
benchmarks.tsv

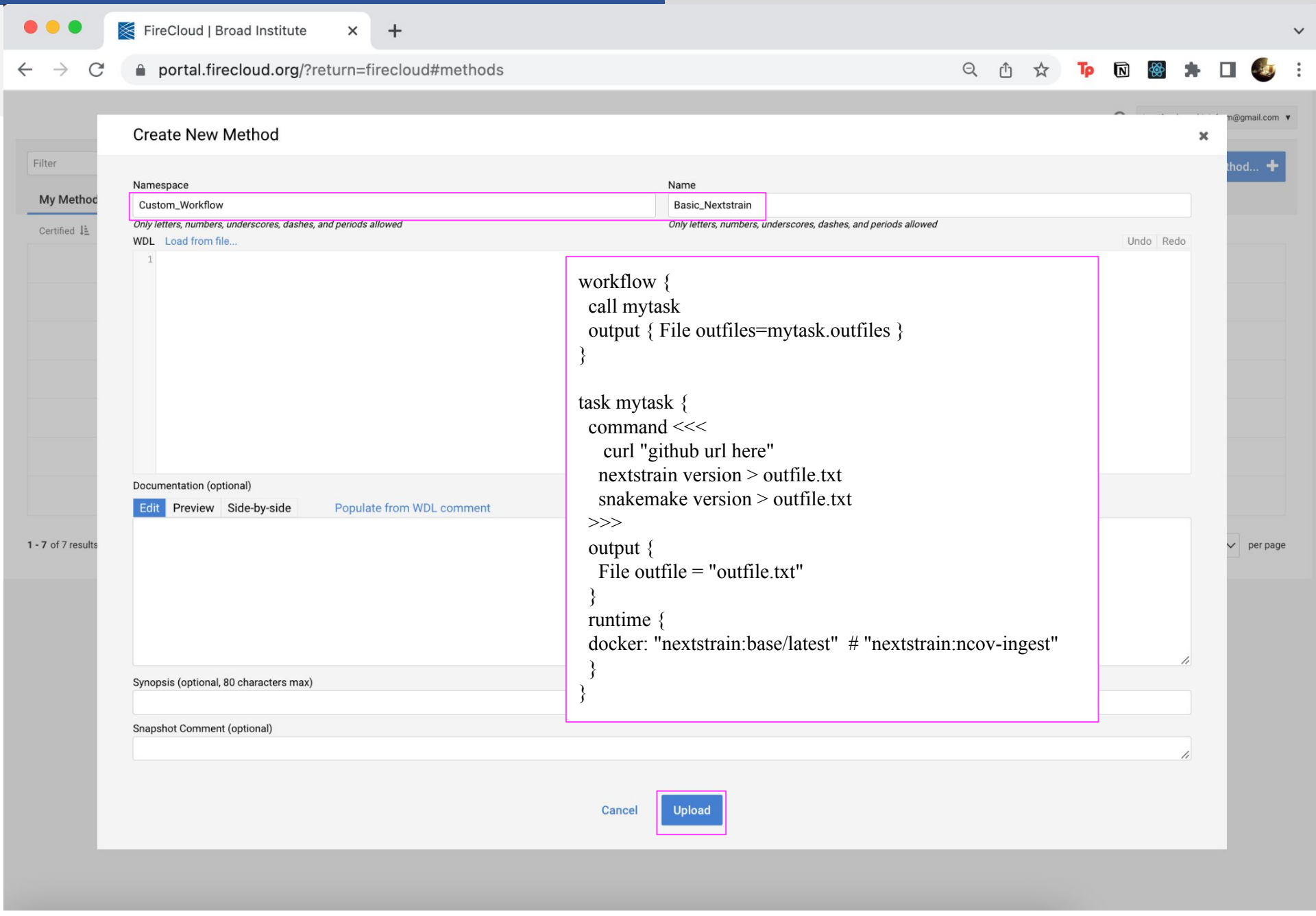
	Date	Workflow	Type	Cost	RunTime	cpu	disk_size	memory	Total_Sequences	Per_10K_Seq_Cost	New_Sequences	Per_New_Seqs	DaysSinceLastRun
2	9/19/22	GenBank_Ingest	New	$143.70	35h 19m	96	1500	180	6172949	$0.23	6172949	$0.23	-1
3	9/22/22	GenBank_Ingest	Cache	$12.70	14h 11m	16	1500	64	6182697	$0.02	9748	$13.03	3
4	9/26/22	GISAID_Ingest	New	$44.25	11h 5m	96	1500	180	13044970	$0.03	13044970	$0.03	-1
5	9/28/22	GISAID_Ingest	Cache	$9.01	10h 6m	16	1500	64	13083914	$0.01	38944	$2.31	2
6	10/13/22	GISAID_Ingest	Cache	$6.66	8h 13m	16	1500	64	13249005	$0.01	165091	$0.40	15
7	10/14/22	GenBank_Ingest	Cache	$13.21	21h 19m	16	1500	64	6238859	$0.02	56162	$2.35	22
8	10/20/22	GISAID_Ingest	Cache	$7.15	8h 11m	16	1500	64	13300565	$0.01	51560	$1.39	7
9	10/22/22	GenBank_Ingest	Cache	$14.65	16h 17m	16	1500	64	6275952	$0.02	37093	$3.95	8
10	10/29/22	GenBank_Ingest	New	$151.79	37h 23m	96	1500	180	6304264	$0.24	6304264	$0.24	-1

Agenda

- Makefiles - 1976, rule, target, dependency, command
- Snakemake - 2012 published, rule, wildcards, params, modules
- Nextflow - 2013 (?) 2017 published, DataFlow, channel, isolated work dir
- CWL (verbose) and **WDL**
  - **GitHub**
  - **Dockstore**
  - **Terra**

Terra - Firecloud




FireCloud | Broad Institute
portal.firecloud.org/?return=firecloud#methods
Create New Method
Namespace
Custom_Workflow
Only letters, numbers, underscores, dashes, and periods allowed
Name
Basic_Nextstrain
Only letters, numbers, underscores, dashes, and periods allowed
WDL
Load from file...
Undo
Redo
1
workflow {
call mytask
output { File outfiles=mytask.outfiles }
}
task mytask {
command <<<
curl "github url here"
nextstrain version > outfile.txt
snakemake version > outfile.txt
>>>
output {
File outfile = "outfile.txt"
}
runtime {
docker: "nextstrain:base/latest" # "nextstrain:ncov-ingest"
}
}
Documentation (optional)
Edit
Preview
Side-by-side
Populate from WDL comment
Synopsis (optional, 80 characters max)
Snapshot Comment (optional)
Cancel
Upload

Agenda

- Makefiles - 1976, rule, target, dependency, command
- Snakemake - 2012 published, rule, wildcards, params, modules
- Nextflow - 2013 (?) 2017 published, DataFlow, channel, isolated work dir
- CWL (verbose) and WDL
  - GitHub
  - Dockstore
  - Terra

Summary

Writing a set of Makefile Rules

```
target : dependency
    command
```

Writing a set of Snakemake Rules

```
rule NAME:
    input: dependency files
    output: target files
    shell:
        """
        command
        """
    script:
    log:
    params:
    thread:
    ....
```

Writing a set of Nextflow Processes

```
process NAME {
    input: tuple, path, value
    output: tuple, path, value
    shell:
        """
        command
        """
}

workflow WK_NAME {
  Name()
  | view
}
```

Writing a set of WDL Tasks

```
task NAME {
    input {
      File infile
      # String, Int, Double, Boolean, Array, Object
    }
    command <<<
        command
        >>>
    output {
      File outfile = "outfile.txt"
    }
    runtime {
      docker: "ubuntu"
    }
}

workflow WK_NAME {
  input {
    File textfile
  }
  call NAME {
    input: infile=textfile
  }
  output {
    File final_out = NAME.outfile
  }
}
```

• 1976 • file-based caching • local runtime	• 2012 published • file-based caching • containers, conda • named rules	• 2013 1st commit • isolated work dir caching • targeting HPC • DataFlow paradigm	• 2012 • specification must be executed on a compute engine • on Terra platform, uses hashed isolated directory caching

To Learn More

Writing a set of Makefile Rules

```
target : dependency
    command
```

Writing a set of Snakemake Rules

```
rule NAME:
    input: dependency files
    output: target files
    shell:
        """
        command
        """
    script:
    log:
    params:
    thread:
    ....
```

Writing a set of Nextflow Processes

```
process NAME {
    input: tuple, path, value
    output: tuple, path, value
    shell:
        """
        command
        """
}

workflow WK_NAME {
  Name()
  | view
}
```

Writing a set of WDL Tasks

```
task NAME {
    input {
      File infile
      # String, Int, Double, Boolean, Array, Object
    }
    command <<<
        command
        >>>
    output {
      File outfile = "outfile.txt"
    }
    runtime {
      docker: "ubuntu"
    }
}

workflow WK_NAME {
  input {
    File textfile
  }
  call NAME {
    input: infile=textfile
  }
  output {
    File final_out = NAME.outfile
  }
}
```

•	• Snakemake RTD • snakemake/snakemake-wrappers	• Nextflow RTD • nf-core/modules	• openwdl/wdl/1.0/SPEC.md

References

- Stallman, R.M. and McGrath, R., 1991. GNU Make-A Program for Directing Recompilation. note: appeared in 1976, this links to the GNU Make manual.
- Köster, J. and Rahmann, S., 2012. Snakemake—a scalable bioinformatics workflow engine. Bioinformatics, 28(19), pp.2520-2522.
- Amstutz, P., Tijanić, N., Soiland-Reyes, S., Kern, J., Stojanovic, L., Pierce, T., Chilton, J., Mikheev, M., Lampa, S., Ménager, H. and Frazer, S., 2015, July. Portable workflow and tool descriptions with the CWL. In Bioinformatics Open Source Conference.
- Amstutz, P., Crusoe, M.R., Tijanić, N., Chapman, B., Chilton, J., Heuer, M., Kartashov, A., Leehr, D., Ménager, H., Nedeljkovich, M. and Scales, M., 2016. Common workflow language, v1.0.
- Michael R. Crusoe, Sanne Abeln, Alexandru Iosup, Peter Amstutz, John Chilton, Nebojša Tijanić, Hervé Ménager, Stian Soiland-Reyes, Bogdan Gavrilović, Carole Goble, and The CWL Community. 2022. Methods Included: Standardizing Computational Reuse and Portability with the Common Workflow Language. Commun. ACM 65, 6 (June 2022), 54–63.
- Di Tommaso, P., Chatzou, M., Floden, E.W., Barja, P.P., Palumbo, E. and Notredame, C., 2017. Nextflow enables reproducible computational workflows. Nature biotechnology, 35(4), pp.316-319.
- Michael Kotliar, Andrey V Kartashov, Artem Barski, CWL-Airflow: a lightweight pipeline manager supporting Common Workflow Language, GigaScience, Volume 8, Issue 7, July 2019, giz084, https://doi.org/10.1093/gigascience/giz084
- Yuen, D., Cabansay, L., Duncan, A., Luu, G., Hogue, G., Overbeck, C., Perez, N., Shands, W., Steinberg, D., Reid, C. and Olunwa, N., 2021. The Dockstore: enhancing a community platform for sharing reproducible and accessible computational protocols. Nucleic acids research, 49(W1), pp.W624-W632.
- Gorzalski, A.J., Boyles, C., Sepcic, V., Verma, S., Sevinsky, J., Libuit, K., Van Hooser, S. and Pandori, M.W., 2022. Rapid repeat infection of SARS-CoV-2 by two highly distinct delta-lineage viruses. Diagnostic Microbiology and Infectious Disease, 104(1), p.115747.
- Perkel, J.M., 2019. Workflow systems turn raw data into scientific knowledge. *Nature*, *573*(7772), pp.149-151.

GitHub

- https://github.com/j23414/compare_workflows