

Introduction to Workflow Description Language (WDL)



Beth Sheets & Juan Pablo Ramos Barroso

Data Sciences Platform, Broad Institute of MIT & Harvard

Reproducing, validating, and extending analyses is essential to science

But reproducing a computational analysis is hard!

- How do I install the software and its dependencies?
- Am I using all the same versions of the software?
- Why is this script in Python and that script in R?
- Can I know that this runs exactly the same on my computer vs my colleague's?

Workflow languages were designed to help

Consistency / Reproducibility

- Across batches
- Across users
- Across platforms

Ease of use:

- Reduces user error
- Helps you scale up to easily process thousands of samples

Workflows are a community resource

There are thousands of workflows published in the community repository,
[Dockstore.org](https://dockstore.org)!

Sharing your workflows with the community:

- Centralizes knowledge
- Builds community resources
- Helps you get feedback
- Makes your scientific method citable
- Gives you metrics to include in proposals



A draft human pangenome reference

[Wen-Wei Liao](#), [Mobin Asri](#), [Jana Ebler](#), [Daniel Doerr](#), [Marina Haukness](#), [Glenn Hickey](#), [Shuangjia Lu](#),
[Julian K. Lucas](#), [Jean Monlong](#), [Haley J. Abel](#), [Silvia Buonaiuto](#), [Xian H. Chang](#), [Haoyu Cheng](#), [Justin](#)

[Chu](#), [Vincen](#):

[Garg](#), [Cristia](#)

Phased assembly pipeline







+ Show a


[Nature](#) **617**,


286k Acces

We describe the main automated and manual steps taken before, during and after assembly. A combined set of workflow description language (WDL) formatted assembly workflows is available from **Dockstore** that captures each of the steps for filtering adapter-contained reads and running Hifiasm



(<https://dockstore.org/organizations/HumanPangenome/collections/Hifiasm>). All




 **Dockstore**  Search  Organizations  About  Docs  Forum [Login](#) [Register](#)

 [Organizations](#) / Human Pangenome Reference Consortium





Human Pangenome Reference Consortium

  7
Assembly and analysis workflows for generating Human Pangenome References

 Collections **5**  Members **5**  Updates **10**

About the Organization

 <https://humanpangenome.org/>



Flagger and Secphase

A read-based pipeline for evaluating dual/diploid assemblies

Which workflow language should I use?

Workflow Description Language (WDL), Common Workflow Language (CWL), Nextflow, Snakemake...they are all helpful!

What workflows are available already that are similar to my analysis?

- Check out [Dockstore.org](https://dockstore.org) for workflows shared by the community.

What language are my colleagues using?

- You have community to learn from!

Workflow Description Language (WDL)



- A language to describe the steps of an analysis
- Created to be human readable and easy to write
- Bioinformatics / genomics focus due to it being developed by the Broad Institute (but can be used for any data science)
- Open source, community-developed (by OpenWDL)

What WDL is *NOT*



- Not a programming language
- Not a tool that does analysis
- Not a GUI or platform
- Not just for Broad pipelines
- Not an alternative to containers or scripting languages

Recipe for a Workflow



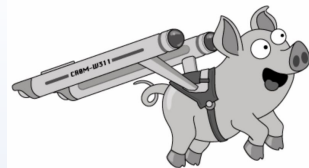
Container



**Descriptor
file (.wdl)**



**Parameter
file**



Cromwell

Recipe for a Workflow



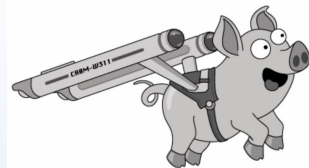
Container



**Descriptor
file (.wdl)**



**Parameter
file**



Cromwell

Intro to containers



Container

- Packaged up tools and all of its dependencies
- Makes software portable from one computing environment to another.

- Docker is commonly used by cloud platforms
- Singularity is common for HPCs because it doesn't require root permissions

Check out [BioContainers.pro](https://biocontainers.pro) from the BioConda community. They also automate releasing Singularity containers for reuse by the community in workflows.

Recipe for a Workflow



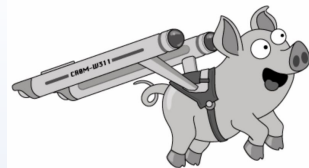
Container



**Descriptor
file (.wdl)**



**Parameter
file**



Cromwell

Intro to workflow descriptor files



This file describes your software commands as “tasks”, specifying inputs, outputs, and the environment (container).

Tasks are organized together to create workflows.

```
task task_A {
```

```
  File ref  
  File in  
  String id
```

```
  command {
```

```
    do_stuff -R ${ref} -I ${in} -O ${id}.ext
```

```
  }
```

```
  runtime {
```

```
    docker: "my_project/do_stuff:1.2.0"
```

```
  }
```

```
  output {
```

```
    File out= "${id}.ext"
```

```
  }
```

```
}
```

Review: Workflow Description Language (WDL)

```
version 1.0
```

```
workflow MyWorkflowName {
```

```
...
```

```
}
```

```
task task A {
```

```
...
```

```
}
```

```
task task B {
```

```
...
```

```
}
```

```
version 1.0
```

```
workflow MyWorkflowName {
```

```
input {
```

```
File my_ref
```

```
File my_input
```

```
String name
```

```
}
```

```
call task A {
```

```
input:
```

```
ref = my_ref
```

```
in = my_input
```

```
id = name
```

```
}
```

```
call task B {
```

```
input:
```

```
in = task_A.out_name_A
```

```
}
```

```
output {
```

```
out_A = task_A.out_name_A
```

```
out_B = task_B.out_name_B
```

```
}
```

```
}
```

```
task task_A {
```

```
...
```

```
...
```

Review: Workflow Description Language (WDL)

```
version 1.0
```

```
workflow MyWorkflowName {
```

```
    ...
```

```
}
```

```
task task_A {
```

```
    ...
```

```
}
```

```
task task_B {
```

```
    ...
```

```
}
```

```
task task_A {
```

```
    input {
```

```
        File ref
```

```
        File in
```

```
        String id
```

```
    }
```

```
    command {
```

```
do_stuff -R ~{ref} -I ~{in} -O  
~{id}.ext
```

```
    }
```

```
    runtime {
```

```
        docker: 'my_project/do_stuff:1.2.0'
```

```
    }
```

```
        File out_name_A = "~{id}.ext"
```

```
    }
```

```
}
```

```
task task_B {
```

```
    ...
```

Review: Workflow Description Language (WDL)

```
version 1.0
```

```
workflow MyWorkflowName {
```

```
...
```

```
}
```

```
task task_A {
```

```
...
```

```
}
```

```
task task_B {
```

```
...
```

```
}
```

```
version 1.0
```

```
import /path/to/wdl/with/new_wf.wdl as
```

```
new_wf
```

```
work
```

```
input {
```

```
File my_ref
```

```
File my_input
```

```
String name
```

```
}
```

```
call task_A {
```

```
}
```

```
call task_B {
```

```
}
```

```
call new_wf.task_C {
```

```
out_A = task_A.out_name_A
```

```
out_B = task_B.out_name_B
```

```
out_C = task_C.out_name_C
```

```
output {
```

```
}
```

```
}
```


Recipe for a Workflow



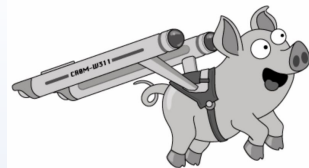
Container



**Descriptor
file (.wdl)**



**Parameter
file**



Cromwell

Workflow Inputs

Example Workflow

```
version1.0

workflow wf {
  input {
    Int int_val
    Int int_val2 = 10
    Array[Int] my_ints
    File ref_file
  }
  String not_an_input = "hello"
  call t1 {
    input: x = int_val
  }
  call t2 {
    input: x = int_val, t=t1.count
  }
}
```

Example JSON Input

```
{
  "wf.int_val": 3,
  "wf.my_ints": [5,6,7,8],
  "wf.ref_file": "/path/to/file.txt"
}
```

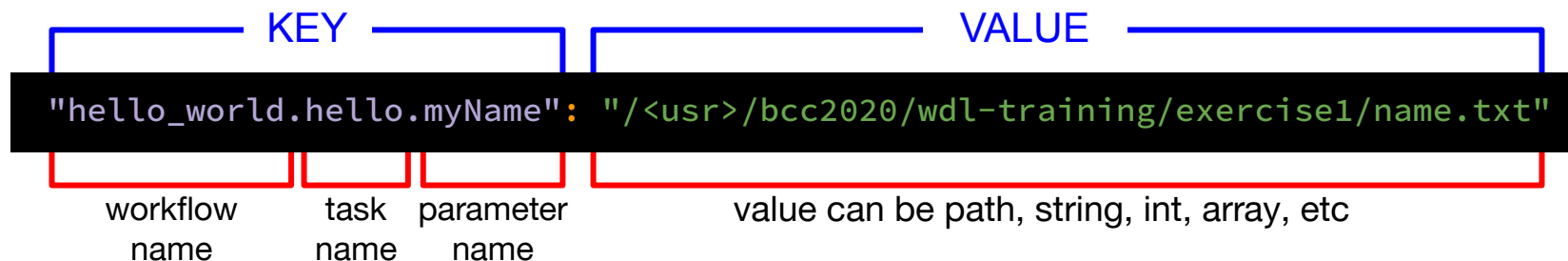
1. Identify all workflow inputs.
2. For each input, determine the input type (str, int, etc).
3. Create a JSON file with key/value pairs for each workflow input.
 - a. Key format : <workflow name>.<input name>
4. Set the value of each key/value pair to the value of the workflow input.

Parameter file (json format)

Specifies inputs, outputs, and computing resources that are specific to your analysis. This is where you can specify cloud vs local environment logic.

hello.json

```
1 {  
2   "hello_world.hello.myName": "/<usr>/bcc2020/wdl-training/exercise1/name.txt"  
3 }
```



Recipe for a Workflow



Container



**Descriptor
file (.wdl)**

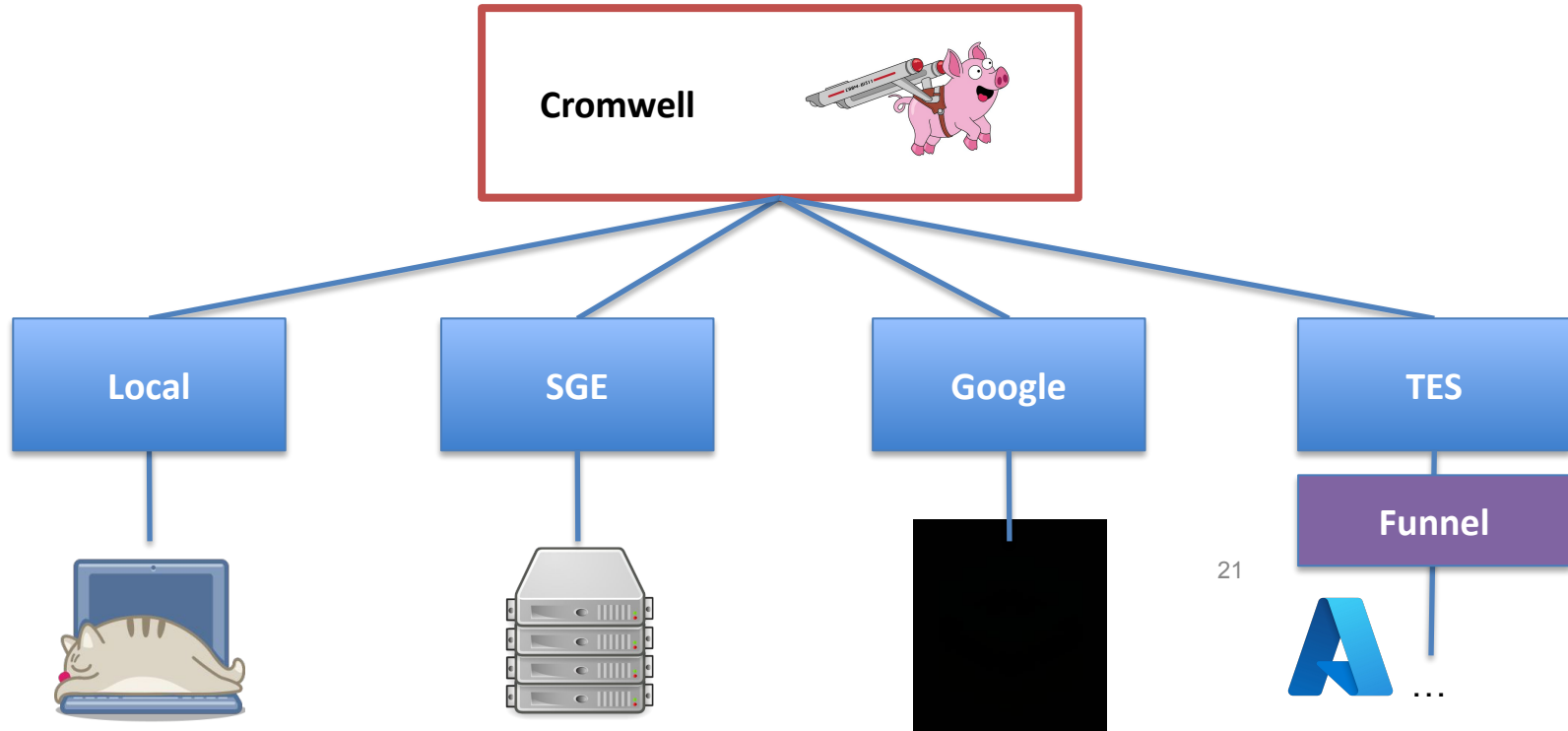


**Parameter
file**



Cromwell

What is Cromwell



21

Two main ways to run Cromwell

One-off

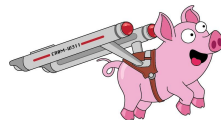
- Simple self-contained command

```
cromwell run \  
  hello.wdl \  
  --inputs hello_inputs.json
```

- Appropriate for independent analysts

Server mode

- API endpoints
- More scalable
- Some devops needs
- Appropriate for production environments
- Call-caching!
(aka ka-ching")



Steps to running a WDL in standalone mode

- Validate syntax

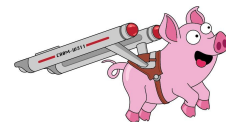
```
womtool validate hello.wdl
```

- Generate inputs JSON

```
womtool inputs hello.wdl > hello_inputs.json
```

- Run

```
cromwell run hello.wdl --inputs hello_inputs.json
```



Resources for getting started

Resource

Link

OpenWDL GitHub

<https://github.com/openwdl/wdl>

WDL 1.0 Spec

<https://github.com/openwdl/wdl/blob/main/versions/1.0/SPEC.md>

WDL 1.0 Docs (wdl-docs)

<https://wdl-docs.readthedocs.io/en/stable/>

WDL Cookbook

https://wdl-docs.readthedocs.io/en/stable/WDL/chain_tasks_together/

Learn-WDL Videos

<https://github.com/openwdl/learn-wdl>

WARP WDL Best Practices

https://broadinstitute.github.io/warp/docs/Best_practices/GC_cost_optimization

OpenWDL Slack

<https://openwdl.slack.com/>

Examples in WARP (WDL Analysis
Research Pipelines; Terra-optimized)

<https://broadinstitute.github.io/warp/>

Theiagen Genomics WDL Workshop

https://github.com/theiagen/wm_training_202205

Task examples in BioWDL

<https://biowdl.github.io/>

Questions?