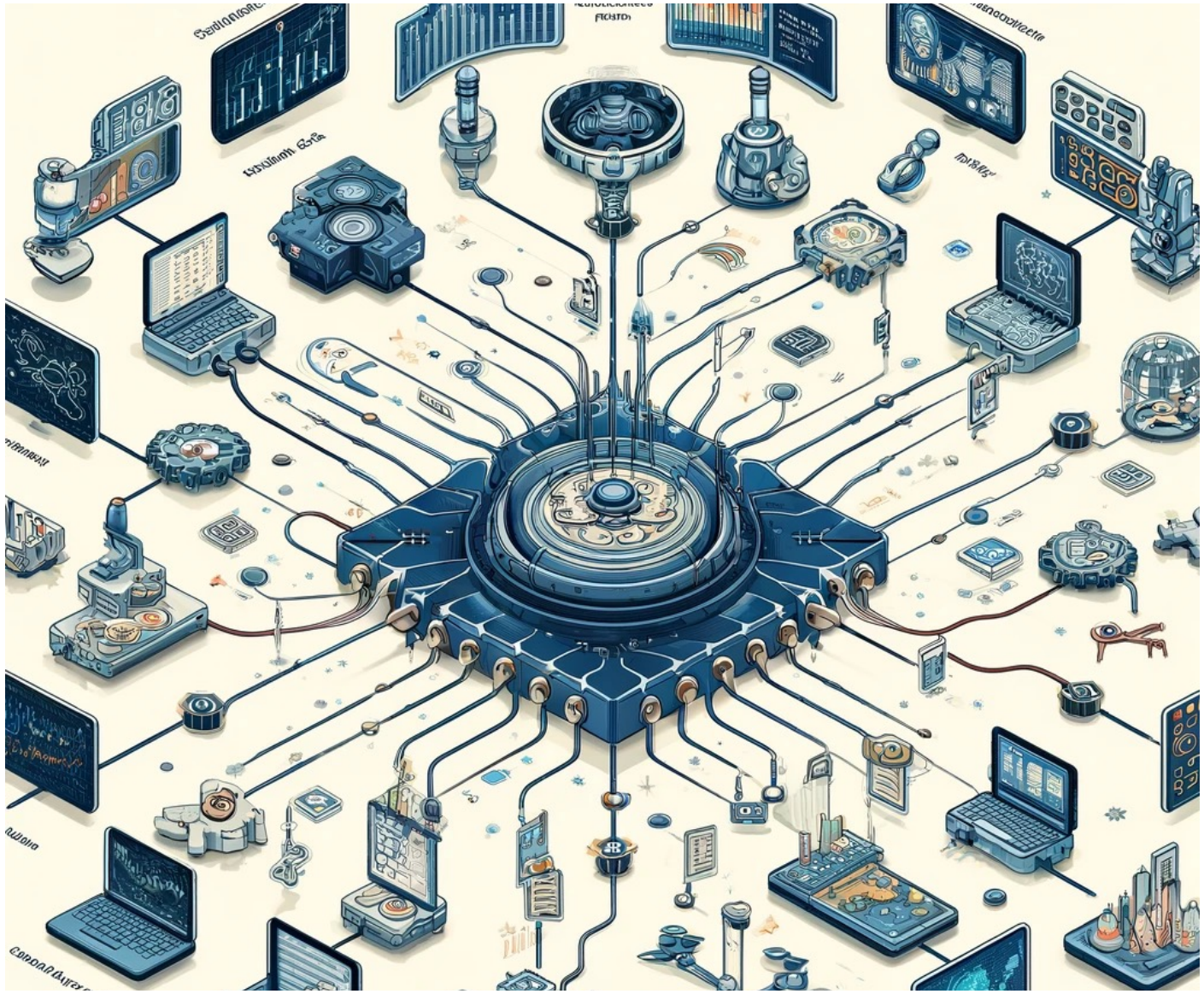
Exploring Scientific Workflows with CWL and dispel4py

Module 1.a

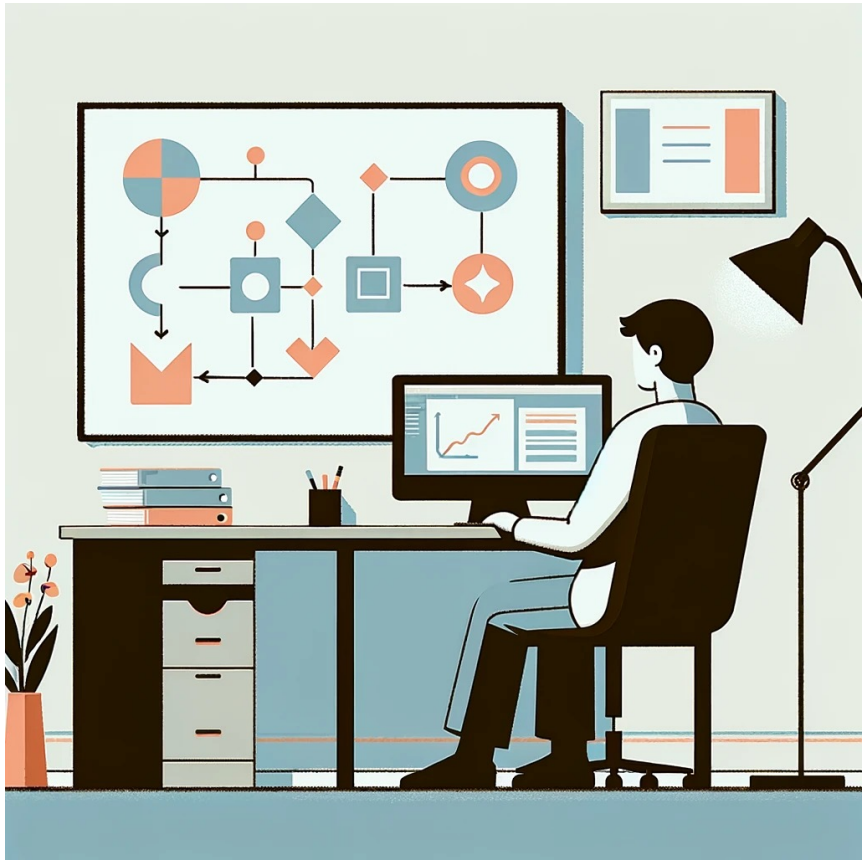
- Dr. Rosa Filgueira
- Lecturer at the School of Computer Science
- University of St Andrews
- rf208@st-andrews.ac.uk
- rosa.filgueira.vicente@gmail.com



Module 2.a- Creating Workflows with CWL

1. Overview
    i. What is CWL
    ii. Why was created
    iii. Components
    iv. File structure
    v. An example of a bioinformatics use case
    vi. Setup
    vii. Google Colab CWL tutorial

2. Hand On- Exercises
    1. Building CWL Tools in Google Colab
        1. (You can just do one)
    2. Creating a CWL workflow in Google Colab
        1. Only if you have extra time



1. Overview

COMMON
WORKFLOW
LANGUAGE

What is CWL ?

COMMON WORKFLOW LANGUAGE

- It is a way to describe command line tools and connect them together to create workflows
- Scripting-Glue type
- Semantic Focus
  - With CWL every component is given a formal description in a YAML format
- We can run CWL workflows with different tools:
  - cwltool, cwl-runner, toil, arvados ....

- Why do we want to learn to use CWL?
  - Explicit IO
  - Repeatability, modularity and scalability
  - Parallelism and performace

Why was it created ?

COMMON WORKFLOW LANGUAGE

- Stated in 2015
- Community based standard
- Collaborations (innovation)
- Publications – reproducibility

- Who is using it ?

wellcome trust sanger institute
Galaxy PROJECT
Global Alliance for Genomics & Health
HARVARD T.H. CHAN SCHOOL OF PUBLIC HEALTH
UCSC
Institut Pasteur
BROAD INSTITUTE

Components

COMMON WORKFLOW LANGUAGE

- a CWL file (.cwl)
  - Describes what is going to run and what inputs the program takes
- a YAML (.yml) file
  - Holds the values that the workflow will be executed with

- A ‘tool’ is a task to preform within a CWL workflow

- CWL is written with YAML:
  - It is similar to JSON
    - easier to work with because it is more human readable.
  - YAML is based on key: value pairs
    - where each key is a string (text) and each value is a primitive type, an array, or an object.

Example – Hello World

COMMON WORKFLOW LANGUAGE

hello_world.cwl

```yaml
cwlVersion: v1.2

# What type of CWL process we have in this document.
class: CommandLineTool
# This CommandLineTool executes the linux "echo" command-line tool.
baseCommand: echo

# The inputs for this process.
inputs:
  message:
    type: string
    # A default value that can be overridden, e.g. --message "Hola mundo"
    default: "Hello World"
    # Bind this message value as an argument to "echo".
    inputBinding:
      position: 1
outputs: []
```

- The example is a wrapper for the echo command-line tool.
- Running the workflow produces → "Hello World".

Setup

It is recommended to setup a virtual environment before installing cwltool

```
$virtualenv env
$source env/bin/activate
```

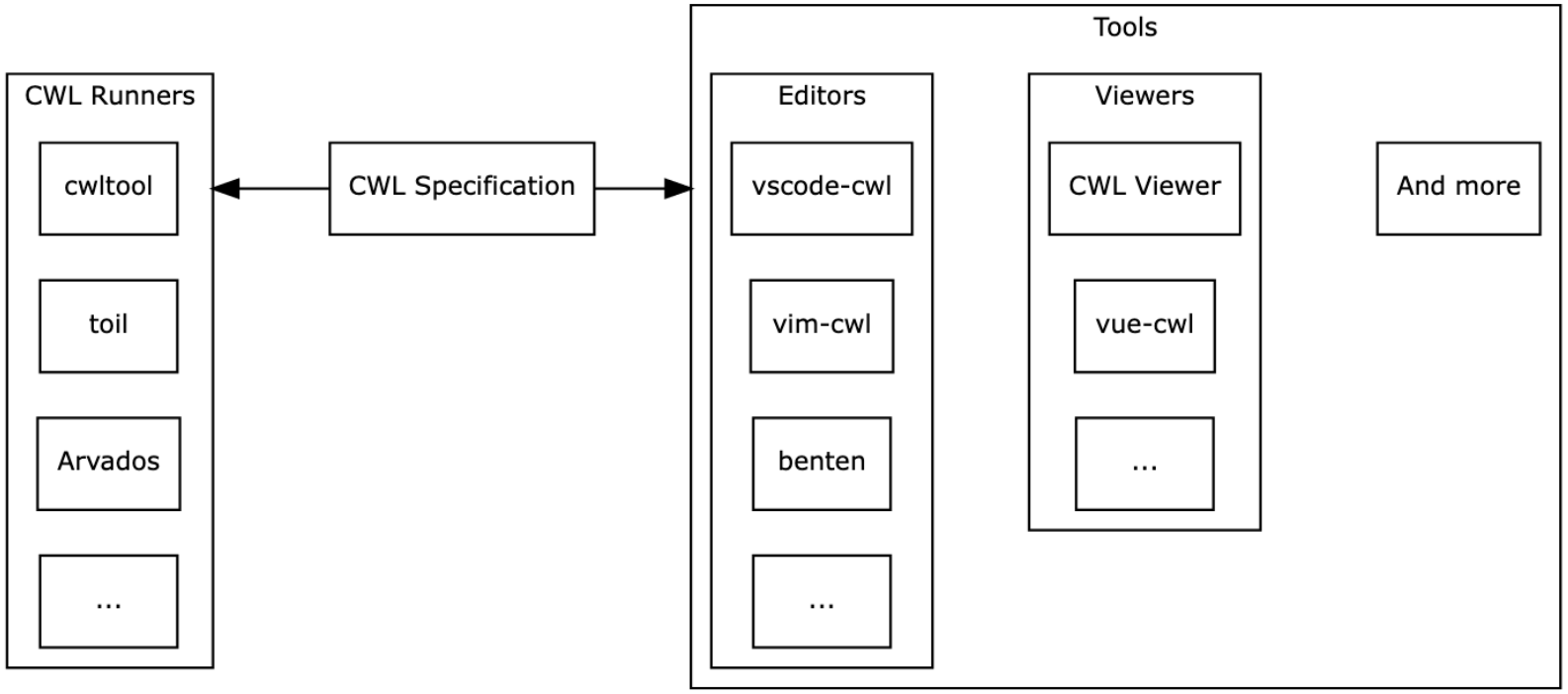
Install the reference implementation from PyPi

```
$pip install cwlref-runner
```

We can also install and use instead cwltool, which another tool for running cwl workflows

```
$pip install cwltool
```

Implementations



CWL is just a specification – we can run CWL workflows, with many tools (cwl runners).
And there are many other tools compatible with CWL

Running “Hello World”

COMMON WORKFLOW LANGUAGE

Running `hello_world.cwl` with `cwltool`.

```
$ cwltool hello_world.cwl
INFO /opt/hostedtoolcache/Python/3.9.18/x64/bin/cwltool 3.1.20240112164112
INFO Resolved 'hello_world.cwl' to 'file:///home/runner/work/user_guide/user_guide/
INFO [job hello_world.cwl] /tmp/_rjof8sp$ echo \
    'Hello World'
Hello World
INFO [job hello_world.cwl] completed success
{}INFO Final process status is success
```

We can run ‘hello_world.cwl’ without specifying any option

Running “Hello World”

Running `hello_world.cwl` with `cwltool` passing an input parameter.

```
$ cwltool hello_world.cwl --message="Hola mundo"
INFO /opt/hostedtoolcache/Python/3.9.18/x64/bin/cwltool 3.1.20240112164112
INFO Resolved 'hello_world.cwl' to 'file:///home/runner/work/user_guide/user_guide/
INFO [job hello_world.cwl] /tmp/mk_0n_30$ echo \
    'Hola mundo'
Hola mundo
INFO [job hello_world.cwl] completed success
{}INFO Final process status is success
```

Or we can override the default value of the input parameter message

Running “Hello World”

`hello_world-job.json`

```
{
  "message": "こんにちは世界"
}
```

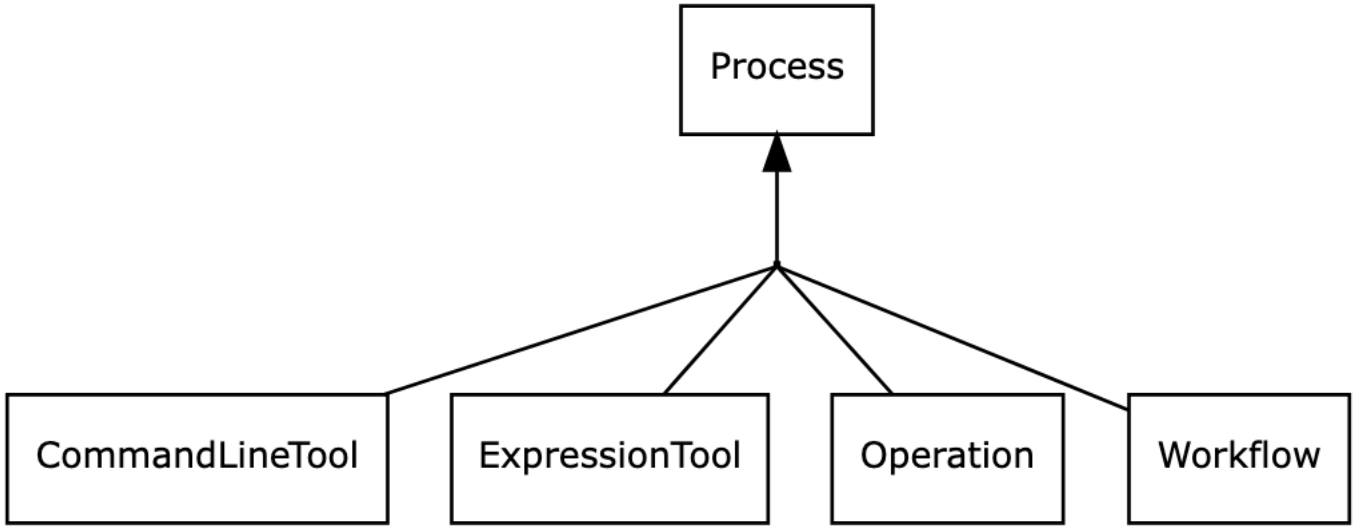
You can use this Inputs Object file now to execute the “Hello World” workflow:

Passing an Inputs Object file to `cwltool`.

```
$ cwltool hello_world.cwl hello_world-job.json
INFO /opt/hostedtoolcache/Python/3.9.18/x64/bin/cwltool 3.1.20240112164112
INFO Resolved 'hello_world.cwl' to 'file:///home/runner/work/user_guide/user_guide/
INFO [job hello_world.cwl] /tmp/8fmewehd$ echo \
    こんにちは世界
こんにちは世界
INFO [job hello_world.cwl] completed success
{}INFO Final process status is success
```

Processes in CWL

- A process is a computing unit that takes inputs and produces outputs. There are four types
  - A command-line tool.
  - An expression tool.
  - An operation.
  - A workflow.



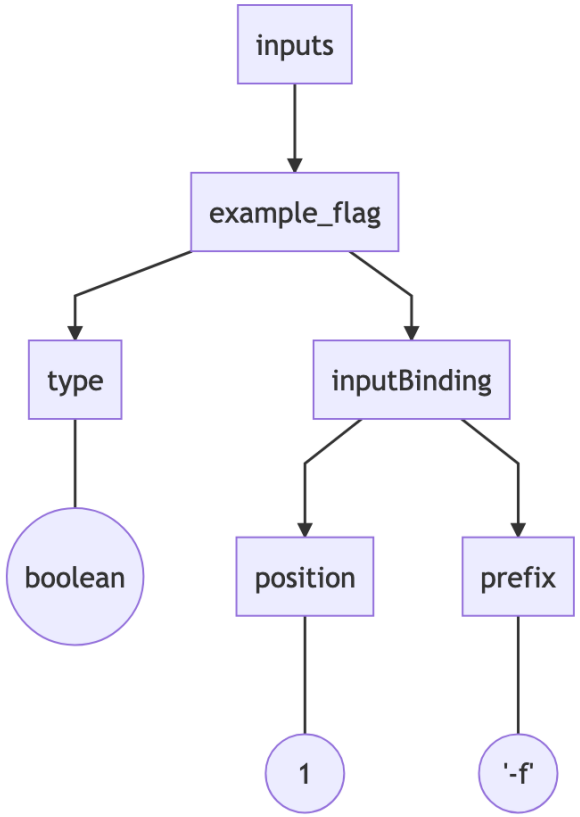
File Structure

- cwlVersion:
  - describes the version of cwl being used
- class:
  - describes what the program is (e.g. CommandLineTool,Workflow)
- baseCommand:
  - provides the name of the program that will actually run
- inputs:
  - declares the inputs of the program
- outputs:
  - declares the outputs of the program
- records:
  - declares relationships between programs/parameters
- requirements:
  - declares special requirements needed by the program such as dependencies
- steps:
  - used for the actual creation of workflows and linking programs together.

Basic concepts - I

- Files written in YAML consists of a set of *key-value pairs*
  - first_name: Bilbo
  - last_name: Baggins
  - age_years: 111
- Comments with #
- Nested structures in CWL – represented with '*Maps*'

```yaml
cwlVersion: v1.0
class: CommandLineTool
baseCommand: echo
inputs: # this key has an object value
  example_flag: # so does this one
    type: boolean
    inputBinding: # and this one too
      position: 1
      prefix: -f
```



Basic concepts - II

- We can also use *Arrays* - multiple values or objects for a single key

  touchfiles:

  - foo.txt
  - bar.dat
  - baz.txt

Basic concepts - III

- Command Line Tool
  - Run by itself or as a workflow step
  - It is a wrapper for a command like *ls*, *echo*, *tar*, etc.
  - Defined in the **`baseCommand`** attribute

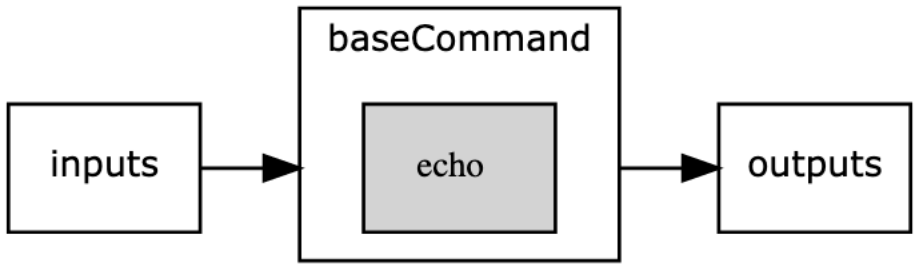
echo.cwl

```yaml
cwlVersion: v1.2
class: CommandLineTool

baseCommand: echo

stdout: output.txt

inputs:
  message:
    type: string
    inputBinding: {}
outputs:
  out:
    type: string
    outputBinding:
      glob: output.txt
      loadContents: true
      outputEval: $(self[0].contents)
```



Basic concepts - IV

- Inputs – it is a list of **input parameters** that control how to run the tool. Each parameter:
  - **id** for the name of parameter
  - **type** describing what types of values are valid for that parameter:
    - *string*, *boolean*, *int*, *long*, *float*, *double*, *null, array* and *record, File*, *Directory* and *Any*.

inp.cwl

```
#!/usr/bin/env cwl-runner
cwlVersion: v1.2
class: CommandLineTool
baseCommand: echo
inputs:
  example_flag:
    type: boolean
    inputBinding:
      position: 1
      prefix: -f
  example_string:
    type: string
    inputBinding:
      position: 3
      prefix: --example-string
  example_int:
    type: int
    inputBinding:
      position: 2
      prefix: -i
      separate: false
  example_file:
    type: File?
    inputBinding:
      prefix: --file=
      separate: false
      position: 4

outputs: []
```

inp-job.yml

```
example_flag: true
example_string: hello
example_int: 42
example_file:
  class: File
  path: whale.txt
```

```
$ cwltool inp.cwl inp-job.yml
INFO /opt/hostedtoolcache/Python/3.9.18/x64/bin/cwltool 3.1.20240112164112
INFO Resolved 'inp.cwl' to 'file:///home/runner/work/user_guide/user_guide/src/_inc
INFO [job inp.cwl] /tmp/vhsa1v9t$ echo \
    -f \
    -i42 \
    --example-string \
    hello \
    --file=/tmp/ie9pfhn1/stg85122967-82b2-4468-bfeb-dede9dc7225a/whale.txt
-f -i42 --example-string hello --file=/tmp/ie9pfhn1/stg85122967-82b2-4468-bfeb-dede
INFO [job inp.cwl] completed success
{}INFO Final process status is success
```

Basic concepts - V

- inputBinding – it is optional
  - whether and how the input parameter should appear on the tool's command line
  - If inputBinding is missing, the parameter does not appear on the command line.

```
example_flag:
  type: boolean
  inputBinding:
    position: 1
    prefix: -f
```

inp-job.yml

```
example_flag: true
```

- Boolean types are treated as a flag.
  - If the input parameter "example_flag" is "true", then prefix will be added to the command line. If false, no flag is added.

More about inputBinding here

Basic concepts - VI

- Array Inputs – 2 ways:

  1) provide **type field** with **type: array** and **items** defining the **valid data types**.
  2) **brackets []** added after the **type name** to indicate that input parameter is array of that type.

array-inputs.cwl

```
#!/usr/bin/env cwl-runner
cwlVersion: v1.2
class: CommandLineTool
inputs:
  filesA:
    type: string[]
    inputBinding:
      prefix: -A
      position: 1

  filesB:
    type:
      type: array
      items: string
      inputBinding:
        prefix: -B=
        separate: false
    inputBinding:
      position: 2

  filesC:
    type: string[]
    inputBinding:
      prefix: -C=
      itemSeparator: ","
      separate: false
      position: 4

outputs:
  example_out:
    type: stdout
stdout: output.txt
baseCommand: echo
```

array-inputs-job.yml

```
filesA: [one, two, three]
filesB: [four, five, six]
filesC: [seven, eight, nine]
```

```
$ cwltool array-inputs.cwl array-inputs-job.yml
INFO /opt/hostedtoolcache/Python/3.9.18/x64/bin/cwltool 3.1.20240112164112
INFO Resolved 'array-inputs.cwl' to 'file:///home/runner/work/user_guide/user_guide/
INFO [job array-inputs.cwl] /tmp/hotijy6e$ echo \
    -A \
    one \
    two \
    three \
    -B=four \
    -B=five \
    -B=six \
    -C=seven,eight,nine > /tmp/hotijy6e/output.txt
INFO [job array-inputs.cwl] completed success
{
    "example_out": {
        "location": "file:///home/runner/work/user_guide/user_guide/src/_includes/cw
        "basename": "output.txt",
        "class": "File",
        "checksum": "sha1$91038e29452bc77dcd21edef90a15075f3071540",
        "size": 60,
        "path": "/home/runner/work/user_guide/user_guide/src/_includes/cwl/inputs/ou
    }
}INFO Final process status is success
```

More about Array Inputs here

Basic concepts - VII

- Sometimes tools require additional command line options that don’t correspond exactly to input parameters.

```
#!/usr/bin/ cwl-runner

cwlVersion: v1.0
class: CommandLineTool
baseCommand: tar
arguments: [-x, -v, -z, -f]
inputs:
  tarfile:
    type: File
    inputBinding:
      position: 1
    label: the file to be decompressed

outputs:
  extractfile:
    type: File
    outputBinding:
      glob: "*"
```

```
tarfile:
  class: File
  path: christmas_carol.tar.gz
```

```
$ tar -xvzf christmas_carol.tar.xz
```

More about Additional Parameters here

Basic concepts - VIII

- outputs – list of **output parameters** that should be returned. Each parameter:
  - **id** for the name of parameter
  - **type** describing what types of values are valid for that parameter:
    - *string*, *boolean*, *int*, *long*, *float*, *double*, *null, array* and *record, File*, *Director*.

tar.cwl

```
#!/usr/bin/env cwl-runner
cwlVersion: v1.2
class: CommandLineTool
baseCommand: [tar, --extract]
inputs:
  tarfile:
    type: File
    inputBinding:
      prefix: --file
outputs:
  example_out:
    type: File
    outputBinding:
      glob: hello.txt
```

tar-job.yml

```
tarfile:
  class: File
  path: hello.tar
```

```
$ cwltool tar.cwl tar-job.yml
INFO /opt/hostedtoolcache/Python/3.9.18/x64/bin/cwltool 3.1.20240112164112
INFO Resolved 'tar.cwl' to 'file:///home/runner/work/user_guide/user_guide/src/_i
INFO [job tar.cwl] /tmp/uz8gyqz5$ tar \
    --extract \
    --file \
    /tmp/z_lhhzdy/stgf915cdc0-161f-4924-8c8f-9daa62ae55e4/hello.tar
INFO [job tar.cwl] completed success
{
    "example_out": {
        "location": "file:///home/runner/work/user_guide/user_guide/src/_includes
        "basename": "hello.txt",
        "class": "File",
        "checksum": "sha1$da39a3ee5e6b4b0d3255bfef95601890afd80709",
        "size": 0,
        "path": "/home/runner/work/user_guide/user_guide/src/_includes/cwl/output
    }
}INFO Final process status is success
```

More about Outputs here

Basic concepts - IX

- The field outputBinding describes how to set the value of each output parameter.

```yaml
outputs:
  example_out:
    type: File
    outputBinding:
      glob: hello.txt
```

The glob field consists of the pattern to match file names in the output directory.

This can simply be the file's exact name.

But if you don't know the name of the file in advance, you can use a wildcard pattern like glob: '*.txt'; or '*'.

tar-job.yml

```yaml
tarfile:
  class: File
  path: hello.tar
```

```
$ cwltool tar.cwl tar-job.yml
INFO /opt/hostedtoolcache/Python/3.9.18/x64/bin/cwltool 3.1.20240112164112
INFO Resolved 'tar.cwl' to 'file:///home/runner/work/user_guide/user_guide/src/_i
INFO [job tar.cwl] /tmp/uz8gyqz5$ tar \
    --extract \
    --file \
    /tmp/z_lhhzdy/stgf915cdc0-161f-4924-8c8f-9daa62ae55e4/hello.tar
INFO [job tar.cwl] completed success
{
    "example_out": {
        "location": "file:///home/runner/work/user_guide/user_guide/src/_includes
        "basename": "hello.txt",
        "class": "File",
        "checksum": "sha1$da39a3ee5e6b4b0d3255bfef95601890afd80709",
        "size": 0,
        "path": "/home/runner/work/user_guide/user_guide/src/_includes/cwl/output
    }
}INFO Final process status is success
```

More about Outputs here

Basic concepts - X

- To capture a tool's standard output stream
  - Add the **stdout field** with **the name of the file where the output stream should go**.
  - Add **type: stdout** on the corresponding output parameter

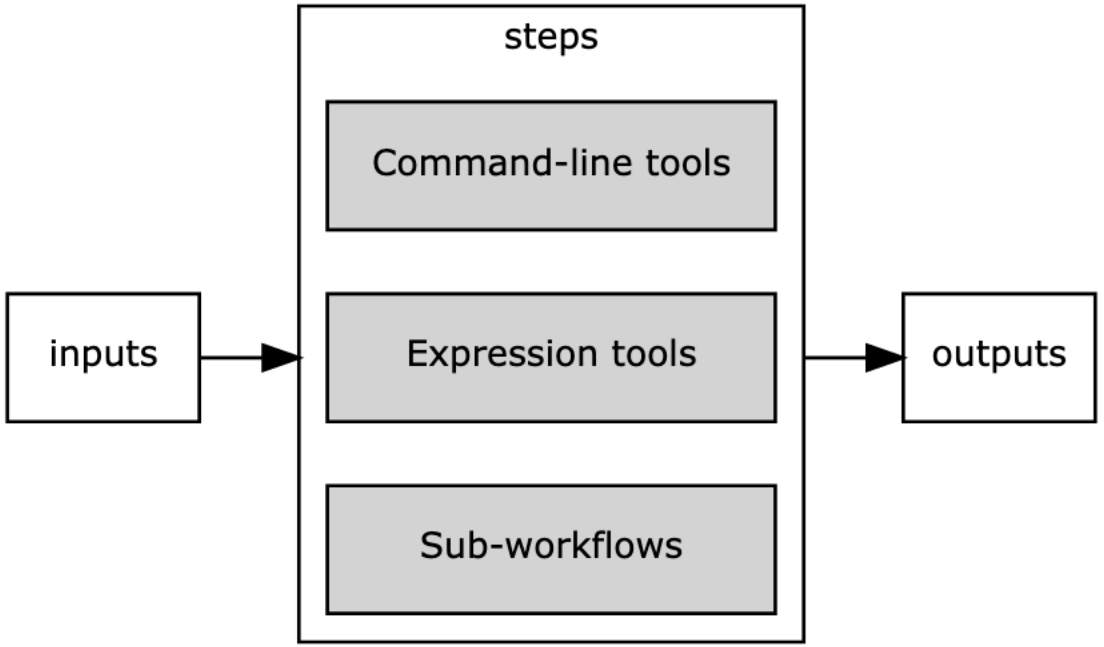
stdout.cwl

```
#!/usr/bin/env cwl-runner
cwlVersion: v1.2
class: CommandLineTool
baseCommand: echo
stdout: output.txt
inputs:
  message:
    type: string
    inputBinding:
      position: 1
outputs:
  example_out:
    type: stdout
```

More about Capturing Standard Output here

Basic concepts - XI

- A workflow is a CWL processing unit that executes
  - command-line tools, expression tools, or workflows (sub-workflows) as steps.
- It must have inputs, outputs, and steps defined in the CWL document.



More about Workflows here

Basic concepts - XI

- A workflow is a CWL processing unit that executes
  - command-line tools, expression tools, or workflows (sub-workflows) as steps.
- It must have inputs, outputs, and steps defined in the CWL document.

echo-uppercase.cwl

```yaml
cwlVersion: v1.2
class: Workflow

requirements:
  InlineJavascriptRequirement: {}

inputs:
  message: string

outputs:
  out:
    type: string
    outputSource: uppercase/uppercase_message

steps:
  echo:
    run: echo.cwl
    in:
      message: message
    out: [out]
  uppercase:
    run: uppercase.cwl
    in:
      message:
        source: echo/out
    out: [uppercase_message]
```

More about Workflows [here]

Example of a Simple Workflow

COMMON WORKFLOW LANGUAGE

- This workflow extracts a java source file from a tar file and then compiles it.

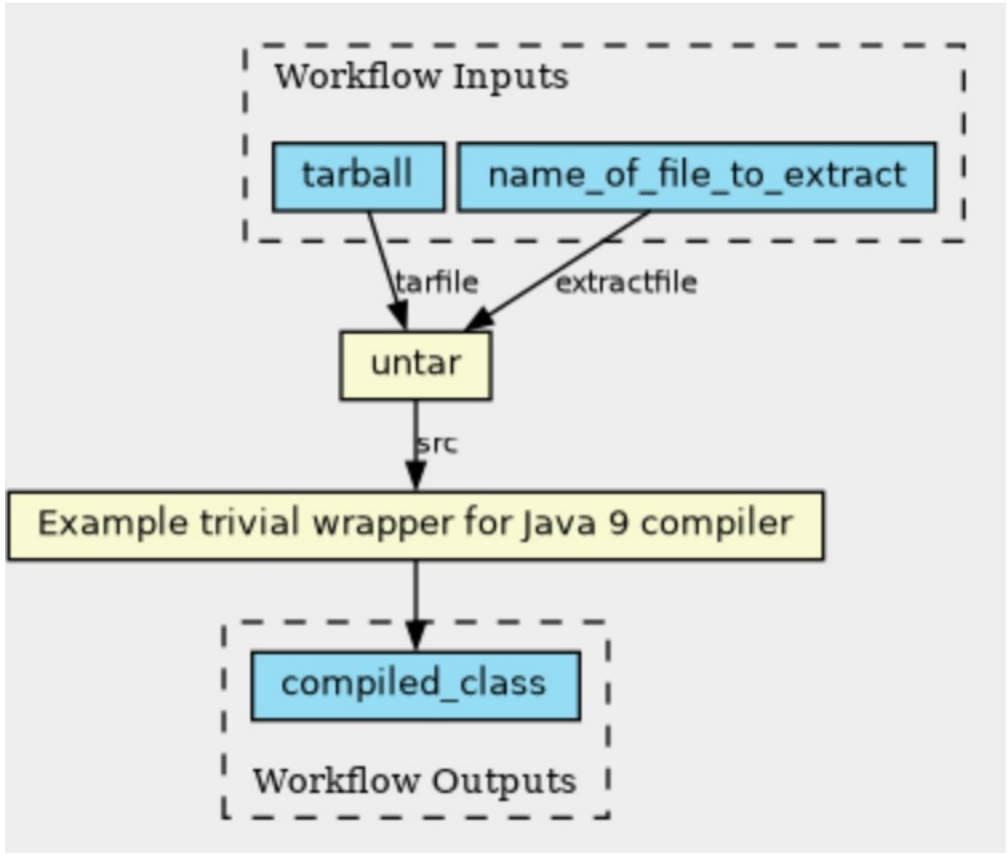
1st-workflow.cwl

```
#!/usr/bin/env cwl-runner
cwlVersion: v1.2
class: Workflow
inputs:
  tarball: File
  name_of_file_to_extract: string

outputs:
  compiled_class:
    type: File
    outputSource: compile/classfile

steps:
  untar:
    run: tar-param.cwl
    in:
      tarfile: tarball
      extractfile: name_of_file_to_extract
    out: [extracted_file]

  compile:
    run: arguments.cwl
    in:
      src: untar/extracted_file
    out: [classfile]
```



More about Workflows here

Example of a Simple Workflow

- This workflow extracts a java source file from a tar file and then compiles it.

```
$ echo "public class Hello {}" > Hello.java && tar -cvf hello.tar Hello.java
Hello.java
```

1st-workflow-job.yml

```yaml
tarball:
  class: File
  path: hello.tar
name_of_file_to_extract: Hello.java
```

```
$ cwltool 1st-workflow.cwl 1st-workflow-job.yml
INFO /opt/hostedtoolcache/Python/3.9.18/x64/bin/cwltool 3.1.20240112164112
INFO Resolved '1st-workflow.cwl' to 'file:///home/runner/work/user_guide/user_guide,
INFO [workflow ] start
INFO [workflow ] starting step untar
INFO [step untar] start
INFO [job untar] /tmp/15x_k_lh$ tar \
    --extract \
    --file \
    /tmp/1fwnbs_f/stg18bf11a8-c7fd-472d-9452-c79778c3b941/hello.tar \
    Hello.java
INFO [job untar] completed success
INFO [step untar] completed success
INFO [workflow ] starting step compile
INFO [step compile] start
INFO [job compile] /tmp/pk5dxm5z$ docker \
    run \
    -i \
    --mount=type=bind,source=/tmp/pk5dxm5z,target=/prWAaF \
    --mount=type=bind,source=/tmp/6t9joy8o,target=/tmp \
    --mount=type=bind,source=/tmp/15x_k_lh/Hello.java,target=/var/lib/cwl/stg1a9b1a
    --workdir=/prWAaF \
    --read-only=true \
    --net=none \
    --user=1001:127 \
    --rm \
    --cidfile=/tmp/su6ybu4m/20240313071624-376067.cid \
    --env=TMPDIR=/tmp \
    --env=HOME=/prWAaF \
    openjdk:9.0.1-11-slim \
    javac \
    -d \
    /prWAaF \
    /var/lib/cwl/stg1a9b1a60-da6b-4c46-b3d3-a740abe8c9ff/Hello.java
INFO [job compile] completed success
INFO [step compile] completed success
INFO [workflow ] completed success
{
    "compiled_class": {
```

More about Workflows [here]

Lets follow our CWL tutorial

- Google Colab Notebook
- You can also open this link to follow the ‘official’ CWL tutorial.

2. Hands-On Exercises

COMMON
WORKFLOW
LANGUAGE

Exercise 1: Building CWL Tools in Google Colab

Go to your copy of Google Colab Notebook (at the end) and create CWL `tools' for wrapping :

GREP :

```
$grep "hello" helloworld.txt
hello world
```

WC :

```
user: $ wc -l somefile.txt
# prints number of lines in somefile.txt to stdout
```

TAR :

```
user $ tar -xvzf some.tar.xz
```

Important: You just can do one of the three!

Exercise 1: Building CWL Tools in Google Colab

GREP Tool: Follow the instructions/code from here
1. Define inputs for the search term and the file to search in.
2. Specify the base command and arguments to reflect GREP's syntax

WC Tool: Follow the instructions/code detailed here
1. Set up parameters to count occurrences.
2. Configure the output to capture the count in a designated file

TAR Tool: Follow the instructions/code detailed here
1. Detail the input as the compressed file
2. Configure the output as the uncompressed content.

1. Note: You can use the input files that you have in 'exercise/cl-tools/' folder
    1. E.g. 'exercise/cl-tools/grep/

Important: You just can do one of the three!

Exercise 2: Creating a CWL workflow in Google Colab

Important: Only If you have extra time ☺ and have created the tree tools

- Design a workflow that integrates the GREP, WC, and TAR tools.
  - The workflow should sequentially uncompressed a file;
  - Search for a string ;
  - And count the occurrences, with the result saved in count.txt.
- Follow the instructions/code detailed here

Workflow Steps:

1. **Step 1:** Start with the TAR tool to uncompress the input file.
2. **Step 2:** Use the GREP tool to search for the desired string in the uncompressed data.
3. **Step 3:** Apply the WC tool on the output of GREP to count the occurrences and output to count.txt.

Workflow Execution:

1. Ensure that each tool's output is correctly piped as the input to the subsequent tool.
2. Set up the final output file count.txt to store the count from the WC tool.

Note: Remember to test each CWL tool individually before integrating them into the workflow.

Acknowledgements

- *Common Workflow Language User Guide* , CWL team
- *Introduction to Workflow Languages*, Melbourne Bioinformatics
- *Common Workflow Language (CWL) Tutorial*
- *Methods Included: Standardizing Computational Reuse and Portability with the Common Workflow Language*