CWL Advanced Training

A tour of features

Topics

Command Line Tool features

1. Simplest script wrapping
2. Input binding arrays
3. Redirecting standard output
4. Capture output files by wildcard
5. Secondary files
6. Initializing the working directory
7. Setting environment variables
8. Setting a time limit
9. Reading output
10. Network access

Workflow features

1. Parameter sweeps
2. Conditionals

Arvados CWL extensions

1. arv:RunInSingleContainer
2. arv:IntermediateOutput
3. cwltool:Secrets

Command Line Tool features

Simplest script wrapping

- When you want to wrap a small, single-file custom script
- Add a “File” input with a default value that is a File object
- The script will be uploaded automatically

script_wrapping/myscript.py

```
import sys
print(sys.argv[1].lower())
```

script_wrapping/main.cwl

```
cwlVersion: v1.2
class: CommandLineTool
inputs:
  val: string
  script:
    type: File
    default:
      class: File
      location: myscript.py
arguments: [python, $(inputs.script), $(inputs.val)]
outputs: []
```

https://www.commonwl.org/v1.2/CommandLineTool.html#CommandLineTool

Input binding arrays

Prefix goes in front of array:

```
--p1 banana strawberry orange
```

Prefix goes in front of each item

```
--p2 blueberry --p2 pear --p2 apple
```

Items are joined by itemSeparator:

```
--p3 blackberry,peach,nectarine
```

No space between prefix and value:

```
--p4=mango
```

inputbinding/main.cwl

```yaml
cwlVersion: v1.2
class: CommandLineTool
inputs:
  input1:
    type: string[]
    inputBinding:
      prefix: --p1
  input2:
    type:
      type: array
      items: string
      inputBinding:
        prefix: --p2
  input3:
    type: string[]
    inputBinding:
      prefix: --p3
      itemSeparator: ","
  input4:
    type: string
    inputBinding:
      prefix: --p4=
      separate: false
baseCommand: echo
outputs: []
```

```yaml
input1:
  - banana
  - strawberry
  - orange
input2:
  - blueberry
  - pear
  - apple
input3:
  - blackberry
  - peach
  - nectarine
input4: mango
```

https://www.commonwl.org/v1.2/CommandLineTool.html#CommandLineBinding

Redirect standard output

- Direct standard output to a file named in the “stdout” field
- Associated with output parameter of type “stdout”

stdout/myscript.py

```
import sys
print(sys.argv[1].lower())
```

stdout/main.cwl

```
cwlVersion: v1.2
class: CommandLineTool
inputs:
  val: string
  script:
    type: File
    default:
      class: File
      location: myscript.py
stdout: lower.txt
arguments: [python, $(inputs.script), $(inputs.val)]
outputs:
  lower:
    type: stdout
```

https://www.commonwl.org/v1.2/CommandLineTool.html#stdout

Capture output files by wildcard

- When you don't know how many output files there will be, or don't know the file name
- "glob" takes a shell wildcard expression to match files

glob/main.cwl

```
cwlVersion: v1.2
class: CommandLineTool
inputs: []
arguments: [touch, first.txt, second.txt]
outputs:
  touched:
    type: File[]
      outputBinding:
      glob: "*.txt"
```

https://www.commonwl.org/v1.2/CommandLineTool.html#CommandOutputBinding

Secondary Files

- Some file formats implicitly or explicitly depend on external files
  - For example, an index file used for seeking within a much larger data file
- CWL has a concept of “secondary files” which are files (or directories) that “follow along” with a primary file
- This means a single “File” type input parameter may actually consist of multiple files
- Secondary files are always placed in the same directory as the primary file

Requiring a secondary file on an input parameter

- Expect a bam file and a ".bam.bai" file
- The secondary file name is constructed by adding ".bai" to the end
- This will produce an error if there's no ".bam.bai" secondary file on the input

secondaryfile/input/main.cwl

```
cwlVersion: v1.2
class: CommandLineTool
inputs:
  bam:
    type: File
    secondaryFiles: .bai
arguments: [ls, $(inputs.bam.dirname)]
outputs: []
```

https://www.commonwl.org/v1.2/CommandLineTool.html#CommandInputParameter

Capturing a secondary file on an output parameter

- Expect a bam file and a ".bai" file
- The leading "^" means strip off the extension and then add ".bai" to get "chr1.bai"

secondaryfile/output/main.cwl

```
cwlVersion: v1.2
class: CommandLineTool
inputs: []
arguments: [touch, chr1.bam, chr1.bai]
outputs:
  bam:
    type: File
    secondaryFiles: ^.bai
      outputBinding:
      glob: "*.bam"
```

https://www.commonwl.org/v1.2/CommandLineTool.html#CommandOutputParameter

Secondary Files in the input object

- When constructing the input to a workflow, you may need to list out the secondary files explicitly:

```
bam:
  class: File
  location: chr1.bam
  secondaryFiles:
    - class: File
      location: chr1.bam.bai
```

Initializing the working directory

- The initial working directory (which is also the output directory) normally starts empty.
- InitialWorkDirRequirement lets you specify files that will be added to the working directory before the program runs
- Some things this can be used for include
  - Renaming input file to a known filename
  - Creating scripts or configuration files on the fly
  - Making input files writable

https://www.commonwl.org/v1.2/CommandLineTool.html#InitialWorkDirRequirement

Renaming input file

- “entryname” is the new name of the file
- “entry” is the contents of the file
- Here it will be the contents of the file in the “script” parameter

initworkdir/renaming/main.cwl

```
cwlVersion: v1.2
class: CommandLineTool
inputs:
  val: string
  script:
      type: File
      default:
      class: File
      location: myscript.py
requirements:
  InitialWorkDirRequirement:
    listing:
      - entryname: renamed.py
        entry: $(inputs.script)
arguments: [python, renamed.py, $(inputs.val)]
outputs: []
```

https://www.commonwl.org/v1.2/CommandLineTool.html#Dirent

Creating a script on the fly

- “entry” is the text of the script
- use “|” to start a multi-line indented text block
- performs parameter substitution on the text at runtime

initworkdir/literal/main.cwl

```
cwlVersion: v1.2
class: CommandLineTool
inputs:
  val: string
requirements:
  InitialWorkDirRequirement:
    listing:
      - entryname: myscript.py
        entry: |
          print("$(inputs.val)")
arguments: [python, myscript.py]
outputs: []
```

https://www.commonwl.org/v1.2/CommandLineTool.html#Dirent

Setting environment variables

- “EnvVarRequirement”
- Under “envDef” provide each variable name and value, can use parameters or expressions

envvar/main.cwl

```
cwlVersion: v1.2
class: CommandLineTool
inputs:
  val: string
requirements:
  EnvVarRequirement:
    envDef:
      VALUE: $(inputs.val)
arguments: [env]
outputs: []
```

https://www.commonwl.org/v1.2/CommandLineTool.html#EnvVarRequirement

Setting a time limit & ShellCommandRequirement

- “ToolTimeLimit”
- If the tool runs for longer than the time limit (specified in seconds), it may be terminated
- Useful to prevent jobs that are known to behave badly from running for 24 hours before anyone notices
- Use ShellCommandRequirement to quote anything not “shellQuote: false”

timelimit/main.cwl

```
cwlVersion: v1.2
class: CommandLineTool
requirements:
  ToolTimeLimit:
    timelimit: 3
  ShellCommandRequirement: {}
inputs:
  val: string
arguments: ["sleep", "8",
            {shellQuote: false, valueFrom: "&&"},
            "echo", $(inputs.val)]
outputs: []
```

https://www.commonwl.org/v1.2/CommandLineTool.html#ToolTimeLimit

https://www.commonwl.org/v1.2/CommandLineTool.html#ShellCommandRequirement

Reading & parsing output

- Want to return an integer, but the program output written to a file
- With “loadContents: true” the text of the file will be read into “contents” field of the file object
- Use “outputEval” with expression to parse the file text and return an integer

outputeval/main.cwl

```
cwlVersion: v1.2
class: CommandLineTool
requirements:
  InlineJavascriptRequirement: {}
inputs: []
arguments: [echo, "5"]
stdout: result.txt
outputs:
  result:
    type: int
    outputBinding:
      glob: result.txt
      loadContents: true
      outputEval: $(parseInt(self[0].contents))
```

https://www.commonwl.org/v1.2/CommandLineTool.html#CommandOutputBinding

Network access

- Tool execution are blocked from accessing network resources by default
- This is for security, and because depending on remote resources is bad for reproducibility
- Use NetworkAccess to enable it

networkaccess/main.cwl

```
cwlVersion: v1.2
class: CommandLineTool
inputs:
  url: string
requirements:
  NetworkAccess:
    networkAccess: true
arguments: [curl, -O, $(inputs.url)]
outputs:
  downloaded:
      type: File
      outputBinding:
      glob: "*"
```

https://www.commonwl.org/v1.2/CommandLineTool.html#NetworkAccess

Workflow features

Parameter sweeps

- ExpressionTool lets you write workflow steps in Javascript

- Generate and return the list of parameters

parametersweep/generate.cwl

```
cwlVersion: v1.2
class: ExpressionTool
requirements:
  InlineJavascriptRequirement: {}
inputs:
  min: int
  max: int
  step: int
outputs:
  parameters: int[]
expression: |-
  ${
  var p = [];
  for (var i = inputs.min; i <= inputs.max; i += inputs.step) {
      p.push(i);
  }
  return {parameters: p};
  }
```

https://www.commonwl.org/v1.2/Workflow.html#ExpressionTool

Parameter sweeps

- Generate list of parameters (previous slide)
- Scatter over parameters to each each one

parametersweep/main.cwl

```yaml
cwlVersion: v1.2
class: Workflow
requirements:
  ScatterFeatureRequirement: {}
inputs: []
steps:
  generateParameters:
      run: generate.cwl
      in:
      min: {default: 5}
      max: {default: 30}
      step: {default: 5}
      out: [parameters]
  checkParameters:
      run: check.cwl
      in:
      parm: generateParameters/parameters
      expect: {default: 20}
      scatter: [parm]
      out: [result]
outputs:
  parameters:
      type: int[]
      outputSource: generateParameters/parameters
  results:
      type: int[]
      outputSource: checkParameters/result
```

https://www.commonwl.org/v1.2/Workflow.html#Scatter/gather

Conditionals

- Use “when” to conditionally execute a workflow step
- A skipped step will have all its output parameters set to “null”
- Use “pickValue” to reduce two or more source values to a single non-null value

conditional/main.cwl

```yaml
class: Workflow
cwlVersion: v1.2
inputs:
  val: int
steps:
  step1:
    in:
      val: val
      msg: {default: "Executed step1 because inputs.val < 2"}
    run: echo.cwl
    when: $(inputs.val < 2)
    out: [out]
  step2:
    in:
      val: val
      msg: {default: "Executed step2 because inputs.val >= 2"}
    run: echo.cwl
    when: $(inputs.val >= 2)
    out: [out]
outputs:
  out1:
    type: File
    outputSource:
      - step1/out
      - step2/out
    pickValue: first_non_null
requirements:
  InlineJavascriptRequirement: {}
  MultipleInputFeatureRequirement: {}
```

https://www.commonwl.org/v1.2/Workflow.html#Conditional_execution_(Optional)

Arvados CWL Extensions

arv:RunInSingleContainer

- On Arvados, each workflow step is normally scheduled as a separate job
- If you have a lot of very short jobs (seconds to a single digit minutes), overhead and queuing time for launching each job can dominate runtime
- Use RunInSingleContainer to submit a subworkflow as a single job
- Only requirements are that the same Docker container can be used for all tools, and must have “cwltool” installed.
- Also useful you have several steps that pass a very large file between them, and you don’t want to store intermediate results in Keep

https://doc.arvados.org/v2.1/user/cwl/cwl-extensions.html

arv:RunInSingleContainer

- Arvados extensions namespace
- Hint that this step should be run in a single container

arvados/RunInSingleContainer/main.cwl

```
class: Workflow
cwlVersion: v1.2
$namespaces:
  arv: "http://arvados.org/cwl#"
inputs: []
requirements:
  SubworkflowFeatureRequirement: {}
steps:
  step1:
    hints:
      arv:RunInSingleContainer: {}
    in:
      msg: {default: "Message for the subworkflow."}
    run: subworkflow.cwl
    out: [out]
  step2:
    in:
      file: step1/out
    run: rev.cwl
    out: [out]
outputs:
  step1out:
    type: File
    outputSource: step1/out
  step2out:
    type: File
    outputSource: step2/out
```

arv:IntermediateOutput

- Workflows produce lots of intermediate results that you may not want to keep around forever
- Set output “time to live”, intermediate collections will be automatically trashed that many seconds after being created
- Make sure the TTL is longer than the longest expected runtime of the workflow

arvados/IntermediateOutput/main.cwl

```
class: Workflow
cwlVersion: v1.2
$namespaces:
  arv: "http://arvados.org/cwl#"
inputs:
  msg: string
hints:
  arv:IntermediateOutput:
    outputTTL: 3600
steps:
  step1:
    in:
      msg: msg
    run: echo.cwl
    out: [out]
  step2:
    in:
      file: step1/out
    run: rev.cwl
    out: [out]
outputs:
  out:
    type: File
    outputSource: step2/out
```

https://doc.arvados.org/v2.1/user/cwl/cwl-extensions.html

cwltool:Secrets

- A workflow step that accesses a network resource is likely to need credentials to access that resource
- Obvious solutions is to provide credentials as a workflow input, but don't want them to leak
- Arvados supports special handling of secrets:
  - Secret values are not returned in API calls
  - Secret values are obscured in logging
  - Secret values are wiped from the database as soon as the job ends

https://doc.arvados.org/v2.1/user/cwl/cwl-extensions.html

cwltool:Secrets

- cwltool extensions namespace
- Specify which parameters are secret
- Can write a config file on the fly containing the secret

arvados/Secrets/main.cwl

```
cwlVersion: v1.0
class: CommandLineTool
$namespaces:
  cwltool: http://commonwl.org/cwltool#
hints:
  "cwltool:Secrets":
    secrets: [pw]
requirements:
  InitialWorkDirRequirement:
    listing:
      - entryname: example.conf
        entry: |
          username: user
          password: $(inputs.pw)
inputs:
  pw: string
outputs:
  out: stdout
stdout: hashed_example.txt
arguments: [md5sum, example.conf]
```

Thanks!