WDL Workflows Guide

February, 2024

Table of Contents

# About this Course

This WDL workflow development guide shows a bioinformatics workflow developer how to strategically develop and scale up a WDL workflow that is iterative, reproducible, and efficient in terms of time and resource used. This guide is flexible regardless of where the data is, what computing resources are being used, and what software is being used.

## 0.1 Target Audience

The course is intended for first time developers of the [WDL workflow language](https://github.com/openwdl/wdl), who wants to iteratively develop a WDL bioinformatics workflow. The audience should be able to run a WDL workflow on a computing engine of their choice, such as Cromwell, miniWDL, or a cloud computing environment such as Terra, AnVIL, or Dockstore.

## 0.2 Curriculum

The course covers…

An excellent WDL companion resource that is not platform dependent is [OpenWDL Docs](https://docs.openwdl.org/en/stable/). OpenWDL Docs focuses on the basic grammar of WDL as well as providing excellent cookbook recipes of common WDL workflow structures. In this guide we will reference these basic grammar structures and common workflow cookbook recipes.

# 1 Introduction to WDL

Welcome!

* Review of basic WDL syntax
* How to use input JSONs
* (optional) Installing Docker and miniwdl
* How to run simple workflows locally

## 1.1 Review of basic WDL syntax

A WDL workflow consists of at least one task.

version 1.0  
  
task do\_something {  
 command <<<  
 exit 0  
 >>>  
}  
  
workflow my\_workflow {  
 call do\_something  
}

A workflow, and the tasks it calls, generally has inputs.

version 1.0  
  
task do\_something {  
 input {  
 File fastq  
 }  
 command <<<  
 exit 0  
 >>>  
}  
  
workflow my\_workflow {  
 input {  
 File fq  
 }  
 call do\_something {  
 input:  
 fastq = fq  
 }  
}

To access a task-level input variable in a task’s command section, it is usually referenced using ~{this} notation. To access a workflow-level variable in a workflow, it is referenced just by its name without any special notation. To access a workflow-level variable in a task, it must be passed into the task as an input.

version 1.0  
  
task do\_something {  
 input {  
 File fastq  
 String basename\_of\_fq  
 }  
 command <<<  
 echo "First ten lines of ~{basename\_of\_fq}: "  
 head ~{fastq}  
 >>>  
}  
  
workflow my\_workflow {  
 input {  
 File fq  
 }  
   
 String basename\_of\_fq = basename(fq)  
   
 call do\_something {  
 input:  
 fastq = fq,  
 basename\_of\_fq = basename\_of\_fq  
 }  
}

Tasks and workflows also typically have outputs. The task-level outputs can be accessed by the workflow or any subsequent tasks. The workflow-level outputs represent the final output of the overall workflow.

version 1.0  
  
task do\_something {  
 input {  
 File fastq  
 String basename\_of\_fq  
 }  
 command <<<  
 echo "First ten lines of ~{basename\_of\_fq}: " >> output.txt  
 head ~{fastq} >> output.txt  
 >>>  
 output {  
 File first\_ten\_lines = "output.txt"  
 }  
}  
  
workflow my\_workflow {  
 input {  
 File fq  
 }  
   
 String basename\_of\_fq = basename(fq)  
   
 call do\_something {  
 input:  
 fastq = fq,  
 basename\_of\_fq = basename\_of\_fq  
 }  
   
 output {  
 File ten\_lines = do\_something.first\_ten\_lines  
 }  
}

## 1.2 Using JSONs to control workflow inputs

Running a WDL workflow generally requires two files: A .wdl file, which contains the actual workflow, and a .json file, which provides the inputs for the workflow.

In the example we showed earlier, the workflow takes in a file referred to by the variable fq. This needs to be provided by the user. Typically, this is done with a JSON file. Here’s what a JSON file for this workflow might look like:

{  
 "my\_workflow.fq": "./data/example.fq"  
}

JSON files consist of key-value pairs. In this case, the key is "my\_workflow.fq" and the value is the path "./data/example.fq". The first part of the key is the name of the workflow as written in the WDL file, in this case my\_workflow. The variable being represented is referred to its name, in this case, fq. So, the file located at the path ./data/example.fq is being input as a variable called fq into the workflow named my\_workflow.

In WDL, like most programming languages, variables have a specific type. Files aren’t the only type of variable you can refer to when using JSONs. Here’s an example JSON for every common WDL variable type.

{  
 "some\_workflow.file": "./data/example.fq",  
 "some\_workflow.string": "Hello world!",  
 "some\_workflow.integer": 1965,  
 "some\_workflow.float": 3.1415,  
 "some\_workflow.boolean": true,  
 "some\_workflow.array\_of\_files": ["./data/example01.fq", "./data/example02.fq"]  
}

Resources:

For more information on types in WDL, we recommend [OpenWDL’s documentation on variable types](https://docs.openwdl.org/en/stable/WDL/variable_types/).

If you are having difficulty writing valid JSON files, considering using <https://jsonlint.com/> to check your JSON for any errors.

## 1.3 How to run simple workflows locally

Not every WDL workflow will run well on a laptop, but it can be helpful to have a basic setup for testing and catching simple syntax errors. Let’s quickly set up a WDL executor to run our WDLs.

The two most popular WDL executors are miniwdl and Cromwell. Both can run WDLs on a local machine, HPC, or cloud computing backend. In this course, we will be using miniwdl, but everything in this course will also be compatible with Cromwell unless explicitly stated otherwise. Additionally, almost all WDLs use Docker images, so you will also need to install Docker or a Docker-like alternative.

**Installing Docker and miniwdl is not required to use this course.** We don’t want anybody to get stuck here! If you already have a method for submitting workflows, such as Terra, feel free to use that for this course instead of running workflows directly on your local machine. If you don’t have any way of running workflows at the moment, that’s also okay – we have provided plenty of examples for following along.

### 1.3.1 Installing Docker

**Note: Although Docker’s own docs recommend installing Docker Desktop for Linux,** [**it has been reported**](https://github.com/dockstore/dockstore/issues/5135) **that some WDL executors work better on Linux when installing only Docker Engine (aka Docker CE).** To install Docker on your machine, follow the instructions specific to your operating system [on Docker’s website](https://docs.docker.com/get-docker/). To specifically install only Docker Engine, [use these instructions instead](https://docs.docker.com/engine/install/).

If you are unable to install Docker on your machine, Dockstore (not affiliated with Docker) [provides some experimental alternatives](https://docs.dockstore.org/en/stable/advanced-topics/docker-alternatives.html). Dockstore also provides [a comprehensive introduction to Docker itself](https://docs.dockstore.org/en/stable/getting-started/getting-started-with-docker.html?highlight=engine#where-can-i-run-docker), including how to write a Dockerfile. Much of that information is outside the scope of this WDL-focused course, but it may be helpful for those looking to eventually create their own Docker images.

### 1.3.2 Installing miniwdl

miniwdl is based on Python. If you do not already have Python 3.6 or higher installed, [you can install Python from here](https://www.python.org/downloads/).

Once Python is installed on your system, you can run pip3 install miniwdl from the command line to install miniwdl. For those who prefer to use conda, use conda install -c conda-forge miniwdl instead. Once miniwdl is installed, you can verify it works properly by running miniwdl run\_self\_test. This will run a built-in hello world workflow.

For more information, see [miniwdl’s GitHub repository](https://github.com/chanzuckerberg/miniwdl).

### 1.3.3 Launching a workflow locally with miniwdl

The generic method for running a WDL with miniwdl is the following:

miniwdl run [path\_to\_wdl\_file] -i [path\_to\_inputs\_json]

If you have successfully installed miniwdl, create the following WDL file and name it greetings.wdl:

version 1.0  
  
task greet {  
 input {  
 String user  
 }  
 command <<<  
 echo "Hello ~{user}!" > greets.txt  
 >>>  
 output {  
 String greeting = read\_string("greets.txt")  
 }  
}  
  
workflow my\_workflow {  
 input {  
 String username  
 }  
 call greet {  
 input:  
 user = username  
 }  
}

Next, use this JSON file (or create one of your own) to provide the string that the workflow expects, and call the JSON file greetings.json:

{  
 "my\_workflow.username": "Ash"  
}

On the command line, run the following:

miniwdl run greetings.wdl -i greetings.json

Once the task completes, you should see something like this in your command line:

[timestamp] wdl.w:my\_workflow finish :: job: "call-greet"  
[timestamp] wdl.w:my\_workflow done  
{  
 "dir": "[working directory]/[timestamp]\_my\_workflow",  
 "outputs": {  
 "my\_workflow.greet.greeting": "Hello Ash!"  
 }  
}

Where [timestamp] is the date and time that you are running the workflow, and [working directory] is the working directory that you are running the workflow from. For example:

2023-12-27 13:54:12.209 wdl.w:my\_workflow finish :: job: "call-greet"  
2023-12-27 13:54:12.210 wdl.w:my\_workflow done  
{  
 "dir": "/Users/ash/github/WDL\_Workflows\_Guide/resources/20231227\_135400\_my\_workflow",  
 "outputs": {  
 "my\_workflow.greet.greeting": "Hello Ash!"  
 }  
}

### 1.3.4 Troubleshooting

#### 1.3.4.1 DockerException

If you are seeing a verbose error message that begins with text like this:

2023-12-27 13:43:37.525 wdl.w:my\_workflow.t:call-greet task greet (greetings.wdl Ln 3 Col 1) failed :: dir: "/Users/sammy/github/WDL\_Workflows\_Guide/resources/20231227\_134337\_my\_workflow/call-greet", error: "DockerException", message: "Error while fetching server API version: ('Connection aborted.', FileNotFoundError(2, 'No such file or directory'))", traceback: ["Traceback (most recent call last):", " File \"/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/urllib3/connectionpool.py\", line 790, in urlopen", " response = self.\_make\_request(", " ^^^^^^^^^^^^^^^^^^^", " File \"/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/urllib3/connectionpool.py\",

This is likely caused by miniwdl being unable to connect to Docker Daemon, the underlying technology that runs Docker images. This is necessary with miniwdl even though our example WDL does not specify a Docker image. Make sure you have Docker installed correctly, and make sure Docker is actively running on your machine. If you installed Docker Desktop, simply opening the Docker Desktop app should start Docker Engine. If you installed Docker without Docker Desktop, running dockerd in your command-line should start it. Be aware that starting the Docker Daemon may take a few minutes.

#### 1.3.4.2 Missing required inputs

If you forget to add -i greetings.json to your call, you will see something like this:

my\_workflow (greetings.wdl)  
---------------------------  
  
required inputs:  
 String username  
  
outputs:  
 String greet.greeting  
  
missing required inputs for my\_workflow: username

You may also see this error if you remember to include a JSON file, but it is missing a required input.

#### 1.3.4.3 Check JSON input

If you see an error message like this:

check JSON input; unknown input/output: greetings.username

Double-check your input JSON. The first part of your JSON’s keys refer to the name of the workflow in the WDL file, not the filename of the WDL itself. Even though our WDL is saved as greetings.wdl, within that file, the workflow is named my\_workflow. This means that the input JSON must say "my\_workflow.username", not "greetings.username".

Other common issues with JSON files are mistyping input variables (such as "my\_workflow.ussername") or forgetting to enclose strings in quotation marks. When in doubt, try using <https://jsonlint.com/> to check your input JSON, and double-check the name of your input variables.

# 2 Defining a workflow plan: Introduce somatic mutation calling workflow

## 2.1 The workflow

## 2.2 The test data

# 3 The first task

after making a public gist…

## 3.1 Anatomy of a task

task BwaMem {  
 input {  
 File input\_fastq  
 String base\_file\_name  
 File ref\_fasta  
 File ref\_fasta\_index  
 File ref\_dict  
 File? ref\_alt  
 File ref\_amb  
 File ref\_ann  
 File ref\_bwt  
 File ref\_pac  
 File ref\_sa  
 String taskDocker  
 }  
 command <<<  
 set -eo pipefail  
  
 bwa mem \  
 -p -v 3 -t 16 -M \  
 ~{ref\_fasta} ~{input\_fastq} > ~{base\_file\_name}.sam   
 samtools view -1bS -@ 15 -o ~{base\_file\_name}.aligned.bam ~{base\_file\_name}.sam  
 >>>  
 output {  
 File analysisReadyBam = "~{base\_file\_name}.aligned.bam"  
   
 }  
 runtime {  
 memory: "48 GB"  
 cpu: 16  
 docker: taskDocker  
 walltime: "2:00:00"  
 }  
}

# About the Authors

These credits are based on our [course contributors table guidelines](https://www.ottrproject.org/more_features.html#giving-credits-to-contributors).

| Credits | Names |
| --- | --- |
| **Pedagogy** |  |
| Lead Content Instructor(s) | [FirstName LastName](link%20to%20personal%20website) |
| Lecturer(s) (include chapter name/link in parentheses if only for specific chapters) - make new line if more than one chapter involved | Delivered the course in some way - video or audio |
| Content Author(s) (include chapter name/link in parentheses if only for specific chapters) - make new line if more than one chapter involved | If any other authors besides lead instructor |
| Content Contributor(s) (include section name/link in parentheses) - make new line if more than one section involved | Wrote less than a chapter |
| Content Editor(s)/Reviewer(s) | Checked your content |
| Content Director(s) | Helped guide the content direction |
| Content Consultants (include chapter name/link in parentheses or word “General”) - make new line if more than one chapter involved | Gave high level advice on content |
| Acknowledgments | Gave small assistance to content but not to the level of consulting |
| **Production** |  |
| Content Publisher(s) | Helped with publishing platform |
| Content Publishing Reviewer(s) | Reviewed overall content and aesthetics on publishing platform |
| **Technical** |  |
| Course Publishing Engineer(s) | Helped with the code for the technical aspects related to the specific course generation |
| Template Publishing Engineers | [Candace Savonen](https://www.cansavvy.com/), [Carrie Wright](https://carriewright11.github.io/), [Ava Hoffman](https://www.avahoffman.com/) |
| Publishing Maintenance Engineer | [Candace Savonen](https://www.cansavvy.com/) |
| Technical Publishing Stylists | [Carrie Wright](https://carriewright11.github.io/), [Ava Hoffman](https://www.avahoffman.com/), [Candace Savonen](https://www.cansavvy.com/) |
| Package Developers ([ottrpal](https://github.com/jhudsl/ottrpal)) [Candace Savonen](https://www.cansavvy.com/), [John Muschelli](https://johnmuschelli.com/), [Carrie Wright](https://carriewright11.github.io/) |  |
| **Art and Design** |  |
| Illustrator(s) | Created graphics for the course |
| Figure Artist(s) | Created figures/plots for course |
| Videographer(s) | Filmed videos |
| Videography Editor(s) | Edited film |
| Audiographer(s) | Recorded audio |
| Audiography Editor(s) | Edited audio recordings |
| **Funding** |  |
| Funder(s) | Institution/individual who funded course including grant number |
| Funding Staff | Staff members who help with funding |

## ─ Session info ───────────────────────────────────────────────────────────────  
## setting value   
## version R version 4.0.2 (2020-06-22)  
## os Ubuntu 20.04.5 LTS   
## system x86\_64, linux-gnu   
## ui X11   
## language (EN)   
## collate en\_US.UTF-8   
## ctype en\_US.UTF-8   
## tz Etc/UTC   
## date 2024-02-21   
##   
## ─ Packages ───────────────────────────────────────────────────────────────────  
## package \* version date lib source   
## assertthat 0.2.1 2019-03-21 [1] RSPM (R 4.0.5)   
## bookdown 0.24 2023-03-28 [1] Github (rstudio/bookdown@88bc4ea)   
## cachem 1.0.7 2023-02-24 [1] CRAN (R 4.0.2)   
## callr 3.5.0 2020-10-08 [1] RSPM (R 4.0.2)   
## cli 3.6.1 2023-03-23 [1] CRAN (R 4.0.2)   
## crayon 1.3.4 2017-09-16 [1] RSPM (R 4.0.0)   
## desc 1.2.0 2018-05-01 [1] RSPM (R 4.0.3)   
## devtools 2.3.2 2020-09-18 [1] RSPM (R 4.0.3)   
## digest 0.6.25 2020-02-23 [1] RSPM (R 4.0.0)   
## ellipsis 0.3.1 2020-05-15 [1] RSPM (R 4.0.3)   
## evaluate 0.20 2023-01-17 [1] CRAN (R 4.0.2)   
## fastmap 1.1.1 2023-02-24 [1] CRAN (R 4.0.2)   
## fs 1.5.0 2020-07-31 [1] RSPM (R 4.0.3)   
## glue 1.4.2 2020-08-27 [1] RSPM (R 4.0.5)   
## htmltools 0.5.5 2023-03-23 [1] CRAN (R 4.0.2)   
## knitr 1.33 2023-03-28 [1] Github (yihui/knitr@a1052d1)   
## magrittr 2.0.3 2022-03-30 [1] CRAN (R 4.0.2)   
## memoise 2.0.1 2021-11-26 [1] CRAN (R 4.0.2)   
## pkgbuild 1.1.0 2020-07-13 [1] RSPM (R 4.0.2)   
## pkgload 1.1.0 2020-05-29 [1] RSPM (R 4.0.3)   
## prettyunits 1.1.1 2020-01-24 [1] RSPM (R 4.0.3)   
## processx 3.4.4 2020-09-03 [1] RSPM (R 4.0.2)   
## ps 1.4.0 2020-10-07 [1] RSPM (R 4.0.2)   
## R6 2.4.1 2019-11-12 [1] RSPM (R 4.0.0)   
## remotes 2.2.0 2020-07-21 [1] RSPM (R 4.0.3)   
## rlang 1.1.0 2023-03-14 [1] CRAN (R 4.0.2)   
## rmarkdown 2.10 2023-03-28 [1] Github (rstudio/rmarkdown@02d3c25)  
## rprojroot 2.0.3 2022-04-02 [1] CRAN (R 4.0.2)   
## sessioninfo 1.1.1 2018-11-05 [1] RSPM (R 4.0.3)   
## stringi 1.5.3 2020-09-09 [1] RSPM (R 4.0.3)   
## stringr 1.4.0 2019-02-10 [1] RSPM (R 4.0.3)   
## testthat 3.0.1 2023-03-28 [1] Github (R-lib/testthat@e99155a)   
## usethis 1.6.3 2020-09-17 [1] RSPM (R 4.0.2)   
## withr 2.3.0 2020-09-22 [1] RSPM (R 4.0.2)   
## xfun 0.26 2023-03-28 [1] Github (yihui/xfun@74c2a66)   
## yaml 2.2.1 2020-02-01 [1] RSPM (R 4.0.3)   
##   
## [1] /usr/local/lib/R/site-library  
## [2] /usr/local/lib/R/library

# 4 References