How to bake a workflow

Goal

The goal of this guide is to take you through the process of structuring and populating a very basic workflow. Work your way through this recipe step by step and end up with a workflow which takes raw read data as input, creates quality assurance reports alongside trimmed reads, as well as generates de-novo assemblies from isolate read data.

The recipe assumes some prior knowledge of executing bash commands, making very simple scripts, positional arguments as script input, and finally utilizing virtual Conda environments to install and execute bioinformatic tools. In the end though, you will get a hang of how you can start your very own basic bioinformatic workflow.

The first steps

The natural first step of building workflows is to execute and test bioinformatic commands. Successful commands can be stored in simple bash scripts to build your desired workflow step by step.

Part 1 - Creating and testing scripts

Scripts are basically simple text files which when executed can be interpreted as simple bash commands.

Exercise 1

- Generate a simple script that will generate a FastQC report for any sample, by copy pasting the following into a text editor.

```
#!/bin/bash

# Input
read1=~/BTG/SequenceData/Ec005...
read2=~/BTG/SequenceData/Ec005...

# Output
outdir="~/BTG/test_workflow/FastQC"

mkdir -p $outdir
micromamba run -n BTG_QC fastqc $read1 $read2 -o $outdir
```

- Replace the paths in the `read1` and `read2` variables, so they point to the read mates of an actual sample. Note the output will be written to a new folder (called `test_workflow/`)
- Save the script as `FastQC.sh` in the `~/BTG/workflow/modules/` folder
- Then, execute the script from bash.

```
cd ~/BTG/workflow
bash ~/BTG/workflow/modules/FastQC.sh
```

Part 2 - Making scripts dynamic

Currently, we have created a script that is very good at making QC reports for ONE specific sample. This is not very useful or practical unless you plan on renaming all future samples to the same position and name. One way to circumvent this is to replace inputs and outputs with positional arguments. Lets generate a script that takes in input and output from positional arguments for performing read trimming with fastp.

Exercise 2

- Copy paste the following into a empty text document called `~/BTG/workflow/modules/fastp.sh`

```
#!/bin/bash

# Input
read1=$1
read2=$2
sample=$3

# Params (Decide by yourself)
avg_qual=20
trim_front=9

# Output
outdir=$4/fastp

mkdir -p $outdir
micromamba run -n BTG_QC fastp --in1 $read1 --out1 $outdir/"$sample"_trim_R1.fastq.gz --in2 $read2 --out
```

- Execute the bash and make sure to provide the required positional arguments to ensure that the execution works. Use below as example
  - `bash ~/BTG/workflow/modules/fastp.sh /path/to/read1.fastq.gz /path/to/read2.fastq.gz sample_name ~/BTG/test_workflow/fastp/`

  ▼ Solution

  ```
  bash ~/BTG/workflow/modules/fastp.sh ~/BTG/SequenceData/Ec005...R1.fastq.gz ~/BTG/SequenceData
  ```

Exercise 3

- Having learned how to use positional arguments, revisit FastQC script and convert the variables to take positional arguments instead of hardcoded file paths (`$1`, `$2`, and `$3`)
- Do a quick test run of the FastQC script.

Part 3 - Mastering environments

Currently we have two scripts for doing quality assurance, however, these steps currently only work on your local laptop (our course computers). So, in order to ensure that everything is easily deployable on other machines, we will create installation instructions for these steps and store these in simple text (`.yaml`) files.

Exercise 4

- Make a `envs/` folder inside `~/BTG/workflow/`.
- Create an empty text file called `QC.yaml` inside the `envs/` folder. Henceforth all files placed there are referred as environment files.
- Fill out the `QC.yaml` file with the following

```
name:
 - QC
channels:
 - conda-forge
 - bioconda
dependencies:
 - fastqc
 - fastp
 - multiqc
 - quast
```

Now you are ready to install and test whether the environment works as intended. We will create an envrionemnt named the same as the environment file (*QC*), yet **FIRST** we have to make sure that there is no enviornment with that name.

- Use `micromamba env list`

- If there already is an environment named *QC*, use `micromamba env remove -n QC` to remove it.

- Use `micromamba create -f /BTG/workflow/envs/QC.yaml` to create the environment.
  - You would have to run these installation commands on your own machine to deploy the workflow.

Exercise 5

Again, its time to update the existing scripts to use the current environment, rather than the previous `BTG_QC` enviornment

- Revisit both the `fastqc.sh` and `fastp.sh` modules and replace the `-n BTG_QC` with `-n QC` inside the `micromamba run` commands.

Part 4 - Generating a workflow

Currently we only have two scripts, henceforth refered as Modules. In order to use these, you would have to call each of them through bash commands, yet as the workflow expands with more scripts, this task becomes increasingly daunting. Therefore, lets optimise the workflow to only require a single command to run. This is done by collecting all positional arguments alongside bash calls of each module, into a single collective bash script called `workflow.sh`. (Yes we are very creative with names!)

Exercise 6

- Make a empty text file inside the `~/BTG/workflow/` folder called `workflow.sh`.
- Insert the following into the file (Please note for simplicity, we skip the parameters for now)

```
#!/bin/bash

# Input
read1=$1
read2=$2
sample=$3

# Output
outdir=$4
```

Now we are set up to implement the modules by calling them inside the script.

- In the `workflow.sh` script add a line which calls `bash modules/FastQC.sh $read1 $read2 $outdir` (Note the **relative** path here!)
- Add another line with `fastp` - !Note fastp takes the sample name as the third positional (`$3`) argument

After this exercise, your setup should now look like

```
workflow/
  workflow.sh
  envs/  # Environment files which micromamba can use to install the required software
    QC.yaml
  modules/ # Bioinformatic commands stored in simple individual scripts
    FastQC.sh
    fastp.sh
```

We are not ready to test the workflow yet. Whenever WE run `micromamba` in the terminal, the bash are told that `micromamba` is actually a shortcut for a executable file.

If we were to run the workflow.sh script, WE will not tell the terminal to call `micromamba` the script does, and therefore the workflow would fail. We have to state what micromamba means.

Exercise 7

- In the terminal, execute `which micromamba`
  - Copy the full **direct** path for micromamba

        - It Should look something along the lines of `/home/gebt/.local/bin/micromamba`. This is actually micromambas **executable file**, meaning that every time you execute the command `micromamba`, you actually execute this file!
    - Open each of the scripts inside the `modules/` folder and then replace `micromamba` with the direct path to micromamba. ***MAKE SURE*** to replace `/home/gebt/` with `~/`, to make the workflow deployable across all other Unix machines.
- Execute the workflow again with empty parameters

Part 5 - Expanding your workflow

NOW, we actually have the concepts to generate and expand workflows, and make them semi-selfconatining. This would be a great time to practise expanding the functionality of the workflow. Therefore, lets make add a module for generating de-novo assemblies from the trimmed read data.

Exercise 8

- Create an envrionment file (in the `envs/` folder) called `Assembly.yaml`.
    - Name the environment `Assembly`.
    - Add `conda-forge` as channel, also both assemblers are available on the `bioconda` channel
    - List `spades` under dependencies

> 💡 Remember if you are stuck, look to Part 3 for inspiration on how to build the environment file!

▼ Solution

`Assembly.yaml`

```
name:
 - Assembly
channels:
 - conda-forge
 - bioconda
depedencies:
 - spades
```

- Create the environment and inspect the help page of SPAdes

    ▼ Solution

    ```
    micromamba create -f ~/BTG/workflow/envs/Assembly.yaml
    micromamba run -n Assembly spades.py --help
    ```

- Create a new module under the `modules/` folder called `SPAdes.sh`
- Add the following to the `SPAdes.sh` file

```
#!/bin/bash

# Input
read1=$1
read2=$2
sample=$3

# Output
outdir=$4/$sample/SPAdes

mkdir -p $outdir
```

- Copy paste the executable path to `micromamba` and attach `run -n Assembly spades.py`

- Add `$read1` and `$read2` as input for the `-1` and `-2` options for spades.
- Add `$outdir` as input for the `-o` option.
- Make sure to add the `--isolate` to determine the assembly mode.

- Execute the module from the terminal to ensure that it works
- Add a line at the end of the `workflow.sh` where the asembly module are called with `bash` and attach the relevant variables (remember to use the trimmed reads from the `fastp` module as input for the assembly.

Part 6 - Making the workflow more user friendly

Currently, each module takes in unnamed positional arguments for determining Input and Output. In order to make the workflow more usefull for others, it would be a great idea to define named arguments and add a software description.

Exercise 9

Lets start with the description

- Add the following just into the `workflow.sh` script after the `!#/bin/bash` line

```
# Define usage function
usage() {
  echo "Usage: $(basename "$0") [-r|--read1 Read mate 1 file] [-R|--read2 Read mate 2 file] [-o|--output_dir
  exit 1
}
```

This is a function which will print out details on how we wish others to use our workflow

- Just beneath the usage function, add the following code

```
# Parse options
while [[ $# -gt 0 ]]; do

  # Making named positional arguments
  case "$1" in
    -r|--read1)
      read1="$2"
      shift
      ;;
    -R|--read2)
      read2="$2"
      shift
      ;;
    -o|--output_dir)
      outdir="$2"
      shift
      ;;
    *)
      usage
      ;;
  esac
  shift
done
```

This will look through all user input and look whether there are any instances of `-r` , `--read1` , `-R`, `--read2` , `-o`, or `--outdir` defined.

Everytime any of these is identified from the user input, the very next inpuit is used to define the value corresponding to the argument.

- Finally, we have to add a section that STOPS the script if none of the above arguments are provided, add the following code:

```
# Check if required arguments are provided
if [[ -z $read1 || -z $read2 || -z $outdir ]]; then
  usage
fi
```

This looks whether the variables `$read1`, `$read2`, or `$outdir` was defined, if not the code calls the `usage` function and then **Shuts down**

- Execute the workflow to ensure that it actually works.

Part 7 workflow on entire directories

With everything up and running so far, it's time to automate the workflow further. Currently, samples needs to be specified one by one, one helpful enhancement would rather to point to an input directory, and then have the workflow screen that directory for input files. This of course means that we would have to update the description, but it would be worth the hassle.

Exercise 9

- Add the following code just before the first bash call

```
read1_files=$(find $read_dir -maxdepth 1 -name *_R1.fastq.gz | sort)

for read1 in $read1_files; do

  # Determine read mate 2
  read2=${read1%_R1.fastq.gz}_R2.fastq.gz
  sample=$(basename $read1 _R1.fastq.gz)
```

This is a for-loop, it takes in a list of input → Here the paths for read1 files and then does your bash commands on each items once. It must be ended with a `done` command.

- Track down all `-r/--read1` and `-R/--read2` arguments in the usage function and the argument handler, and replace these two arguments with a single `-r/--read_dir` argument
- Finally, at the very bottom of the script input `done` as the final line, to denote the ending of the for loop
  - Your workflow file should look like this

```
read1_files=$(find $read_dir -maxdepth 1 -type f -name *_R1.*f*q* | sort)

for read1 in $read1_files; do

  # Determine read mate 2
  read2=${read1%_R1.fastq.gz}_R2.fastq.gz
  sample=$(basename $read1 _R1.fastq.gz)

  bash modules/moduleA.sh $read1 $read2 $outdir
  bash modules/moduleB.sh $read1 $read2 $outdir
  ...

done
```

▼ Solution

```
#!/bin/bash

# Define usage function
usage() {
  echo "Usage:"
  echo " $(basename "$0") [-r|--read_dir Raw read directory] [-o|--outdir Output directory]" >&2
  exit 1
}
```

```
# Parse options
while [[ $# -gt 0 ]]; do

  # Making named positional arguments
  case "$1" in
    -r|--read_dir)
      read_dir="$2"
      shift
      ;;
    -o|--outdir)
      outdir="$2"
      shift
      ;;
    *)
      usage
      ;;
  esac
  shift
done

# Check if required arguments are provided
if [[ -z $read_dir || -z $outdir ]]; then
  usage
fi

read1_files=$(find $read_dir -maxdepth 1 -type f -name *_R1.*f*q* | sort)

for read1 in $read1_files; do

  # Determine read mate 2 and sample name
  read2=${read1%_R1.fastq.gz}_R2.fastq.gz
  sample=$(basename $read1 _R1.fastq.gz)

  # Deducted
  trimmed1=$outdir/fastp/"$sample"_trim_R1.fastq.gz
  trimmed2=$outdir/fastp/"$sample"_trim_R2.fastq.gz

  echo Running FastQC
  bash modules/FastQC.sh $read1 $read2 $outdir

  echo Running fastp
  bash modules/fastp.sh $read1 $read2 $sample $outdir

  echo Running SPAdes
  bash modules/SPAdes.sh $trimmed1 $trimmed2 $sample $outdir

done
```