Nextflow Workshop

Release 0.0.1

Barry Digby

Jan 21, 2022

INSTALLATION

1 Nextflow **3**
- 1.1 Download nextflow binary file . . . 3

2 Docker **5**
- 2.1 Install dependencies . . . 5
- 2.2 Install Docker GPG Key . . . 5
- 2.3 Install from Repository . . . 5
- 2.4 Verify Installation . . . 6
- 2.5 Dockerhub . . . 6

3 Singularity **7**
- 3.1 Install dependencies . . . 7
- 3.2 Install Go . . . 7
- 3.3 Download stable release . . . 8
- 3.4 Compile Singularity . . . 8
- 3.5 Verify Installation . . . 8

4 Github **9**
- 4.1 Generating a new SSH key . . . 9
- 4.2 Add SSH key to Github account . . . 10

5 VSCode **11**
- 5.1 Link to Github . . . 11

6 Anaconda **13**
- 6.1 Install dependencies . . . 13
- 6.2 Download Installer . . . 13
- 6.3 Install Anaconda . . . 13
- 6.4 Verify Installation . . . 14

7 Windows Users (WSL) **15**
- 7.1 Creating a Nextflow Environment . . . 16
- 7.2 Installing Singularity . . . 16

8 Conda **19**
- 8.1 Creating Environments . . . 19
- 8.2 Installing packages . . . 19
- 8.3 YAML . . . 20
- 8.4 Executable directory . . . 20

9 Docker **21**

9.1 Dockerfile . . . 21
9.2 Build image . . . 21
9.3 Check image . . . 22
9.4 Push to Dockerhub . . . 22
9.5 Advanced use . . . 22

10 Singularity **25**
10.1 Singularity pull . . . 25

11 Github Syncing **27**
11.1 Initialise Repository . . . 27
11.2 Docker Workflow . . . 28
11.3 Github Secrets . . . 29

12 First Script **31**
12.1 Scripting Language . . . 32
12.2 Channels . . . 33
12.3 Processes . . . 36
12.4 Configuration file . . . 37
12.5 Parameters . . . 37

13 Github Syncing **39**

14 Operators **41**
14.1 Map . . . 41
14.2 Join . . . 42
14.3 BaseName . . . 42
14.4 Flatten . . . 43

15 Conditionals **45**
15.1 Update .gitignore . . . 46
15.2 Update Script . . . 46

16 Assignment I **49**
16.1 Part I: QC Sequencing Reads . . . 49
16.2 Part II: Advanced Container Creation . . . 50

17 Assignment II **55**
17.1 Part 1 . . . 55
17.2 Part 2 . . . 56
17.3 Part 3 . . . 57

18 Assignment III **59**
18.1 Template script . . . 59
18.2 DAG . . . 59

19 Continuous Integration **61**
19.1 Test profile . . . 61
19.2 Sample File . . . 62
19.3 CI.yml . . . 64

20 Intro to Bash **67**
20.1 bashrc . . . 67
20.2 $PATH . . . 68
20.3 Variable Expansion . . . 69

21 Indices and tables 71

Documentation for CRT nextflow workshop. Please note this is for DSL1.

CHAPTER

ONE

NEXTFLOW

Please follow the installation steps below to install `nextflow` on your local machine:

Warning: Java 8+ must be installed on your laptop. If your version of `openjdk` is below 8, please run the code block below.

```
$ sudo apt update \
    sudo apt install default-jre \
    java -version
```

1.1 Download nextflow binary file

Note: The documentation is based on nextflow v21.04.1. Using a newer version might cause conflicts.

```
$ wget https://github.com/nextflow-io/nextflow/releases/download/v21.04.1/nextflow

$ chmod 777 ./nextflow

$ sudo mv ./nextflow /usr/local/bin/

$ nextflow -v
```

CHAPTER

TWO

DOCKER

Please follow the installation steps below to install the latest version of `Docker` on your local machine:

2.1 Install dependencies

```
$ sudo apt-get update

$ sudo apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
```

2.2 Install Docker GPG Key

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/
↪share/keyrings/docker-archive-keyring.gpg
```

2.3 Install from Repository

```
$ echo \
    "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-
↪keyring.gpg] https://download.docker.com/linux/ubuntu \
    $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

2.4 Verify Installation

```
$ docker -v
```

2.5 Dockerhub

Please set up an account on Dockerhub.

Note: I highly recommend using the same username as your Github account.

CHAPTER

THREE

SINGULARITY

Please follow the installation steps below to install the latest version of `singularity` on your local machine.

3.1 Install dependencies

```
$ sudo apt-get update && sudo apt-get install -y \
    build-essential \
    uuid-dev \
    libgpgme-dev \
    squashfs-tools \
    libseccomp-dev \
    wget \
    pkg-config \
    git \
    cryptsetup-bin
```

3.2 Install Go

```
$ export VERSION=1.13.5 OS=linux ARCH=amd64 && \
    wget https://dl.google.com/go/go$VERSION.$OS-$ARCH.tar.gz && \
    sudo tar -C /usr/local -xzvf go$VERSION.$OS-$ARCH.tar.gz && \
    rm go$VERSION.$OS-$ARCH.tar.gz
```

```
$ echo 'export GOPATH=${HOME}/go' >> ~/.bashrc && \
    echo 'export PATH=/usr/local/go/bin:${PATH}:${GOPATH}/bin' >> ~/.bashrc && \
    source ~/.bashrc
```

3.3 Download stable release

```
$ export VERSION=3.8.4 && \
    wget https://github.com/apptainer/singularity/releases/download/v${VERSION}/
↪singularity-${VERSION}.tar.gz && \
    tar -xzf singularity-${VERSION}.tar.gz && \
    cd singularity-${VERSION}
```

3.4 Compile Singularity

```
$ ./mconfig && \
    make -C ./builddir && \
    sudo make -C ./builddir install
```

3.5 Verify Installation

```
$ singularity --version
```

CHAPTER

FOUR

GITHUB

I trust you can all make a Github account without assistance.. (?)

During the practical we will be pushing commits from your local machine to Github. Please set up a SSH key linked to your Github account (so that you do not have to enter your username and password on every push):

4.1 Generating a new SSH key

```
$ ssh-keygen -t ed25519 -C "your_email@example.com"
```

Note: If it asks you to overwrite an existing key under `/home/user/.ssh/id_XXXXXXX` you might have previously set this up. Check your Github account to verfiy this.

Important: Do not enter a passphrase, leave empty and hit ENTER

Worked example:

```
$ ssh-keygen -t ed25519 -C b.digby237@gmail.com
    Generating public/private ed25519 key pair.
    Enter file in which to save the key (/home/barry/.ssh/id_ed25519):
    /home/barry/.ssh/id_ed25519 already exists.
    Overwrite (y/n)? y
    Enter passphrase (empty for no passphrase):
    Enter same passphrase again:
    Your identification has been saved in /home/barry/.ssh/id_ed25519
    Your public key has been saved in /home/barry/.ssh/id_ed25519.pub
    The key fingerprint is:
    SHA256:24A6oPDbveaBBO6qaLnIuoD4Whvc0l78EFfAGqx+vTU b.digby237@gmail.com
    The key's randomart image is:
    +--[ED25519 256]--+
    |     . ..        |
    |      o ..       |
    |  .  . o  .      |
    | . .. .. .       |
    |. o.. o.S        |
    |+=.=.+.o.+F      |
    |= O *.= .o..     |
```

(continues on next page)

(continued from previous page)

```
    |=* B +.+.        |
    |%== oo+..        |
    +----[SHA256]-----+
```

4.2 Add SSH key to Github account

```
$ cat ~/.ssh/id_ed25519.pub
```

Copy the key in `~/.ssh/id_ed25519.pub` and add it to your account:

CHAPTER

FIVE

VSCODE

Visual Studio Code or VSCode for short, is a code editor redefined and optimized for building and debugging modern web and cloud applications. VSCode has a huge ecosystem of packages that can be installed to extend functionality.

The creators of `nf-core` have put together a collection of extension packages to make it easier to get up and running when working with nf-core pipelines.

You can browse, pick and choose the ones you think look good, or can install them all in a single click.

To use, just search for `nf-core-extensionpack` in the VSCode Packages tab search bar, or visit https://marketplace.visualstudio.com/items?itemName=nf-core.nf-core-extensionpack

5.1 Link to Github

VScode should prompt you to sign in to your Github account. Enter you account details, and test it out by cloning a repository locally. Once you open the directory in VSCode, you will be able to perform every Github action `add, pull, push, commit, merge, etc..` within VSCode which is not only more efficient, but also easier to track changes.

CHAPTER

SIX

ANACONDA

Please follow the installation steps below to install the latest version of `Anaconda` on your local machine:

6.1 Install dependencies

```
sudo apt-get install libgl1-mesa-glx \
    libegl1-mesa \
    libxrandr2 \
    libxss1 \
    libxcursor1 \
    libxcomposite1 \
    libasound2 \
    libxi6 \
    libxtst6
```

6.2 Download Installer

```
$ wget https://repo.anaconda.com/archive/Anaconda3-2021.05-Linux-x86_64.sh
```

6.3 Install Anaconda

```
$ bash Anaconda3-2021.05-Linux-x86_64.sh
```

Note: Follow the instructions in the terminal regarding your $PATH, default suggestions are fine.

6.4 Verify Installation

```
$ conda -v
```

CHAPTER

SEVEN

WINDOWS USERS (WSL)

Just kidding (sort of). Follow the steps below.

7.1 Creating a Nextflow Environment

Please follow the steps outlined in the following tutorial: Setting up a Nextflow environment on Windows 10.

1. Powershell
2. WSL2
3. Windows terminal
4. Docker
5. VS Code (optional)
6. Nextflow
7. Git

Please make sure that you install `nextflow v21.04.1`. When following the tutorial, substitute the following during the `nextflow` installation:

```
$ wget https://github.com/nextflow-io/nextflow/releases/download/v21.04.1/nextflow

$ chmod 777 ./nextflow

$ mv ./nextflow /usr/local/bin/

$ nextflow -v
```

Note: If the path `/usr/local/bin` does not exist, move the executable to `/usr/bin/` instead.

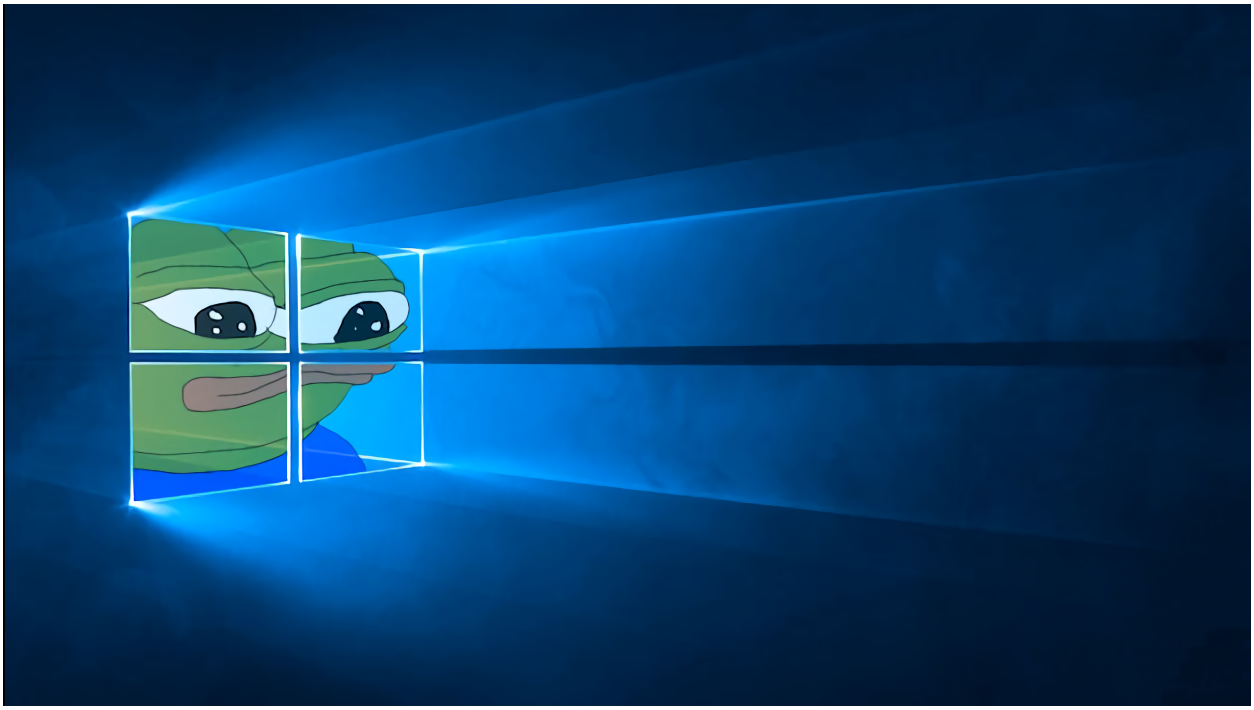
7.2 Installing Singularity

The above tutorial does not cover `singularity` installations. Please follow this tutorial to install `singularity`: **`using singularity on windows with WSL2<https://www.blopig.com/blog/2021/09/using-singularity-on-windows-with-wsl2/>`_**.

When you reach the step to download the tarball from GitHub, use the same version as ubuntu users for the workshop:

```
$ export VERSION=3.8.4 && \
wget https://github.com/apptainer/singularity/releases/download/v${VERSION}/singularity-$
↪{VERSION}.tar.gz && \
tar -xzf singularity-${VERSION}.tar.gz && \
cd singularity-${VERSION}
```

Windows users in the class - please help each other during the installation steps. I do not have a Windows partition on my laptop and cannot troubleshoot any issues you may encounter.



CHAPTER

EIGHT

CONDA

We will be using `conda` to create environments and install packages within our container.

You can achieve the same results by manually creating a `Dockerfile` or `Singularity definition` file, however you must explicitly download and compile every package within the instruction file, which can be extremely tedious. `Conda` alleviates this problem by pulling packages from the conda repository for you and installing them to the conda environment $PATH.

8.1 Creating Environments

The main advantage of `conda` is that one can create a 'clean slate' environment for a project - a directory that contains a specific collection of conda packages you have installed that will not interfere with other environments or your system.

To create a new environment, run the following command:

```
$ conda create -n test_env
```

Activate/deactivate the environment using:

```
$ conda activate test_env

$ conda deactivate test_env
```

8.2 Installing packages

There are 2 ways to install packages using conda:

```
$ conda activate test_env

$ conda install bioconda::fastqc
```

Or specify the package version:

```
$ conda activate test_env

$ conda install bioconda::fastqc=0.11.9
```

Warning: Be very careful using pinned versions of packages. In some scenarios a pinned package will require outdated dependencies, causing a conflict when compiling the environment.

8.3 YAML

The preferred, reproducible method for installing `conda` packages is to use a `YAML` file.

See below for a `YAML` file to recapitulate the `test_env` we created above:

Note: Delete `test_env` - we will recreate it using `YAML` files as a proof of concept: `conda env remove --name test_env --all`

```
name: test_env
channels:
 - bioconda
dependencies:
 - fastqc
```

Save the file and name it `environment.yml`. Now compile the environment using conda:

```
$ conda env create -f environment.yml && conda clean -a

$ conda activate test_env

$ fastqc -h
```

8.4 Executable directory

Where have the environments and packages been installed?

The environments are stored under:

```
$ ls -la ~/.conda/envs/
```

To take a look at the executables in the `test_env` environment:

```
$ ls -la ~/.conda/envs/test_env/bin/
```

We will keep this in mind when creating containers using a `Dockerfile`.

CHAPTER

NINE

DOCKER

9.1 Dockerfile

To create a Docker container, we need to construct a Dockerfile which contains instructions on which base image to use, and installation rules.

In the same directory where you previously created the Conda YAML file, copy the following file and save it as a Dockerfile:

```
FROM nfcore/base:1.14
LABEL authors="Barry Digby" \
      description="Docker container containing fastqc"

WORKDIR ./
COPY environment.yml ./
RUN conda env create -f environment.yml && conda clean -a
ENV PATH /opt/conda/envs/test_env/bin:$PATH
```

We are using a pre-built ubuntu image (FROM nfcore/base:1.14) that comes with Conda pre-installed developed by nf-core.

Note: In your Dockerhub account, create a repository called 'test'. We will build and push the docker image in the following section.

9.2 Build image

To build the image, run the following command:

```
$ docker build -t USERNAME/test $(pwd)
```

9.3 Check image

You can shell into your image to double check that the tools have been installed correctly:

```
$ docker images # check images in cache

$ docker run -it barryd237/test
```

9.4 Push to Dockerhub

Now the image has been created, push to `Dockerhub`:

First time push requires you to login:

```
$ docker login
```

```
$ sudo chmod 666 /var/run/docker.sock
```

```
$ (sudo)?? docker push USERNAME/test
```

9.5 Advanced use

There will be scenarios in which your tool of choice is not in the Anaconda repository meaning you cannot download it via the `environment.yml` file.

You will have to provide install instructions to the `Dockerfile`.

Note: This is fairly tedious, you have to perform a dry-run locally before providing the instructions to the `Dockerfile`.

Let's pretend that `Bowtie2` is not available via the Anaconda repository - go to the Github repository containing the latest release: https://github.com/BenLangmead/bowtie2

1. Download the lastest release (`2.4.X`) of `Bowtie2`. Make sure to download the `Source code (tar.gz)` file.
2. Untar the archive file by running `tar -xvzf v2.4.5.tar.gz`.
3. Move to the unzipped directory and figure out if you need to compile the source code. (There is a `Makefile` present - we do need to compile the code).
4. In the `bowtie2-2.4.5/` directory, run the command `make` to compile the code.
5. Do you need to change permissions for the executables?
6. Move the executables to somewhere in your `$PATH`. This can be done two ways:
    1. By moving the executables to a directory in your `$PATH` such as `/usr/local/bin`, `/usr/bin` etc like so: `sudo mv bowtie2-2.4.5/bowtie2* /usr/local/bin/`.
    2. By manually adding a directory to your `$PATH`: `export PATH="/data/bowtie2-2.4.5/:$PATH"`.
7. Test the install by printing the documentation: `bowtie2 -h`

You will need to perform each of the above tasks in your `Dockerfile` - which is done 'blind' hence the need for a dry-run.

Note: Whilst the `nf-core` image we are using contains a handful of tools, containers are usually a clean slate. You have to install basics such as `unzip`, `curl` etc..

```
FROM nfcore/base:1.14
LABEL authors="Barry Digby" \
      description="Docker container containing stuff"

# We need to install tar
RUN apt-get update; apt-get clean all; apt-get install --yes tar

# Install our conda environment, if you want to.
WORKDIR ./
COPY environment.yml ./
RUN conda env create -f environment.yml && conda clean -a
ENV PATH=/opt/conda/envs/test_env/bin:$PATH

# Make a 'scratch' directory.
RUN mkdir -p /usr/src/scratch
# Set scratch directory as working directory (where we will download the source code to)
WORKDIR /usr/src/scratch
# Download the source code
RUN wget https://github.com/BenLangmead/bowtie2/archive/refs/tags/v2.4.5.tar.gz
# untar the source code
RUN tar -xvzf v2.4.5.tar.gz
# Compile the source code
RUN cd bowtie2-2.4.5/ && make
# Add the executable directory to your path
ENV PATH=/usr/src/scratch/bowtie2-2.4.5/:$PATH
```

CHAPTER

TEN

SINGULARITY

Why use both `Docker` and `Singularity`?

`Singularity` can be thought of as the bona fide 'open source' container platform, and offers some advantages over `Docker`:

- `Singularity` does not require sudo privelages. This means we can run `Singularity` on HPC/cloud platforms.
- `Singularity` is compatible with `Docker` i.e we can pull images from `Dockerhub` using `Singularity`.

The main reason we are using `Docker` is that it is compatitble with Github actions (CI testing).

10.1 Singularity pull

On your local machine, pull the docker image we created in the previous step:

```
$ singularity pull --name test.img docker://USERNAME/test
```

The container *test.img* should be present in your directory. Shell into the container:

```
$ singularity shell -B $(pwd) test.img
```

Note: The `-B` flag indicates the bind path for the container. Your container will not be able to access files above `$(pwd)` in the directory tree.

Confirm the installation path of `fastqc` within the container:

```
$ whereis fastqc
```

CHAPTER

ELEVEN

GITHUB SYNCING

Now that we have created a container for our project, we will use `Github Actions` to set up automated `Docker build` and `Docker push` triggers.

11.1 Initialise Repository

Create a repository on Github called `rtp_workshop`. Initialise it with a README.md file. Note how the default branch is called `main`.

On your laptop, move to the directory where you have created the `Dockerfile`, `environment.yml` file and the container `test.img`.

> **Warning:** The maximum file size permitted on Github is 20Mb. We will create a `.gitignore` file to tell github to ignore the container when pushing to Github.

Create the following `.gitignore` file in the directory:

```
*.img
```

> **Note:** Please make sure you have the latest version of `git` installed.

Create a `remote` connection (`origin`) pointing to our Github repository `rtp_workshop`:

```
$ git init # automatically checks out as master branch

$ git remote add origin git@github.com:BarryDigby/rtp_workshop.git

$ git pull origin main # pull README.md

$ git add . # stage local files

$ git commit -m "push to master"

$ git push --set-upstream origin master
```

We have now created a branch `master` that contains all of our local files, and the original README.md file used to initialise the repository. If you want to run a sanity check, run `git branch -a` - you will see that we are on the local `master` branch, and there are two remotes - `main` and `master` (on Github).

Now create a `dev` branch, and make it even with `master`:

```
$ git checkout -b dev # create dev branch

$ git pull origin master # make even with master

$ git push --set-upstream origin dev # create dev branch on Github

$ git branch -a
```

Great! We have both `master` and `dev` locally and on Github. Now I want to delete the `main` branch (this caused a world of pain when Github decided to stop using the term `master`).

Do this via Github:

- In the `rtp_workshop` repository, click on the branch icon.
- Change the default branch to `dev`.
- Delete the branch `main`.

We must update these changes in our remote. Go to the directory containing the repository:

```
$ git checkout dev # go to dev if not already there

$ git remote prune origin # update local to reflect changes we made on Gtihub

$ git branch -a # sanity check.
```

11.2 Docker Workflow

Now that we have set up our github branches correctly, let's add a trigger: every time we push to `dev`, the Docker container is built and pushed to Dockerhub.

Locally, using VSCode or the command line, create the file `.github/workflows/push_dockerhub_dev.yml`:

```
name: RTP Docker push (dev)
# This builds the docker image and pushes it to DockerHub
# Runs on push events to 'dev', including PRs.
on:
  push:
    branches:
      - dev

jobs:
  push_dockerhub:
    name: Push new Docker image to Docker Hub (dev)
    runs-on: ubuntu-latest
    # Only run for your repo
    if: ${{ github.repository == 'BarryDigby/rtp_workshop' }}
    env:
      DOCKERHUB_USERNAME: ${{ secrets.DOCKERHUB_USERNAME }}
      DOCKERHUB_PASS: ${{ secrets.DOCKERHUB_PASS }}
    steps:
      - name: Check out pipeline code
```

(continues on next page)

(continued from previous page)

```
      uses: actions/checkout@v2

    - name: Build new docker image
      run: docker build --no-cache . -t barryd237/test:dev

    - name: Push Docker image to DockerHub (dev)
      run: |
        echo "$DOCKERHUB_PASS" | docker login -u "$DOCKERHUB_USERNAME" --password-stdin
        docker push barryd237/test:dev
```

Note: Please substitute `BarryDigby` with your Github username, and `barryd237` with your Dockerhub username.

11.3 Github Secrets

Those of you with a keen eye will have noticed two environment variables in the `push_dockerhub_dev.yml` file: `DOCKERHUB_USERNAME` and `DOCKERHUB_PASS`, no prizes for guessing what these stand for.

To set up Github secrets, navigate to your GitHub repository and click Settings > Secrets > New secret. Please add both secrets, your username and password.

Your `dev` branch should now be set up to automatically push to Dockerhub.

CHAPTER

TWELVE

FIRST SCRIPT

We will write a basic nextflow script to perform QC on sequencing reads using `FastQC`.

Before getting started with the nextflow script, add the tools needed for todays container (overwrite yesterdays assignment/save elsewhere). We will be working from our local clone of the `rtp_workshop` repository today.

```
name: test_env
channels:
 - bioconda
dependencies:
 - fastqc
 - multiqc
 - gffread
 - kallisto
```

```
FROM nfcore/base:1.14
LABEL authors="Barry Digby" \
    description="Docker container containing fastqc"

WORKDIR ./
COPY environment.yml ./
RUN conda env create -f environment.yml && conda clean -a
ENV PATH /opt/conda/envs/test_env/bin:$PATH
```

Your local directory of the repository should look like:

```
barry@YT-1300:/data/github/test$ ls -la
total 321416
drwxrwxr-x  4 barry barry      4096 Dec 13 14:22 .
drwxrwxr-x 21 barry barry      4096 Dec 13 16:50 ..
-rw-rw-r--  1 barry barry       245 Dec 13 12:20 Dockerfile
-rw-rw-r--  1 barry barry        61 Dec 13 12:20 environment.yml
drwxrwxr-x  8 barry barry      4096 Dec 13 15:07 .git
drwxrwxr-x  3 barry barry      4096 Dec 13 14:22 .github
-rw-rw-r--  1 barry barry         6 Dec 13 14:06 .gitignore
-rw-rw-r--  1 barry barry        32 Dec 13 15:07 README.md
-rwxrwxr-x  1 barry barry 329093120 Dec 13 13:48 test.img
```

Warning: We will build on this directory as the day goes on - make sure you have everything in order now.

12.1 Scripting Language

Nextflow scripts use `groovy` as the main scripting language however, the script body within processes are polyglot - one of the main attractions of nextflow.

```
#!/usr/bin/env nextflow

params.foo = "String"
params.bar = 5

println params.foo.size()

process TEST{

    echo true

    input:
    val(foo) from params.foo
    val(bar) from params.bar

    script:
    """
    echo "Script body printing foo: $foo, bar: $bar"
    """
}
```

Save the script to a file `test.nf` and run it using `nextflow run test.nf`:

```
nextflow run test.nf
N E X T F L O W  ~  version 21.04.1
Launching `test.nf` [nice_austin] - revision: 56da2768ff
6
executor >  local (1)
[ab/90ba6d] process > TEST [100%] 1 of 1 ✓
Script body printing foo: String, bar: 5
```

Warning: Please use 4 whitespaces as indentation for process blocks. Do not use tabs.

Notice that the scripting language outside of the process (`println`) is written in `groovy`. The process body script automatically uses `bash` - but we can perscribe a different language using a `shebang` line:

```
#!/usr/bin/env nextflow

params.foo = "String"
params.bar = 5

println params.foo.size()

process TEST{

    echo true
```

(continues on next page)

(continued from previous page)

```
    input:
    val(foo) from params.foo
    val(bar) from params.bar

    script:
    """
    #!/usr/bin/perl

    print scalar reverse ("Script body printing foo:, $foo, bar:, $bar")
    """
}
```

```
nextflow run test.nf
N E X T F L O W  ~  version 21.04.1
Launching `test.nf` [gloomy_perlman] - revision: 6e0da47179
6
executor >  local (1)
[17/92a7c9] process > TEST [100%] 1 of 1 ✓
5 ,:rab ,gnirtS ,:oof gnitnirp ydob tpircS
```

12.2 Channels

Channels are used to stage files and values in nextflow. There are two types of channels - `queue channels` and `value channels`. Broadly speaking, queue channels are used to connect files to processes and cannot be reused. Value channels on the other hand hold a value (or file value - i.e a path to a file), and can be re-used mutliple times.

Let's use some simulated RNA-Seq reads:

```
wget https://github.com/BarryDigby/circ_data/releases/download/RTP/test-datasets.tar.gz &
↪& tar -xvzf test-datasets.tar.gz

ls -la test-datasets/fastq
total 151M
-rw-rw-r-- 1 barry 11M Nov 22 12:16 fust1_rep1_1.fastq.gz
-rw-rw-r-- 1 barry 12M Nov 22 12:16 fust1_rep1_2.fastq.gz
-rw-rw-r-- 1 barry 14M Nov 22 12:16 fust1_rep2_1.fastq.gz
-rw-rw-r-- 1 barry 15M Nov 22 12:16 fust1_rep2_2.fastq.gz
-rw-rw-r-- 1 barry 14M Nov 22 12:16 fust1_rep3_1.fastq.gz
-rw-rw-r-- 1 barry 16M Nov 22 12:16 fust1_rep3_2.fastq.gz
-rw-rw-r-- 1 barry 11M Nov 22 12:16 N2_rep1_1.fastq.gz
-rw-rw-r-- 1 barry 12M Nov 22 12:16 N2_rep1_2.fastq.gz
-rw-rw-r-- 1 barry 12M Nov 22 12:16 N2_rep2_1.fastq.gz
-rw-rw-r-- 1 barry 15M Nov 22 12:16 N2_rep2_2.fastq.gz
-rw-rw-r-- 1 barry 11M Nov 22 12:16 N2_rep3_1.fastq.gz
-rw-rw-r-- 1 barry 13M Nov 22 12:16 N2_rep3_2.fastq.gz
```

12.2.1 Queue Channels

Now that we have real data to work with, practice staging the files using the `fromFilePairs()` operator:

```
#!/usr/bin/env nextflow

Channel.fromFilePairs("test-datasets/fastq/*_{1,2}.fastq.gz", checkIfExists: true)
       .set{ ch_reads }

ch_reads.view()
```

Overwrite the `test.nf` script and run it using `nextflow run test.nf`. The output should look like:

```
nextflow run foo.nf
N E X T F L O W  ~  version 21.04.1
Launching `foo.nf` [sleepy_brahmagupta] - revision: d316cf84b0
[fust1_rep3, [/data/test/test-datasets/fastq/fust1_rep3_1.fastq.gz, /data/test/test-
↪datasets/fastq/fust1_rep3_2.fastq.gz]]
[N2_rep3, [/data/test/test-datasets/fastq/N2_rep3_1.fastq.gz, /data/test/test-datasets/
↪fastq/N2_rep3_2.fastq.gz]]
[fust1_rep1, [/data/test/test-datasets/fastq/fust1_rep1_1.fastq.gz, /data/test/test-
↪datasets/fastq/fust1_rep1_2.fastq.gz]]
[fust1_rep2, [/data/test/test-datasets/fastq/fust1_rep2_1.fastq.gz, /data/test/test-
↪datasets/fastq/fust1_rep2_2.fastq.gz]]
[N2_rep2, [/data/test/test-datasets/fastq/N2_rep2_1.fastq.gz, /data/test/test-datasets/
↪fastq/N2_rep2_2.fastq.gz]]
[N2_rep1, [/data/test/test-datasets/fastq/N2_rep1_1.fastq.gz, /data/test/test-datasets/
↪fastq/N2_rep1_2.fastq.gz]]
```

The files have been stored in a `tuple`, which is similar to dictionaries in python, or a list of lists. The `fromFilePairs()` operator automatically names each tuple according to the grouping key - e.g `fust1_rep3` - and places the fastq file pairs in a list within the tuple.

When used as inputs, the process will submit a job for each line in the channel in parallel.

Note: Queue channels are FIFO.

To read in a single file, use the `fromPath()` operator:

```
#!/usr/bin/env nextflow

Channel.fromPath("test-datasets/reference/chrI.gtf")
    .set{ ch_gtf }

ch_gtf.view()
```

```
N E X T F L O W  ~  version 21.04.1
Launching `foo.nf` [scruffy_marconi] - revision: 45988ab471
/data/test/test-datasets/reference/chrI.gtf
```

One can also use wildcard glob patterns in conjunction with `fromPath()`:

```
#!/usr/bin/env nextflow

Channel.fromPath("test-datasets/reference/*")
    .set{ ch_reference_files }

ch_reference_files.view()
```

```
nextflow run foo.nf
N E X T F L O W  ~  version 21.04.1
Launching `foo.nf` [soggy_descartes] - revision: e3125b3a9e
/data/test/test-datasets/reference/mature.fa
/data/test/test-datasets/reference/chrI.fa.fai
/data/test/test-datasets/reference/chrI.gtf
/data/test/test-datasets/reference/chrI.fa
```

This is not a great idea in this example - you will have to manually extract each file from the channel. It makes more sense to stage each file in their own channel for downstream analysis.

12.2.2 Value Channels

Value channels (singleton channels) are bound to a single variable and can be read mutliple times - unlike queue channels.

One would typically stage a single file path here, or a parameter variable:

```
#!/usr/bin/env nextflow

Channel.value("test-datasets/reference/chrI.gtf")
   .set{ ch_gtf }

ch_gtf.view()
ch_gtf.view()
```

```
nextflow run foo.nf
N E X T F L O W  ~  version 21.04.1
Launching `foo.nf` [sleepy_thompson] - revision: 76d154a8f4
test-datasets/reference/chrI.gtf
test-datasets/reference/chrI.gtf
```

Note: You cannot perform operations on a value channel.

```
#!/usr/bin/env nextflow

Channel.value("test-datasets/reference/chrI.gtf")
    .set{ ch_gtf }

ch_gtf.map{ it -> it.baseName }.view()
```

```
nextflow run foo.nf
N E X T F L O W  ~  version 21.04.1
```

(continues on next page)

(continued from previous page)

```
Launching `foo.nf` [clever_mclean] - revision: 4cf48e7013
No such variable: baseName

-- Check script 'foo.nf' at line: 6 or see '.nextflow.log' file for more details
```

12.2.3 Channel.value(file())

There exists a workaround for staging a value channel that can both be re-used and allow operations.

`nf-core` devs never raised an issue with my using this method, as far as I am aware it is legitimate.

```
#!/usr/bin/env nextflow

Channel.value(file("test-datasets/reference/chrI.gtf"))
    .set{ ch_gtf }

ch_gtf.view()
ch_gtf.map{ it -> it.baseName }.view()
```

```
nextflow run foo.nf
N E X T F L O W  ~  version 21.04.1
Launching `foo.nf` [gloomy_almeida] - revision: 6b54fe867d
/data/test/test-datasets/reference/chrI.gtf
chrI
```

12.3 Processes

After staging the sequncing reads, we will create a process called `FASTQC` to perform quality control analysis:

```
#!/usr/bin/env nextflow

Channel.fromFilePairs("test-datasets/fastq/*_{1,2}.fastq.gz", checkIfExists: true)
       .set{ ch_reads }

process FASTQC{
    publishDir "./fastqc", mode: 'copy'

    input:
    tuple val(base), file(reads) from ch_reads

    output:
    file("*.{html,zip}") into ch_multiqc

    script:
    """
    fastqc -q $reads
    """
}
```

To run the script, we need to point to the container which holds the `FastQC` executable. To do this, we specify `-with-singularity 'path/to/image'`.

```
nextflow run test.nf -with-singularity 'test.img'
```

This should raise an error about 'no such file or directory'. In short, the singularity container does not know where to look for the files when we run the script.

12.4 Configuration file

This brings us along nicely to the `nextflow.config` file. This file is used to specify nextflow variables and parameters for the workflow.

In the file below, we specify the `bind path` of the container for each process, and enable singularity (we could specify `podman`, `docker`, etc here if we needed to).

In the same directory, save the contents below to a file named `nextflow.config`:

```
process{
  containerOptions = '-B /data/'
}

singularity.enabled = true
singularity.autoMounts = true
```

Now run the script again:

```
nextflow run test.nf -with-singularity 'test.img' -c nextflow.config
```

Tip: You can save the file under `~/.nextflow/config` - nextflow will automatically check this location for a configuration file, bypassing the need to specify the `-c` flag.

The results of `fastqc` are stored in the output directory `fastqc/`. We specified two output file types, `.html` and `.zip`, and as such, these are the files published in the output directory.

12.5 Parameters

Parameters are variables passed to the nextflow workflow.

It is poor practice to hardcode paths within a workflow - nextflow offers two methods to pass parameters to a workflow:

1. Via the command line
2. Via a configuration file

12.5.1 Command Line Parameters

Using the previous script as an example, we will remove the hardcoded variables and pass the parameter via the command line. Edit your script like so (I'm only showing the relevant lines):

```
#!/usr/bin/env nextflow

Channel.fromFilePairs( params.input, checkIfExists: true )
       .set{ ch_reads }
```

Pass the path to `params.input`:

```
$ nextflow run test.nf --input "test-dataset/fastq/*_{1,2}.fastq.gz" -with-singularity
↪'test.img' -c nextflow.config
```

12.5.2 Configuration Parameters

Alternatively, we can specify parameters via any `*.config` file. You can supply multiple configuration profiles to a workflow. Please bear in mind that the order matters - duplicate parameters will be overwritten by subsequent configuration profiles.

For now, add them to the `nextflow.config` file we created:

```
process{
  containerOptions = '-B /data/'
}

params{
  input = "/data/test/test-dataset/fastq/*_{1,2}.fastq.gz"
}

singularity.enabled = true
singularity.autoMounts = true
```

This circumvents the need to pass multiple parameters via the command line.

```
nextflow run test.nf -with-singularity 'test.img' -c nextflow.config
```

Note: Please use double quotes when using a wildcard glob pattern.

Note: It is good practice to provide the absolute paths to files.

Please complete Assignment II Part 1.

CHAPTER

THIRTEEN

GITHUB SYNCING

One of the coolest things about nextflow is that you can deploy scripts directly from Github - provided the repository is set up correctly.

Go to your local clone of your `rtp_workshop` repository:

1. Rename `test.nf` as `main.nf`.
2. Update your `nextflow.config` file:

```
process{
  container = 'barryd237/test:dev'
  containerOptions = '-B /data/'
}

params{
  input = "/data/test/test-dataset/fastq/*_{1,2}.fastq.gz"
}

singularity.enabled = true
singularity.autoMounts = true
singularity.cacheDir = '/data/containers'
```

Push the changes to Github:

```
git add .
git commit -m "Prepare repo for deployment"
git push
```

The repository can now be deployed directly from GitHub:

```
nextflow pull BarryDigby/rtp_workshop

nextflow run -r dev BarryDigby/rtp_workshop
```

Pretty nifty.

CHAPTER

FOURTEEN

OPERATORS

Nextflow uses `operators` to filter, transform, split, combine and carry out mathematical operations on channels.

We will cover some of the most commonly used operators below, using `dummy files`.

Dummy files are empty files that contain file extensions we can test within the script.

> **Warning:** One of the most common mistakes is to test the workflow on a full size dataset. This can be extremely time consuming and burns through uneccessary computational resources.

14.1 Map

The `map{}` operator performs a mapping function on an input channel. Conceptually, `map` allows you to re-organise the structure of a channel.

> **Hint:** nextflow uses 0 based indexing

```
#!/usr/bin/env nextflow

Channel.from( ['A', 1, 2], ['B', 3, 4] )
    .map{ it -> it[0] }
    .view()

Channel.from( ['A', 1, 2], ['B', 3, 4] )
    .map{ it -> [ it[1], it[2] ] }
    .view()
```

```
$nextflow run map.nf
N E X T F L O W  ~  version 21.04.1
Launching `map.nf` [jovial_stallman] - revision: 476751b062
A
B
[1, 2]
[3, 4]
```

14.2 Join

The `join()` operator combines two channels according to a common tuple key. The order in which you supply channels to `join()` matters:

```
#!/usr/bin/env nextflow

ch_genes = Channel.from( ['SRR0001', 'SRR0001_mRNA.txt'], ['SRR0002', 'SRR0002_mRNA.txt
↪'] )
                  .view()

ch_mirna = Channel.from( ['SRR0001', 'SRR0001_miRNA.txt'], ['SRR0002', 'SRR0002_miRNA.txt
↪'] )
                  .view()

all_files = ch_genes.join(ch_mirna).view()
```

```
$ nextflow run map.nf
N E X T F L O W  ~  version 21.04.1
Launching `join.nf` [gloomy_elion] - revision: 85b961030d
[SRR0001, SRR0001_mRNA.txt]
[SRR0002, SRR0002_mRNA.txt]
[SRR0001, SRR0001_miRNA.txt]
[SRR0002, SRR0002_miRNA.txt]
[SRR0001, SRR0001_mRNA.txt, SRR0001_miRNA.txt]
[SRR0002, SRR0002_mRNA.txt, SRR0002_miRNA.txt]
```

14.3 BaseName

Those familiar with bash will recognise commands such as `basename /path/to/file.txt`, `${VAR%pattern}` to strip the path and file extension, respectively.

In nextflow, the same can be achieved using `Name`, `baseName`, `simpleName` and `Extension`.

Let's use it in conjunction with `map{}`:

Note: This operation must be performed on a `file`, not a string. We must read in a dummy file using `fromPath()`. Don't get too caught up on this, I am just demonstrating the functions.

```
#!/usr/bin/env nextflow

Channel.fromPath( "dummy_files/SRR0001_R{1,2}.fastq.gz" )
    .view()
    .map{ it -> [ it.Name, it.baseName, it.simpleName, it.Extension ] }
    .view()
```

```
nextflow run map.nf
N E X T F L O W  ~  version 21.04.1
Launching `map.nf` [curious_newton] - revision: cd2c4772e7
/data/test/dummy_files/SRR0001_R1.fastq.gz
```

(continues on next page)

(continued from previous page)

```
/data/test/dummy_files/SRR0001_R2.fastq.gz
[SRR0001_R1.fastq.gz, SRR0001_R1.fastq, SRR0001_R1, gz]
[SRR0001_R2.fastq.gz, SRR0001_R2.fastq, SRR0001_R2, gz]
```

14.4 Flatten

The `flatten()` operator will transform channels in a manner such that each item in the channel is output one by one.

Say for example we wanted to feed in our fastq files one by one to a process (each process is run in parallel - this could speed up our workflow) we would use `flatten()`.

Let's use the dummy files as an example:

```
#!/usr/bin/env nextflow

Channel.fromFilePairs( "dummy_files/SRR000*_R{1,2}.fastq.gz" )
    .map{ it -> [ it[1][0], it[1][1] ] }
    .flatten()
    .view()
```

```
$nextflow run map.nf
N E X T F L O W  ~  version 21.04.1
Launching `map.nf` [nice_sinoussi] - revision: 403faf87e0
/data/test/dummy_files/SRR0002_R1.fastq.gz
/data/test/dummy_files/SRR0002_R2.fastq.gz
/data/test/dummy_files/SRR0007_R1.fastq.gz
/data/test/dummy_files/SRR0007_R2.fastq.gz
/data/test/dummy_files/SRR0003_R1.fastq.gz
/data/test/dummy_files/SRR0003_R2.fastq.gz
/data/test/dummy_files/SRR0004_R1.fastq.gz
/data/test/dummy_files/SRR0004_R2.fastq.gz
/data/test/dummy_files/SRR0009_R1.fastq.gz
/data/test/dummy_files/SRR0009_R2.fastq.gz
/data/test/dummy_files/SRR0008_R1.fastq.gz
/data/test/dummy_files/SRR0008_R2.fastq.gz
/data/test/dummy_files/SRR0006_R1.fastq.gz
/data/test/dummy_files/SRR0006_R2.fastq.gz
/data/test/dummy_files/SRR0001_R1.fastq.gz
/data/test/dummy_files/SRR0001_R2.fastq.gz
/data/test/dummy_files/SRR0005_R1.fastq.gz
/data/test/dummy_files/SRR0005_R2.fastq.gz
```

CHAPTER

FIFTEEN

CONDITIONALS

Nextflow uses conditionals as a 'flow control' mechanism to skip process blocks or operations. The most obvious example is when re-running an alignment workflow where we have already created or downloaded the index file. We don't want to keep re-running the labor intensive indexing process.

There are some strict rules regarding the use of conditionals:

- Boolean parameters should be set to `true/false` in `nextflow.config`.
- File Paths / Strings / Integers / Floats / Lists / Maps should be set to `null` in `nextflow.config`.

Nextflow also makes heavy use of `ternary operators`. The code line `A ? B : C` reads if A is true, choose B, else C.

Note: We will continue with our `RNA-Seq` workflow example in this section.

Let's flesh out our `nextflow.config`:

```
process{
  container = "barryd237/test:dev"
  containerOptions = ' -B /data/'
}

params{
  input = "/data/test/test-datasets/fastq/*_{1,2}.fastq.gz"
  fasta = "/data/test/test-datasets/reference/chrI.fa"
  gtf   = "/data/test/test-datasets/reference/chrI.gtf"
  transcriptome = null
  outdir = "/data/test/"
  save_qc_intermediates = true
  save_transcriptome = true
  run_qc = true
}

singularity.enabled = true
singularity.autoMounts = true
singularity.cacheDir = "/data/containers"
```

15.1 Update .gitignore

Update your `.gitignore` file so you don't upload the directories output by our script. As of writing the documentation, this is what mine looks like:

```
*.img
test-datasets/
work/
.nextflow.*
.nextflow/
dummy_files/
fastqc/
multiqc/
```

15.2 Update Script

Overwrite the contents of `main.nf` with the following, and push to GitHub:

```
#!/usr/bin/env nextflow

Channel.fromFilePairs("${params.input}", checkIfExists: true)
       .into{ ch_qc_reads; ch_alignment_reads }

ch_fasta = Channel.value(file(params.fasta))
ch_gtf = Channel.value(file(params.gtf))

process FASTQC{
    tag "${base}"
    publishDir params.outdir, mode: 'copy',
        saveAs: { params.save_qc_intermediates ? "fastqc/${it}" : null }

    when:
    params.run_qc

    input:
    tuple val(base), file(reads) from ch_qc_reads

    output:
    file("*.{html,zip}") into ch_multiqc

    script:
    """
    fastqc -q $reads
    """
}

process MULTIQC{
    publishDir "${params.outdir}/multiqc", mode: 'copy'

    when:
    params.run_qc
```

(continues on next page)

(continued from previous page)

```
    input:
    file(htmls) from ch_multiqc.collect()

    output:
    file("multiqc_report.html") into multiqc_out

    script:
    """
    multiqc .
    """
}

process TX{
    publishDir params.outdir, mode: 'copy',
        saveAs: { params.save_transcriptome ? "reference/transcriptome/${it}" : null }

    when:
    !params.transcriptome && params.fasta

    input:
    file(fasta) from ch_fasta
    file(gtf) from ch_gtf

    output:
    file("${fasta.baseName}.tx.fa") into transcriptome_created

    script:
    """
    gffread -F -w "${fasta.baseName}.tx.fa" -g $fasta $gtf
    """
}

ch_transcriptome = params.transcriptome ? Channel.value(file(params.transcriptome)) :␣
↪transcriptome_created
```

Push to changes to github and run the workflow:

```
git add .

git commit -m "Update repo"

git push

nextflow pull <username>/rtp_workshop

nextflow run -r dev <username>/rtp_workshop

nextflow run main.nf -profile docker -c nextflow.config
```

Note: For those curious, workflows are staged under `~/.nextflow/assets/<github-username>/`

cool.

Go to Assignment II Part 3 :)

CHAPTER

SIXTEEN

ASSIGNMENT I

16.1 Part I: QC Sequencing Reads

Download some simulated RNA-Seq reads:

```
wget https://github.com/BarryDigby/circ_data/releases/download/RTP/fastq.tar.gz && tar -
↪xvzf fastq.tar.gz

ls -la test-datasets/fastq
total 151M
-rw-rw-r-- 1 barry 11M Nov 22 12:16 fust1_rep1_1.fastq.gz
-rw-rw-r-- 1 barry 12M Nov 22 12:16 fust1_rep1_2.fastq.gz
-rw-rw-r-- 1 barry 14M Nov 22 12:16 fust1_rep2_1.fastq.gz
-rw-rw-r-- 1 barry 15M Nov 22 12:16 fust1_rep2_2.fastq.gz
-rw-rw-r-- 1 barry 14M Nov 22 12:16 fust1_rep3_1.fastq.gz
-rw-rw-r-- 1 barry 16M Nov 22 12:16 fust1_rep3_2.fastq.gz
-rw-rw-r-- 1 barry 11M Nov 22 12:16 N2_rep1_1.fastq.gz
-rw-rw-r-- 1 barry 12M Nov 22 12:16 N2_rep1_2.fastq.gz
-rw-rw-r-- 1 barry 12M Nov 22 12:16 N2_rep2_1.fastq.gz
-rw-rw-r-- 1 barry 15M Nov 22 12:16 N2_rep2_2.fastq.gz
-rw-rw-r-- 1 barry 11M Nov 22 12:16 N2_rep3_1.fastq.gz
-rw-rw-r-- 1 barry 13M Nov 22 12:16 N2_rep3_2.fastq.gz
```

Your task is to create a container using a both a conda `environment.yml` file and a suitable `Dockerfile` file hosting the following quality control tools:

- `fastqc`
- `multiqc`

Once the container has been created, shell into the container and run `fastqc` on the sequencing reads. Once all of the outputs have been generated for each fastq file ("*.{html,zip}"), run `multiqc` to generate a summary report.

16.1.1 Bonus

1. Push your Docker container (which should have both `fastqc` and `multiqc` installed) to DockerHub.
2. Download the container using the `singularity pull` command - we are mimicking behaviour on a HPC here where Docker is not available to us.
3. Write a bash script that loops over each fastq file performing `fastqc`.
4. At the end of the script, run `multiqc` on the outputs of the `fastqc` runs.
5. Run the script from within the container by using the `singularity shell` command. Be careful to specify the correct bind path using `-B`.

16.2 Part II: Advanced Container Creation

You are tasked with creating a container to faithfully reproduce the analysis performed by Zhao et al

An excerpt of the methods are given in the screenshot below - create a container using a `Dockerfile` and an `environment.yml` file as shown in previous examples.

Note: There are three tools which you will need to install manually: 1) `RSEM v.1.3.0` 2) `BLASTX v2.6.0` 3) `CNCI v2.0`. You will have to perform a dry-run installation of these tools locally first.

Note: Use pinned tool versions! We want the precise versions used in the analysis.

Note: If a tool is present in multiple channels, be sure to specifically select the channel you want to download it from e.g: `conda-forge::<tool>=<version>`. If you do not do this, conda will not know which channel to use and fail during the install.

16.2.1 RSEM

`RSEM` is written in `C++` and requires a bunch of dependencies which are beyond the scope of this workshop. I have included the dependencies for `RSEM` installation in the `Dockerfile` for you:

```
# Add dependencies
RUN apt-get update; apt-get clean all;

RUN apt-get install --yes build-essential \
                         gcc-multilib \
                         tar \
                         unzip \
                         ncurses-base \
                         zlib1g \
```

(continues on next page)

RNA-seq

Mapping and assembly. The reads were mapped to susScr11 (*Sus scrofa* 11.1) reference genome assemblies using TopHat v2.1.1[83]. The transcript assembly of each sample was performed separately by using a BAM file as the input file for Cufflinks v2.2.1[84]. The resulting individual transcript assemblies were merged further to a single transcript assembly using the Cuffcompare utility, which was included with the Cufflinks package v2.2.1[84].

Identification of lncRNAs and other types of transcripts. To identify lncRNAs, the transcripts from the single transcript assembly were filtered by the following steps: (i) the identified transcripts, which must have class code "x", "i", or "." in at least two samples, but did not overlap with any known transcripts or with class code "u" transcripts; (ii) transcripts with length ≥200 bp and exon number ≥2; (iii) transcripts with FPKM ≥1 and reads ≥5 in at least two replicates; (iv) the identified transcripts without potential coding regions, which were filtered using CNCI v2.0[85] and CPC v2.0[86].

Next, we identified other types of new transcripts of the pig genome. At first, we selected the transcripts that were assembled in two replicates in the transcript assembly file of each sample. Second, the filtered transcript assembly files were merged using the Cuffmerge program of Cufflinks v2.2.1[84]. Then, the Cuffcompare utility of Cufflinks v2.2.1[84] was applied again to integrate the merged transcript assembly file and the lncRNA GTF file. The new GTF file was used to calculate transcript expression levels using RSEM v1.3.0[87]. The last step was to filter the transcripts to identify new transcripts by the following criteria: (i) transcripts on the same strand of known transcripts and exons overlapped with any known transcripts were omitted; (ii) lncRNAs which were identified before were removed; (iii) transcripts with class code "x", "i", or "u" and with FPKM ≥5 at least in any two replicates.

We further calculated the H3K4me3 intensities near to lncRNAs and other types of newly identified transcripts via the computeMatrix function of deepTools v2.0[88], and created a profile plot by using the plotProfile function of deepTools v2.0[88]. The ChIP-seq data were processed using the procedures described below in the "ChIP-seq" section. Then, the newly identified transcripts with protein-coding capacity were selected by CPC v2.0[86] and compared with the NCBI Reference protein database of humans (taxid:9606) using BLASTX v.2.6.0[89]. The cutoff of BLAST results was set as EXPECT threshold $<10^{-5}$.

(continued from previous page)

```
                        liblzma5 \
                        libbz2-1.0 \
                        gcc \
                        g++ \
                        zlib1g-dev
```

When installing `RSEM` in the `Dockerfile`, chain the `wget`, `tar -zxvf`, `cd`, `make` and `make install` commands using `&&`.

Each `RUN` line triggers a new layer - breaking up installation commands over multiple `RUN` lines will fail - Thank you Bianca! :)

16.2.2 CNCI

CNCI is available on Github at the following link. There are two issues here:

1. The authors never bothered to make a stable release, so you cannot download a versioned tarball containing the contents of the repository.
2. Running `git clone` in a `Dockerfile` will fail (`Host key verification failed.`). You need to generate unique `ssh keys` for the container, which are then saved in the image layer. **This is extremely unsecure - don't do this**.

To overcome these issues, I forked the repository and created a stable release - I cloned the repo locally, tarzipped it and uploaded the tarball as a release file. The stable release is available at the following link.

Within the `Dockerfile`, use `wget` to download the archived repository. You can follow the installation steps from there.

Hint: Once downloaded and de-compressed, make the `CNCI` folder fully accessible: `chmod -R 777 CNCI/`. You must do this in order to add the executables to your `$PATH`.

16.2.3 Check Installations

If you need a reminder, the steps to build the container are:

```
docker build -t <dockerhub_username>/<repo_name> $(pwd) # run in directory containing␣
↪both Dockerfile and environment.yml file
docker run -it <dockerhub_username>/<repo_name>
```

Check the installs worked:

```
tophat

cufflinks

rsem-bam2wig

makeblastdb -help

CPC2.py
```

(continues on next page)

(continued from previous page)

```
CNCI.py -h

computeMatrix
```

All of the tools should work except for `Deeptools (computeMatrix)`. This looks like a particularly nasty error to debug - particularly when the tool is coming from the Anaconda repository. You will come across situations like this that will force you to look for alternative tools, or comb through their source code and locate and remedy the error.

Once you are happy with the installations, push your changes to Github to trigger an automated build. (i.e push the `Dockerfile` & `environment.yml` to your repo).

CHAPTER

SEVENTEEN

ASSIGNMENT II

17.1 Part 1

You will need to add the process `MULTIQC` to the `test.nf` script.

Before proceeding, please update your `.gitignore` file:

```
*.img
work/
test-datasets/
work/
.nextflow.*
.nextflow/
fastqc/
```

17.1.1 Update container

Update your `environment.yml` file to include `multiqc`. Push the change to Github to trigger the `Dockerhub` build. You will need to delete your local `test.img` and download the updated version containing `multiqc`.

17.1.2 Update parameters, test.nf

`MultiQC` expects the output from `FastQC` for **all samples**. As such, use the line `file(htmls) from ch_multiqc.collect()` for the input directive to stage every file from the output channel `ch_multiqc` from the process `FASTQC` in our new process `MULTIQC`.

There is no need to specify `tuple val(base)` in the input/output directive. Why? I have responded to a post explaining this, available here: https://www.biostars.org/p/495108/#495150

In addition, add the parameter `outdir` to the `nextflow.config` file - this is the directory we will write results to. Nextflow uses variable expansion just like bash i.e: `"${params.outdir}/fastqc"`.

Hint: The output of `multiqc` is a html file, use the appropriate wildcard glob pattern in the output directive.

When completed, proceed to the section `Github Syncing`.

Warning: Add the folder your multiqc results are in to the `.gitignore` file.

17.2 Part 2

Test your knowledge of the operators we covered.

17.2.1 Map

Create a set of dummy `fastq` files in a directory called `dummy_files`:

> **Warning:** Update your `.gitignore` file now to include `dummy_files/`.

```
mkdir dummy_files
touch dummy_files/SRR000{1..9}_R{1,2}.fastq.gz
```

The directory should now contain 9 dummy paired end fastq files:

```
$ l dummy_files
total 0
-rw-rw-r-- 1 barry 0 Nov 22 09:02 SRR0001_R1.fastq.gz
-rw-rw-r-- 1 barry 0 Nov 22 09:02 SRR0001_R2.fastq.gz
-rw-rw-r-- 1 barry 0 Nov 22 09:02 SRR0002_R1.fastq.gz
-rw-rw-r-- 1 barry 0 Nov 22 09:02 SRR0002_R2.fastq.gz
-rw-rw-r-- 1 barry 0 Nov 22 09:02 SRR0003_R1.fastq.gz
-rw-rw-r-- 1 barry 0 Nov 22 09:02 SRR0003_R2.fastq.gz
-rw-rw-r-- 1 barry 0 Nov 22 09:02 SRR0004_R1.fastq.gz
-rw-rw-r-- 1 barry 0 Nov 22 09:02 SRR0004_R2.fastq.gz
-rw-rw-r-- 1 barry 0 Nov 22 09:02 SRR0005_R1.fastq.gz
-rw-rw-r-- 1 barry 0 Nov 22 09:02 SRR0005_R2.fastq.gz
-rw-rw-r-- 1 barry 0 Nov 22 09:02 SRR0006_R1.fastq.gz
-rw-rw-r-- 1 barry 0 Nov 22 09:02 SRR0006_R2.fastq.gz
-rw-rw-r-- 1 barry 0 Nov 22 09:02 SRR0007_R1.fastq.gz
-rw-rw-r-- 1 barry 0 Nov 22 09:02 SRR0007_R2.fastq.gz
-rw-rw-r-- 1 barry 0 Nov 22 09:02 SRR0008_R1.fastq.gz
-rw-rw-r-- 1 barry 0 Nov 22 09:02 SRR0008_R2.fastq.gz
-rw-rw-r-- 1 barry 0 Nov 22 09:02 SRR0009_R1.fastq.gz
-rw-rw-r-- 1 barry 0 Nov 22 09:02 SRR0009_R2.fastq.gz
```

Create a nextflow script that does the following:

1. Read in the dummy files using `fromFilePairs()`.
2. Place the reads into 2 channels `ch_fwd` and `ch_rev` using `into{a;b}` instead of `.set{}`.
3. Splits the reads into two new channels `forward_reads` and `reverse_reads` using `map`.
4. Use as inputs to a process the forward or reverse read channels and echo their contents in the script body (Hint: use `echo true` at the top of the process).

Hint: Before proceeding to the next step, append the `.view()` operator to double check that the channels hold the correct values.

17.2.2 Join

In the `map` script you created above, use the `join` operator to join the forward and reverse reads into a single channel in the input directive of the process where you `echo` the reads.

In essence, I want you to stage both forward and reverse reads in the process and `echo` them.

You can use `join` outside of, or inside the process - the choice is up to you.

17.3 Part 3

Update your `main.nf` script to include:

17.3.1 Transcriptome indexing

1. Create a process that creates an index file using the transcriptome fasta file.
    1. Name the process `INDEX`.
    2. Include 2 boolean parameters `kallisto_index` and `save_index` in your `nextflow.config` file and script. Use these in a similar fashion to `transcriptome` and `save_transcriptome` parameters.
    3. Include a suitable ternary operator after the `INDEX` process to accept pre-built index files when supplied to the workflow.

17.3.2 Kallisto quantification

1. Create a process that performs kallisto quantification using the index file and sequencing reads.
    1. Name the process `KALLISTO_QUANT`.
    2. Use the reads staged in `ch_alignment_reads` as input to the process - the `ch_qc_reads` channel has already been consumed.

Refer to the Kallisto documentation and inspect the `kalisto index` and `kallisto quant` commands.

Before designing a nextflow workflow, you need to be familiar with the expected outputs generated by the process script body. Shell into your container to run the quantification analysis in bash before implementing the process in nextflow.

CHAPTER

EIGHTEEN

ASSIGNMENT III

Your `main.nf` script should be able to perform `FASTQC` and `MULTIQC` on your sequencing data.

I have added a process called `TX` - which generates a transcriptome FASTA file for `Kallisto`.

Update your `main.nf` script to include:

1. Transcriptome Indexing.
2. Kallisto Quantification.

Refer to the Kallisto documentation and inspect the `kalisto index` and `kallisto quant` commands.

Before designing a nextflow workflow, you need to be familiar with the expected outputs generated by the process script body. Shell into your container to run the quantification analysis in bash before implementing the process in nextflow.

18.1 Template script

I have a heavily commented template script for you to use as a starting point: https://github.com/BarryDigby/rtp_workshop2/blob/dev/template.nf

18.2 DAG

Hopefully the dag gives you clues how to proceed,.,.,.,.??

18.2.1 Advanced script

As it stands, your `main.nf` script will run each process every time we run it. We want to make use of conditional statements to allow users to skip processes - saving computational time and resources.

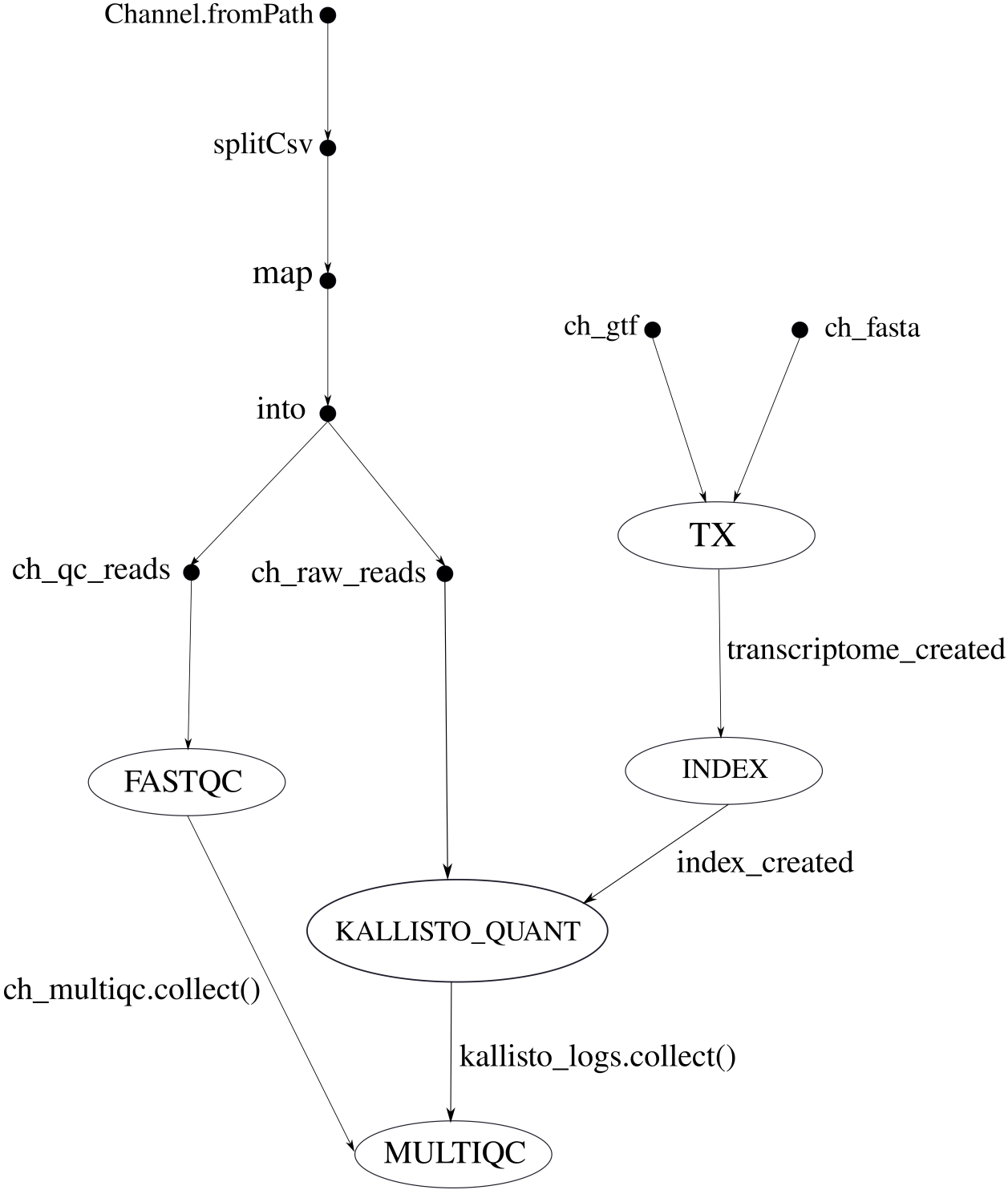
Refer to the `Condtionals` documentation - I have included appropriate parameters and code in both our `nextflow.config` and `main.nf` scripts for quality control and generating the transcriptome file.

Your task is to:

- Include 2 boolean parameters `kallisto_index` and `save_index` in your `nextflow.config` file and script. Use these in a similar fashion to the `transcriptome` and `save_transcriptome` parameters in the documentation.

We do not need any condional statements for the quantification step - we want to run this every time.

Please ask for help if anything is unclear.


Channel.fromPath
splitCsv
map
into
ch_gtf
ch_fasta
TX
ch_qc_reads
ch_raw_reads
transcriptome_created
FASTQC
INDEX
index_created
KALLISTO_QUANT
ch_multiqc.collect()
kallisto_logs.collect()
MULTIQC

CHAPTER

NINETEEN

CONTINUOUS INTEGRATION

Github actions can perform a test run of your workflow using the minimal test-dataset. Just like the Dockerhub continuous integration, the actions are performed upon each push to the `dev` branch.

In order to set this up, we will need to specify both a `test` configuration profile and a `ci.yml` workflow file.

19.1 Test profile

The test configuration profile contains a series of input parameters that will be used as inputs to the workflow for the test run. These parameters point to the URL of the test-dataset hosted on GitHub.

Unfortunately, wildcard glob patterns are not supported via `html` links, so the following is not valid:

```
params{
  input = "https://raw.githubusercontent.com/nf-core/test-datasets/circrna/fastq/*_{1,2}.
↪fastq.gz"
}
```

Here is a valid `test.config` file for our simulated RNA-Seq dataset we have been working with:

```
params {
    config_profile_name = 'Test profile'
    config_profile_description = 'Test dataset to check pipeline function'

    // Limit resources so that this can run on GitHub Actions
    max_cpus = 2
    max_memory = 6.GB
    max_time = 48.h

    // Input data for test data
    input = 'https://raw.githubusercontent.com/BarryDigby/CRT_workshop/master/docs/
↪source/test_datasets/samples.csv'
    fasta = 'https://raw.githubusercontent.com/BarryDigby/CRT_workshop/master/docs/
↪source/test_datasets/chrI.fa'
    gtf = 'https://raw.githubusercontent.com/BarryDigby/CRT_workshop/master/docs/source/
↪test_datasets/chrI.gtf'
    outdir = 'test_outdir/'
}
```

Save the file to `conf/test.config` in your repository.

19.2 Sample File

To overcome the html glob limitation, we need to construct an input samples file.

See below for a valid example of a `samples.csv` file, specifying the links to each fastq file:

```
Sample_ID,Read1,Read2
cel_N2_1,https://raw.githubusercontent.com/nf-core/test-datasets/circrna/fastq/N2_rep1_1.
↪fastq.gz,https://raw.githubusercontent.com/nf-core/test-datasets/circrna/fastq/N2_rep1_
↪2.fastq.gz
cel_N2_2,https://raw.githubusercontent.com/nf-core/test-datasets/circrna/fastq/N2_rep2_1.
↪fastq.gz,https://raw.githubusercontent.com/nf-core/test-datasets/circrna/fastq/N2_rep2_
↪2.fastq.gz
cel_N2_3,https://raw.githubusercontent.com/nf-core/test-datasets/circrna/fastq/N2_rep3_1.
↪fastq.gz,https://raw.githubusercontent.com/nf-core/test-datasets/circrna/fastq/N2_rep3_
↪2.fastq.gz
fust1_1,https://raw.githubusercontent.com/nf-core/test-datasets/circrna/fastq/fust1_rep1_
↪1.fastq.gz,https://raw.githubusercontent.com/nf-core/test-datasets/circrna/fastq/fust1_
↪rep1_2.fastq.gz
fust1_2,https://raw.githubusercontent.com/nf-core/test-datasets/circrna/fastq/fust1_rep2_
↪1.fastq.gz,https://raw.githubusercontent.com/nf-core/test-datasets/circrna/fastq/fust1_
↪rep2_2.fastq.gz
fust1_3,https://raw.githubusercontent.com/nf-core/test-datasets/circrna/fastq/fust1_rep3_
↪1.fastq.gz,https://raw.githubusercontent.com/nf-core/test-datasets/circrna/fastq/fust1_
↪rep3_2.fastq.gz
```

Instead of supplying the path to sequencing reads as `params.input`, we can provide the `samples.csv` file. Save this file in your directory to test it out.

We will need to use custom functions to read in the file and stage them as inputs for our workflow.

See the nextflow script below. Save it and run `nextflow run <script_name>.nf --input 'samples.csv'`

Note: We are testing this locally, so we are not deploying from Github. If you are not in the directory containing the `nextflow.config` file, specify it's path with the `-c` argument.

```
#!/usr/bin/env nextflow

// parse input data
if(has_extension(params.input, ".csv")){

   csv_file = file(params.input, checkIfExists: true)
   ch_input = extract_data(csv_file)

}else{

   exit 1, "error: The sample input file must have the extension '.csv'."

}

// stage input data
( ch_qc_reads, ch_raw_reads) = ch_input.into(2)
```

(continues on next page)

(continued from previous page)

```
ch_raw_reads.view()

process FASTQC{
    tag "${base}"
    publishDir params.outdir, mode: 'copy',
        saveAs: { params.save_qc_intermediates ? "fastqc/${it}" : null }

    when:
    params.run_qc

    input:
    tuple val(base), file(reads) from ch_qc_reads

    output:
    tuple val(base), file("*.{html,zip}") into ch_multiqc

    script:
    """
    fastqc -q $reads
    """
}

/*
================================================================================
                              AUXILLARY FUNCTIONS
================================================================================
*/

// Check if a row has the expected number of item
def checkNumberOfItem(row, number) {
    if (row.size() != number) exit 1, "error:  Invalid CSV input - malformed row (e.g.␣
↪missing column) in ${row}, consult documentation."
    return true
}

// Return file if it exists
def return_file(it) {
    if (!file(it).exists()) exit 1, "error: Cannot find supplied FASTQ input file. Check␣
↪file: ${it}"
    return file(it)
}

// Check file extension
def has_extension(it, extension) {
    it.toString().toLowerCase().endsWith(extension.toLowerCase())
}

// Parse samples.csv file
def extract_data(csvFile){
    Channel
        .fromPath(csvFile)
        .splitCsv(header: true, sep: ',')
```

(continues on next page)

(continued from previous page)

```
        .map{ row ->

        def expected_keys = ["Sample_ID", "Read1", "Read2"]
        if(!row.keySet().containsAll(expected_keys)) exit 1, "error: Invalid CSV input -␣
↪malformed column names. Please use the column names 'Sample_ID', 'Read1', 'Read2'."

        checkNumberOfItem(row, 3)

        def samples = row.Sample_ID
        def read1 = row.Read1.matches('NA') ? 'NA' : return_file(row.Read1)
        def read2 = row.Read2.matches('NA') ? 'NA' : return_file(row.Read2)

        if( samples == '' || read1 == '' || read2 == '' ) exit 1, "error: a field does␣
↪not contain any information. Please check your CSV file"
        if( !has_extension(read1, "fastq.gz") && !has_extension(read1, "fq.gz") && !has_
↪extension(read1, "fastq") && !has_extension(read1, "fq")) exit 1, "error: A R1 file␣
↪has a non-recognizable FASTQ extension. Check: ${r1}"
        if( !has_extension(read2, "fastq.gz") && !has_extension(read2, "fq.gz") && !has_
↪extension(read2, "fastq") && !has_extension(read2, "fq")) exit 1, "error: A R2 file␣
↪has a non-recognizable FASTQ extension. Check: ${r2}"

        // output tuple mimicking fromFilePairs
        [ samples, [read1, read2] ]

        }
}
```

Note: nextflow will only download the files once they are passed to a process.

Note: note to barry: integrate these functions to students main.nf before proceeding.

19.3 CI.yml

'All' that is left is to set up the Github actions file and integrate two profiles, `test` and `docker`.

Create the following file in your directory: `.github/workflows/ci.yml`:

Warning: I cannot stress how important indentation is with .yml files.

```
name: CI
# This workflow runs the pipeline with the minimal test dataset to check that it␣
↪completes without any syntax errors
on:
  push:
    branches:
      - dev
```

(continues on next page)

(continued from previous page)

```
  pull_request:
  release:
    types: [published]

jobs:
  test:
    name: Run workflow tests
    # Only run on push if this is the nf-core dev branch (merged PRs)
    if: ${{ github.event_name != 'push' || (github.event_name == 'push' && github.
↪repository == 'BarryDigby/rtp_workshop') }}
    runs-on: ubuntu-latest
    env:
      NXF_VER: ${{ matrix.nxf_ver }}
      NXF_ANSI_LOG: false
    strategy:
      matrix:
        # Nextflow versions: specify nextflow version to use
        nxf_ver: ['21.04.0', '']
    steps:
      - name: Check out pipeline code
        uses: actions/checkout@v2.4.0

      - name: Check if Dockerfile or Conda environment changed
        uses: technote-space/get-diff-action@v4
        with:
          FILES: |
            Dockerfile
            environment.yml

      - name: Build new docker image
        if: env.MATCHED_FILES
        run: docker build --no-cache . -t barryd237/test:dev

      - name: Pull docker image
        if: ${{ !env.MATCHED_FILES }}
        run: |
          docker pull barryd237/test:dev
          docker tag barryd237/test:dev barryd237/test:dev

      - name: Install Nextflow
        env:
          CAPSULE_LOG: none
        run: |
          wget https://github.com/nextflow-io/nextflow/releases/download/v21.04.1/
↪nextflow
          sudo chmod 777 ./nextflow
          sudo mv nextflow /usr/local/bin/

      - name: Run pipeline with test data
        run: |
          nextflow run ${GITHUB_WORKSPACE} -profile test,docker
```

In your `nexflow.config` file, add the following:

```
profiles {
    docker {
        docker.enabled = true
        singularity.enabled = false
        podman.enabled = false
        shifter.enabled = false
        charliecloud.enabled = false
        docker.runOptions = '-u \$(id -u):\$(id -g)'
    }
    test { includeConfig 'conf/test.config' }
}
```

In your `conf/test.config` file, add the following:

```
// overwrite the -B bind path we used for singularity
// Docker will fail trying to use it
process{
  containerOptions = null
}
```

Add, commit and push the changes and cross your fingers!

CHAPTER

TWENTY

INTRO TO BASH

20.1 bashrc

A good place to start is your `.bashrc` file, which acts as a set of 'start up' instructions whenever you open your terminal. The `.bashrc` file is located in `~/.bashrc` i.e your home directory. You may never have noticed it because it is a hidden file (any file with a dot prefix is hidden). To show hidden files when running `ls`, use `ls -a`.

Commonly included items in a `.bashrc` file are:

1. `Aliases`: custom short hand commands that are aliases for longer, tricky to remember commands.
2. `Terminal colors`: not overly important..
3. `$PATH variables`: a list of directories that are searched for executables.

20.1.1 Aliases

Below are a set of aliases I use frequently which may save you some time. Copy the contents of the code block below to your `~/.bashrc` file and save the file. To initiate the changes, open a new terminal or run `source ~/.bashrc`.

```
# bash alias
alias l="ls -lhg --color=auto"
alias ls='ls --color=auto'
alias tarzip="tar -cvzf"
alias tarunzip="tar -xvzf"
alias vbrc="vi ~/.bashrc"
alias sbrc="source ~/.bashrc"
alias lugh="ssh bdigby@lugh.nuigalway.ie"
```

20.1.2 Inputrc

A very handy trick is the ability to scroll through your history based on a partial string match to a command previously run. You will need to create the file: `~/.inputrc`:

`~/.inputrc` :

```
#Page up/page down
"\e[B": history-search-forward
"\e[A": history-search-backward

$include /etc/inputrc
```

Now add the following line to your ~/.bashrc file:

```
#auto complete
export INPUTRC=$HOME/.inputrc
```

Source both files to initiate the changes:

```
source ~/.bashrc
bind -f ~/.inputrc
```

Test it out by cycling through your history with the arrow keys, and (for example) typing cd and then press the arrow keys to cycle thorugh all previous cd commands (as opposed to the most recent command).

20.2 $PATH

I will demonstrate the utility of the $PATH variable I showed you in the tutorial.

Start by making a new directory and navigate to that directory:

```
mkdir -p ~/foo/bar/qux
cd ~/foo/bar/qux
```

Create a file, we are going to pretend this is an executable like fastqc or bowtie2 - the principle is the exact same.

```
touch executable && chmod 777 executable
```

In the ~/foo/bar/qux directory, we are able to "run" executable by typing ./executable. You can type ./exec and hit TAB to autocomplete the command.

Navigate to your $HOME directory and "run" the executable by file. We need to provide either the relative or absolute path to the executable:

```
# relative path
foo/bar/qux/executable

# absolute path
/home/barry/foo/bar/qux/executable
```

Add the /home/barry/foo/bar/qux/ directory to the $PATH variable:

```
export PATH=$PATH:/home/barry/foo/bar/qux/
```

Now type execu and hit TAB to autocomplete the command. You should be able to access executable from anywhere on your system. To confirm this, type which executable to view where the executable is located.

To make this permanent, add export PATH=$PATH:/home/barry/foo/bar/qux/ to your ~/.bashrc file.

Note: This will allow your system to see **all** files in *foo/bar/qux/* and all subdirectories. For the sake of the demonstration I have only used one file.

20.3 Variable Expansion

When running a bioinformatics workflow, from a scripting perspective all we are doing is making sure that samples retain their correct names as they are passed to different file types (e.g `fastq` to `bam`).

You will need to have a concept of `basename` and variable expansion such that you can name samples correctly in an automated manner when scripting.

Note: please use the fastq files from Assignment one here

```
#!/usr/bin/env bash

    # place path to fastq files here (substitute your own)
    fastq_dir="/data/MA5112/week1/fastq"

    # we are reading R1 and R2 at once here (*{1,2}).
    for file in ${fastq_dir}/*{1,2}.fastq.gz; do

        # get the sample name (remove extension)
        # we will need this for naming outputs
        sample_name=$( basename $file .fastq.gz )

        # print sample name
        echo "File name without extension: $sample_name"

        # we still have _1 and _2 in the name for read 1 and 2 which messes up naming.
        # remove them before continuing
        base_name=$(basename $sample_name | cut -d'_' -f1,2)

        #print base name with R1 R2 (1 , 2) stripped:
        echo "File name without Read identifier: $base_name"

        # What if the process needs both R1 and R2? (e.g alignment)
        R1=${base_name}_1.fastq.gz
        R2=${base_name}_2.fastq.gz

        # sanity check below to see if R1 and R2 VAR are set properly:
        echo "Staging sample: $base_name"
        echo "Beginning to count lines in files..."
        lines_R1=$(zcat $fastq_dir/$R1 | wc -l)
        lines_R2=$(zcat $fastq_dir/$R2 | wc -l)
        echo "Done!"
        echo "$lines_R1 lines in $R1 and $lines_R2 lines in $R2"

        printf "\n\n"

        # make script pause for a sec to see output
        sleep 5

    done
```

Take your time going through this script. Personally, I would 'comment out' each line inside the for loop (add a # at

the beginning of the line) and then run the script, removing comments as you gain understanding.

To run the script, type `bash <scriptname>.sh` in your terminal.

CHAPTER

TWENTYONE

INDICES AND TABLES

- genindex
- modindex
- search