Introduction to Nextflow

COMP3550/BIOL3951,BIOL7941
Lourdes Peña-Castillo,PhD
Departments of Computer Science & Biology

What is Nextflow?

- A workflow management system
  - It supports:
    - Pipeline development
    - Use of external software
    - Intermediate files handling
- A domain specific language for the quick implementation of computational pipelines with complex data (designed for and by bioinformaticians)
- It requires familiarity with programming

Why to use Nextflow?

- For computational reproducibility
  - Integration with software repositories (GitHub & BitBucket)
  - Multi-scale containerization (Docker & Singularity)
- Asynchronous execution
  - Each operation (workflow task) is isolated in its own execution context
  - Outputs from one operation are directed to other operations through *channels*
- Processing follows the natural flow of data analysis
- Resumption of Pipelines

Programming Paradigm

- Dataflow programming paradigm
  - The program is modelled as a directed graph of the data flowing between operations
  - Inputs and outputs are explicitly defined to connect operations
  - Operations are automatically started once data are received through input channels

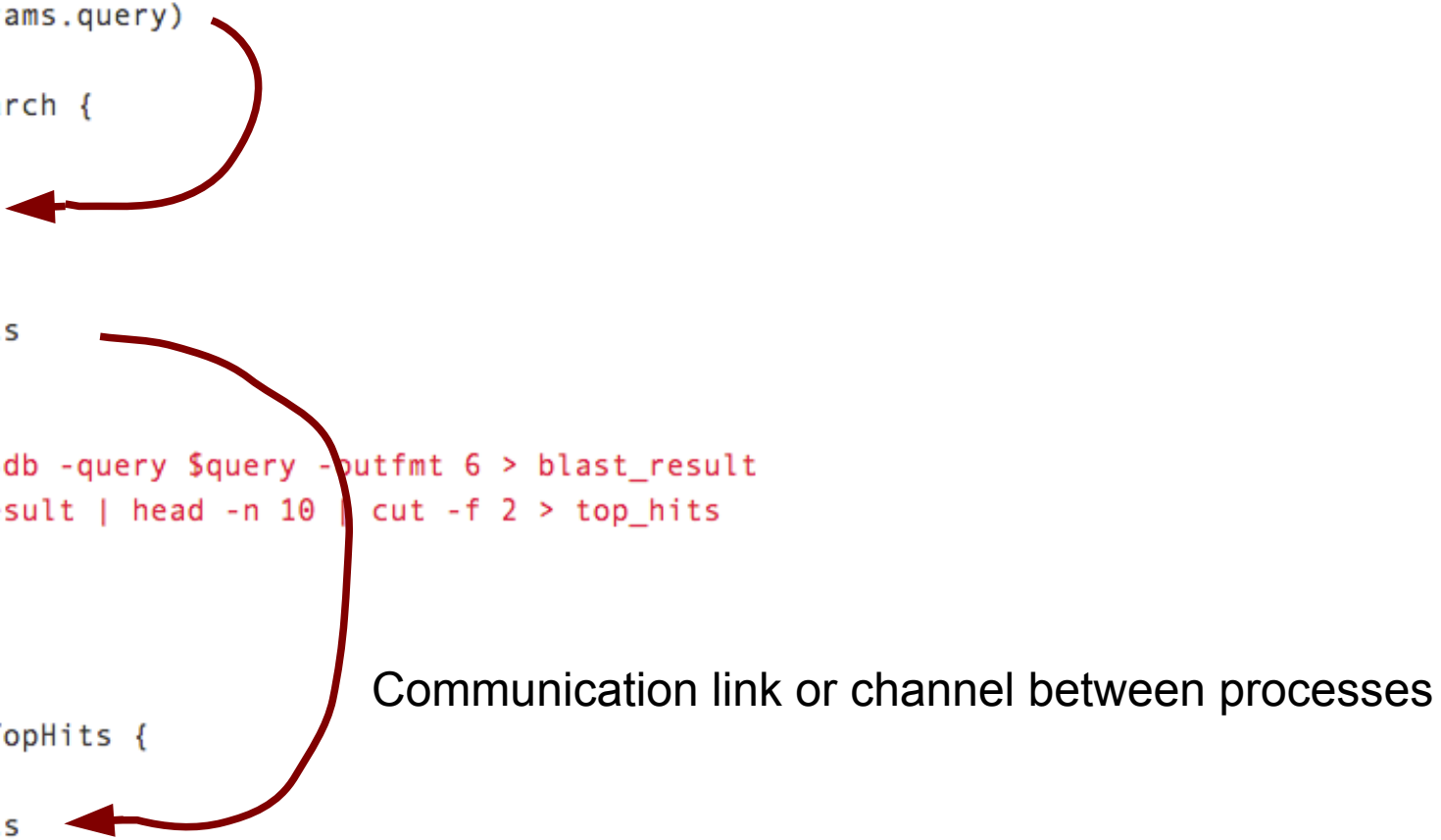
Nextflow Basic Concepts

- A *pipeline* is made by putting together several different *processes*
- A *process* represents an operation or task
- Each process can be implemented (written) in any scripting language that can be executed by the Linux platform (Linux shell, Perl, Python, R, etc.)
- Processes communicate via *channels (dataflows)*
- Each process needs to define at least one input channel and one output channel
- Execution order is established by communication channels (data dependencies)

Architecture of a Nextflow Program

```
params.query = "$HOME/sample.fa"
params.db = "$HOME/tools/blast-db/pdb/pdb"

db = file(params.db)
query = file(params.query)

process blastSearch {
    input:
    file query

    output:
    file top_hits

    """
    blastp -db $db -query $query -outfmt 6 > blast_result
    cat blast_result | head -n 10 | cut -f 2 > top_hits
    """
}


process extractTopHits {
    input:
    file top_hits

    output:
    file sequences

    """
    blastdbcmd -db ${db} -entry_batch $top_hits > sequences
    """
}
```

```
params.query = "$HOME/sample.fa"
params.db = "$HOME/tools/blast-db/pdb/pdb"

db = file(params.db)
query = file(params.query)

process blastSearch {
    input:
    file query

    output:
    file top_hits

    """
    blastp -db $db -query $query -outfmt 6 > blast_result
    cat blast_result | head -n 10 | cut -f 2 > top_hits
    """
}


process extractTopHits {
    input:
    file top_hits

    output:
    file sequences

    """
    blastdbcmd -db ${db} -entry_batch $top_hits > sequences
    """
}
```



Defining a Process

- A process has three main parts:
  - input definition
  - output definition
  - script
- Single-line comments are indicated with //
- Multi-line comments are given between:

  /*

  comment here

  */

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

process processName {
input:
    //Here inputs are defined
output:
    //Here outputs are defined
script:
//Commands to run are given here
//Between triple quotation marks
"""

"""
}
```

Defining a Process

- A process has three main parts:
  - input definition
  - output definition
  - script

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

greetings = Channel.from ('hello', 'hola', 'ciao', 'bonjour')

process sayHelloWorld {
input:
    val x from greetings
output:
    file 'HelloWorld.txt' into hellos
script:
"""
echo "$x world!" > HelloWorld.txt
"""
}
```

Go ahead, write your first nextflow script.
Copy the script below into a text editor and save it as:
helloWorld.nf

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

greetings = Channel.from ('hello', 'hola', 'ciao',
'bonjour')

process sayHelloWorld {
input:
    val x from greetings
output:
    file 'HelloWorld.txt' into hellos
script:
"""
echo "$x world!" > HelloWorld.txt
"""
}
```

Run it by typing in the terminal the command below

(make sure you are in directory containing the script helloWorld.nf):

```
nextflow run helloWorld.nf
```

What's going on?
One process executed per input value

```
orizaba:Lab2_Nextflow lourdes$ nextflow run helloWorld.nf
N E X T F L O W  ~  version 0.31.1
Launching `helloWorld.nf` [astonishing_austin] - revision: 4c8b1e72a0
[warm up] executor > local
[ad/5b9253] Submitted process > sayHelloWorld (3)
[88/1ec5cd] Submitted process > sayHelloWorld (2)
[9b/38f7ff] Submitted process > sayHelloWorld (1)
[9d/5e2687] Submitted process > sayHelloWorld (4)
orizaba:Lab2_Nextflow lourdes$
```

Why we don't have a file “HelloWorld.txt” in our directory?

- During execution Nextflow creates a directory called `work`
- Files created by each process are stored in the subdirectory corresponding to each process
- Let's take a look at one of those subdirectories
  - To re-run a process:

    ```
    bash .command.run
    ```

  - To see the actual command:

    ```
    cat .command.sh
    ```

Why we don't have a file “HelloWorld.txt” in our directory?

- We need to explicitly collect all the files created by the individual processes into a single file
- Add the last line to your script and run it again using:

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

greetings = Channel.from ('hello', 'hola', 'ciao',
'bonjour')

process sayHelloWorld {
input:
    val x from greetings
output:
    file 'HelloWorld.txt' into hellos
script:
"""
echo "$x world!" > HelloWorld.txt
"""
}

hellos
.collectFile(name: file("allHelloWorld.txt"))
```

```
nextflow run helloWorld.nf -resume
```

What's going on? Variable Interpolation

- Nextflow will parse the strings in the script section and replace placeholders such as $x with their corresponding values before sending for execution.
- Double-quoted strings support variable interpolation, while single-quoted strings do not.

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

greetings = Channel.from ('hello', 'hola',
'ciao', 'bonjour')

process sayHelloWorld {
input:
    val x from greetings
output:
    file 'HelloWorld.txt' into hellos
script:
"""
echo "$x world!" > HelloWorld.txt
"""
}
```

Notes

- A channel can only be used once as process output and once as process input.
- Special characters such as $, \n, \t need to be escaped:
  - write a \ before the special character

Let's write other Nextflow pipelines.
See the instructions posted in D2L