Workflows

Autors: Guerau Fernandez Isern, Joan Colomer Vila, Maria Begoña Hernández Olasagarre, Enrique Blanco García
L'encàrrec i la creació d'aquest recurs d'aprenentatge UOC han estat coordinats pel professor: Josep Jorba Esteve
PID_00298304
Primera edició: setembre 2023

Introducció

Els *pipelines* tradicionals estan molt lligats a les infraestructures de computació locals on s'executen. No tenen la capacitat de resumir un procés que s'hagi aturat, tenen poca documentació, no compten amb una traçabilitat dels paràmetres i versions de paquets utilitzats i requereixen instal·lació manual, la qual cosa impedeix una fàcil distribució d'aquest. Per poder solucionar aquests inconvenients s'han creat els Workflow Managers. Aquests permeten la utilització de *pipelines* d'anàlisis complexes en diferents entorns de computació assegurant la màxima reproductibilitat dels processos executats.

Diversos Workflow Managers s'han desenvolupat específicament per als camps de recerca i salut integrant entorns, contenidors i computació al núvol.

Hi ha cinc característiques que fan als Workflow Managers eines de gran utilitat:

1. Reproductibilitat. La utilització d'entorns i contenidors assegura una reproductibilitat apropiada dels processos executats.

2. Portabilitat. És un dels grans avantatges de la utilització de Workflow Managers, ja que crea els fluxos de treball necessaris per poder-se exportar a qualsevol entorn computacional. Molts d'ells permeten la fàcil migració a diferents entorns, inclosos els d'alta computació i serveis al núvol. Encara més, és possible la interacció directa amb orquestradors com Kubernetes o DockerSwarm.

3. Escalabilitat. Ser capaç de manejar i analitzar dades amb una complexitat creixent és cada vegada més comú. En aquest sentit, hi ha dos aspectes que s'han de tenir en compte: el maneig eficient dels recursos i ser capaç d'utilitzar dades més complexes i de major grandària. La majoria de Workflow Managers implementen la paral·lelització en diversos passos, sigui mitjançant gestor de cues o *scheduling* estàtic o adaptatiu. La paral·lelització es pot produir en l'àmbit de dades, processos o *pipelines*. Una assignació dinàmica dels recursos permet que els processos més intensius no es vegin afectats respecte dels que no en requereixen tants. Aquest balanceig minimitza colls d'ampolla i redueix els temps de computació. Els recursos es poden assignar específicament per a cada pas del flux de treball.

4. Robustesa. Molts *pipelines* requereixen processos complexos i de llarga durada. En el possible esdeveniment de la interrupció del *pipeline* en algun procés a causa d'un error, sigui programàtic o per l'absència d'un *input* requerit, els Workflow Managers són capaços de resumir el procés des del lloc on hi va haver l'últim pas correcte, resultant en l'estalvi en la utilització de recursos i temps. Aquest procés s'aconsegueix mitjançant la producció d'arxius i resultats intermedis, essent comparats amb els resultats esperats. Aquest procés genera un augment en les necessitats d'emmagatzematge, però comporta un avantatge substancial en el cas de tenir la necessitat d'una reentrada en el *pipeline.*

5. Modularitat. La compartimentació dels processos permet un gran dinamisme en l'actualització de certs passos del procés, així com de la introducció de punts de control per a cada etapa. La modularitat també permet la reutilització d'un procés en diversos *pipelines* simultàniament.

Finalment, cal indicar que alguns Workflow Managers també tenen recursos per augmentar la seguretat en l'execució dels processos, com la validació de l'origen de les dades o utilitzar autenticació d'usuaris.

Objectius

1. Oferir una visió general dels Workflow Managers.

2. Introduir els lectors a Nextflow.

1. Diferents tipus de *workflows*

La implementació d'un *pipeline* en un Workflow Manager requereix la definició precisa d'on s'extreuen les dades inicials (*input*), quin és el flux entre les diferents eines i quin és el resultat final (*output*). Per poder estructurar els diferents paràmetres s'utilitzen llenguatges de domini específic (DSL). La utilització de DSL augmenta la portabilitat i l'escalabilitat. Nextflow i Snakemake són dos dels exemples més populars de *workflows* que fan servir DSL propis en l'àmbit de la bioinformàtica. La diversitat de DSL amb la seva pròpia estructuració i nomenclatura propicia la reducció d'interoperabilitat entre els diferents *workflows*. Per mitigar aquesta disparitat s'han creat unes especificacions que afegeixen un nivell més gran d'abstracció, permetent un marc comú entre els diferents *workflows*. Exemples d'especificacions per a *workflows* són el Common Workflow Language (CWL) i Workflow Description Language (WDL). CWL prioritza la portabilitat i la reproductibilitat, mentre que WDL té com a principal objectiu reduir la corba d'aprenentatge mitjançant un llenguatge més comprensible. CWL defineix els *pipelines* usant fitxers YAML, mentre que WDL utilitza els seus propis fitxers descriptius. Alguns Workflow Managers, com *cwltool* o Cromwell, s'han creat basant-se en aquestes especificacions, mentre que d'altres com Snakemake implementen opcions d'exportació a aquestes especificacions.

La utilització de Workflow Managers per a la creació de *pipelines* bioinformàtics està creixent en popularitat. Diferents *workflows* estan disponibles per a la comunitat variant en la seva flexibilitat i facilitat d'ús (https://github.com/pditommaso/awesome-pipeline). Mentre que alguns *workflows* necessiten un coneixement avançat de programació, d'altres com Galaxy, KNIME o BioWorkflow implementen una interfície gràfica per facilitar la seva utilització. Tot i que el desenvolupament de sistemes de *workflow* es va iniciar a principis dels anys noranta, ha estat la capacitat de córrer *pipeline*s en clústers de computació o al núvol el que ha facilitat el seu ús més generalitzat.

2. Nextflow

2.1. Introducció

Com a exemple de Workflow Manager en aquesta secció utilitzarem un dels programes més estesos amb una comunitat creixent i amb més suport, com és Nextflow.

En primer lloc, hem d'instal·lar Nextflow seguint les instruccions directament de la seva pàgina web (https://www.nextflow.io/).

Nextflow combina tres elements: un entorn d'execució, un programa específic per llançar altres programes i un DSL específic. L'estratègia que implementa Nextflow en la creació de *pipelines* és la de crear mòduls per a cada pas en el *pipeline* i vehicular-los i interconnectar-los a través de canals. Una de les característiques més importants de Nextflow és la reutilització d'aquests mòduls per a diferents estadis d'un mateix *pipeline* o entre diferents *pipelines*. L'execució dels diferents mòduls no és lineal, sinó que depèn de la disponibilitat dels *inputs* per executar-se. Si dos mòduls independents requereixen els mateixos *inputs* o *inputs* independents es poden executar simultàniament per després poder convergir en un altre posteriorment si es requereix.

2. Nextflow

2.2. DSL2

Nextflow utilitza com a DSL una extensió del llenguatge de programació Groovy. A partir de la versió 20.07.1 de Nextflow es va actualitzar la sintaxi del DSL original creant DSL2. Al llarg d'aquesta secció farem ús d'aquesta nova implementació, DSL2. Aquesta actualització, entre moltes altres característiques, permet la utilització de més d'un *script* per definir el *workflow*, a diferència de DSL1, en el qual s'acumulava tot en un únic *script*. En el codi de Nextflow s'ha d'especificar la utilització de DSL2, ja que per defecte fa servir DSL1. Per a això s'indicarà a l'inici de l'*script*:

```
nextflow.enable.dsl=2
```

2. Nextflow

2.3. Estructura de *workflows*

Els *workflows* creats per a Nextflow contenen tres parts diferenciades: processos, canals i *workflows*. Un procés executa una tasca. Cada procés és independent de l'altre i pot tenir més d'un canal d'entrada i de sortida. Un canal és un sistema de cues asíncron que permet el flux d'informació entre processos (figura 1). Per ajuntar els diferents processos i el seu flux d'execució (canals) es resumeix en un apartat específic en l'*script* que es denomina *workflow*.

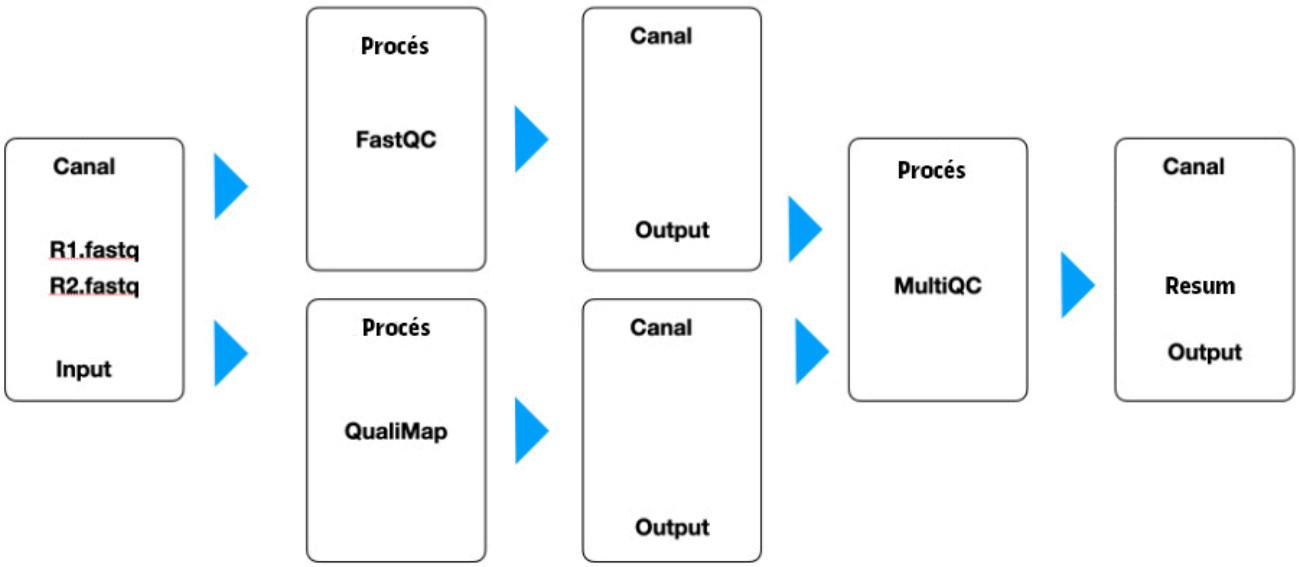

Figura 1. Exemple d'esquema de workflow a Nextflow.
Font: elaboració pròpia.

Nextflow diferencia les ordres que s'executaran dins d'un procés i qui serà l'encarregat d'executar-les. Això permet tenir un marc general que descrigui què es vol fer independentment de les eines que s'utilitzaran per executar-lo. Aquesta estructura permet llançar un procés en diferents entorns computacionals variant simplement un fitxer de configuració, el qual defineix els executors específics de l'entorn on un es trobi.

2. Nextflow

2.4. «Hello world»

Començarem per un *script* senzill (helloworld.nf) on imprimim una frase, per exemple «Hello world».

```
nextflow.enable.dsl=2


params.str = 'Hello world!'


process printStr {

 output:

    stdout


  """
echo '${params.str}'
  """

}


workflow {

 printStr | view ( )

}
```

- En primer lloc, a l'*script* `nf` li indicarem que utilitzem `DSL2`.
- Crearem un paràmetre que anomenarem «Hello world!». El nom del paràmetre està precedit per un punt i l'expressió `params`. En aquesta secció, com que es tracta d'un únic paràmetre, no cal crear un canal, ja que Nextflow assigna directament un canal *value*. Hi ha diferents tipus de canals, com veurem més endavant, i el canal *value* és el més simple.
- El següent segment defineix un procés al qual anomenarem printStr. La nomenclatura dels processos és *process* seguit del nom que li assignem. Els processos estan formats per tres parts: *input*, *output* i *script*. En aquest cas, li indiquem que l'*output* sigui l'estàndard, i en ser un valor simple tampoc cal que definim un canal específicament. Dins del procés definirem què volem fer. Imprimim el paràmetre str.
- Finalment, assignem l'última secció al flux del *workflow*.

Un cop creat l'*script* l'executarem:

```
$ nextflow run helloworld.nf
```

El resultat es mostra a la figura 2.

```
N E X T F L O W  ~  version 20.10.0
Launching `helloworld.nf` [boring_pare] - revision: 6452698d90
executor >  local (1)
[e7/6bcd02] process > printStr [100%] 1 of 1 ✔
Hello world!
```

Figura 2. Execució de l'*script* helloworld.nf.
Font: elaboració pròpia.

Com es pot observar, s'ha executat un procés `printStr` i se'n visualitza el resultat. Si voleu modificar un paràmetre, en aquest cas `str`, des de la línia d'ordres executarem:

```
$ nextflow run helloworld.nf --str 'Bye Bye World'
```

I obtindrem el nou resultat.

Ara li afegirem un segon procés: posar totes les lletres en majúscula. Per això afegirem el procés *allToUpper*.

```
nextflow.enable.dsl=2

params.str = 'Hello world!'

process printStr {

  output:

   path 'test.txt'


  """
echo '${params.str}' > test.txt
  """

}

process allToUpper {

 input:

   path x

  output:

    stdout

  """
cat $x | tr '[a-z]' '[A-Z]'
  """
```

```
}

workflow {

 printStr | allToUpper | view ( )

}
```

I el resultat el visualitzem a la figura 3.

```
N E X T F L O W  ~  version 20.10.0
Launching `helloworld.nf` [small_venter] - revision: 628d37e1e8
executor >  local (2)
[a1/5d17b2] process > printStr   [100%] 1 of 1 ✔
[a9/2c5e3e] process > allToUpper [100%] 1 of 1 ✔
HELLO WORLD!
```

Figura 3. *Output* de dos processos a Nextflow.
Font: elaboració pròpia.

Com podeu comprovar a la secció *workflow*, primer executem `printStr`, posteriorment `allToUpper` i finalment el visualitzem mitjançant `view( ). view` és un operador que veurem més endavant.

Seguidament substituïm l'ordre `allToUpper`:

```
cat $x | tr '[a-z]' '[A-Z]'
```

per:

```
rev $x
```

i tornem a executar l'*script*. En aquest cas, volem imprimir «Hello world!» al revés. Com que el primer pas ja l'havíem llançat anteriorment, podem resumir l'execució mitjançant l'opció *resume*:

```
$ nextflow run helloworld.nf -resume
```

i ens generarà *l'output* de la figura 4.

```
N E X T F L O W  ~  version 20.10.0
Launching `helloworld.nf` [zen_rosalind] - revision: 00b2557d9b
executor >  local (1)
[f9/a37153] process > printStr   [100%] 1 of 1, cached: 1 ✔
[b2/de6b1e] process > allToUpper [100%] 1 of 1 ✔
!dlrow olleH
```

Figura 4. Resumir l'execució de l'*script* helloworld.nf.
Font: elaboració pròpia.

Com podeu veure, el primer pas de *l'script* no ha estat calculat de nou, sinó que s'ha utilitzat el procés ja generat anteriorment (*cached*). D'aquesta manera, si en algun pas del *pipeline* hi ha hagut algun error que ha aturat el procés, es pot resumir prèvia esmena del problema.

2. Nextflow

2.5. Canals a Nextflow

Com s'ha comentat anteriorment, els canals dirigeixen el flux d'informació a través dels diferents processos. Per crear explícitament un canal s'ha d'utilitzar un mètode de Channel Factory proporcionat per Nextflow.

A Nextflow hi ha dos tipus de canals:

- *Queue channels*: són canals asincrònics, unidireccionals i FIFO (*first-in-first-out*).

Alguns exemples són:

```
- of:

      ch = Channel.of(1, 3, 5, 7)


- fromPath:

      file_ch = Channel.fromPath('test/*.txt')


- fromFilePairs:

      fastq = Channel.fromFilePairs('/my/data/SRR*_{1,2}.fastq')
```

- *Value channels* (*singleton channel*): són molt similars als *queue channels*, però només admeten un valor.

```
          value:
  pi = Channel.value('3.1416')
```

Per entendre una mica millor el funcionament dels canals crearem un *script* anomenat *orden.nf* amb un canal `value` i un altre `of` i imprimirem el resultat.

```
nextflow.enable.dsl=2
pi = Channel.value(3.1416)
queue_ch = Channel.of( 1, 3, 5, 7 )

process ordenE {
    input:
      val x
      val y
    output:
      stdout
"""
     echo $x $y
"""
}
workflow{
ordenE(pi,queue_ch) | view( )
```

```
}
```

Com podem observar, a l'apartat *workflow* estem especificant els *inputs* del procés `ordenE`. En el resultat d'aquest *script* (figura 5), l'ordre d'aparició dels valors del canal `of` no és el mateix que li hem indicat. Si repetim l'execució, segurament ens sortirà un resultat diferent. Es produeixen quatre processos de forma no correlativa.

```
N E X T F L O W  ~  version 20.10.0
Launching `orden.nf` [irreverent_bardeen] - revision: 861a33a302
executor >  local (4)
[fc/6d400c] process > ordenE (4) [100%] 4 of 4 ✔
3.1416 5

3.1416 3

3.1416 1

3.1416 7
```

Figura 5. *Output* de l'*script* `orden.nf`.
Font: elaboració pròpia.

Nextflow també és capaç de generar mètriques i reports mitjançant opcions introduïdes via terminal.

Uns exemples serien:

- *with-report*: crea un informe d'execució.
- *with-trace*: generarà un arxiu on s'indiquin paràmetres de l'execució, com memòria i *cpus* utilitzades, inici d'execució...
- *with-timeline*: permet identificar els colls d'ampolla indicant el temps que consumeix cada procés.

```
$ nextflow run orden.nf -with-timeline<p></p>
```

2. Nextflow

2.6. Operadors

A la secció anterior hem vist com moure canals per dirigir les dades entre els processos. Per poder modificar el contingut o el comportament d'un canal, Nexflow ha creat el que es denominen *operadors*. En els *scripts* anteriors hem vist l'operador *view*, però podem trobar operadors de filtratge, combinació o d'operacions matemàtiques entre molts altres. En aquesta secció en veurem alguns exemples.

Els operadors es poden introduir mitjançant un *pipe* (|), com hem vist anteriorment, o precedits per un punt. Així:

```
workflow{

ordenE(pi,queue_ch) | view( )

}
```

és anàleg a:

```
workflow{

ordenE(pi,queue_ch).view( )

}
```

A partir de l'*script* anterior eliminarem el canal *valor pi* i ens quedarem amb un exemple més senzill amb el canal *of queue_ch*. Com veureu a continuació hi afegim la notació *.view*, i dins d'aquest operador hi introduïm un prefix, *chr*, i un valor, *$it*, entre {}. Aquests parèntesis defineixen un bloc de codi que va al costat i utilitza la nomenclatura de *goovy* (*it*, d'*ítem*) per definir els paràmetres.

```
nextflow.enable.dsl=2

queue_ch = Channel.of( 1, 3, 5, 7 ).view({"chr$it"})


process ordenE {

    input:

      val x


    output:

      stdout


"""

    echo $x

"""

}
```

```
workflow{

ordenE(queue_ch)

}
```

En aquest cas l'*output* es visualitza a la figura 6.

```
N E X T F L O W  ~  version 20.10.0
Launching `orden2.nf` [jovial_brahmagupta] - revision: 3e9c59f53f
executor >  local (4)
[3b/f3d7ba] process > ordenE (3) [100%] 4 of 4 ✔
chr7

chr3

chr1

chr5
```

Figura 6. Resultat de l'operador *view* amb prefix.
Font: elaboració pròpia.

Podem introduir un filtre al canal per mostrar únicament els valors superiors a 4:

```
queue_ch = Channel.of( 1, 3, 5, 7 ).filter { it > 4 }.view({"chr$it"})
```

També podem combinar canals utilitzant l'operador *mix*:

```
ch1 = channel.of( 1..22 )

ch2 = channel.of( 'X','Y' )

ch3 = channel.of( 'MT' )


queue_ch  = ch1.mix(ch2,ch3).view({"chr$it"})
```

i fer operacions com comptar el nombre d'elements:

```
queue_ch  = ch1.mix(ch2,ch3).count().view()
```

Les possibilitats dels operadors proporcionen una versatilitat molt gran de poder manipular les dades que volem analitzar.

2. Nextflow

2.7. Configuració

Finalment tractarem els arxius de configuració de Nextflow. Aquests arxius són rellevants per poder migrar els *scripts* entre entorns de computació i per al control dels recursos a utilitzar.

Podem trobar diversos arxius de configuració i alguns poden entrar en conflicte entre ells. Per això Nextflow té una priorització:

1. Paràmetres especificats a la línia d'ordres.
2. Paràmetres procedents de l'arxiu especificat mitjançant l'opció -params-file.
3. Arxiu de configuració especificat mitjançant l'opció -c my_config.
4. Arxiu de configuració nexflow.config en el directori de treball.
5. Arxiu de configuració nexflow.config en el directori del projecte de *workflow*.
6. Paràmetres definits en el mateix *script* de Nextflow.

Si un paràmetre es troba en més d'una d'aquestes fonts, Nextflow fa servir la primera font com a referència i no fa servir les subsegüents.

L'arxiu consta de parelles nom = valor

```
process.memory = '10G'
```

Un arxiu de configuració es pot incloure en un altre. Per exemple:

```
includeConfig 'path/foo.config'
```

La instrucció `includeConfig` busca l'arxiu de paràmetres i els inclou com a propis.

Els paràmetres de configuració es poden especificar en el que s'anomenen *scopes*. Són paràmetres que afecten específicament un tipus de configuració. Hi ha diversos *scopes* de configuració, els més habituals dels quals són:

- *aws*: Amazon S3.
- *conda*: entorns Conda.
- *docker*: contenidors Docker.
- *k8s*: clúster de Kubernetes.

Per habilitar la utilització d'una imatge Docker, per exemple, es podria introduir al nextflow.config:

```
docker.enabled = true
```

o es podria especificar directament a la línia d'ordres:

```
nextflow run <script> -with-docker [imatge Docker]
```

També és possible especificar una imatge Docker per a un procés determinat:

```
process uno {

  container 'nombre_imatge_1'

```

```
    '''
    execució
    '''
}

process dos {
    container 'nombre_imatge_2'

    '''
    execució
    '''
}
```

2. Nextflow

2.8. *nf-core*

Com altres iniciatives que ja hem tractat anteriorment, Nextflow també té un repositori de *workflows* per poder ser executats. Aquest conjunt de *pipelines* s'agrupa en el projecte *nf-core* (https://nf-co.re/). Per poder treballar directament des de el terminal s'ha generat el paquet *nf-core tools* per poder gestionar els diferents *workflows* accessibles en *nf-core*. *nf-core tools* està escrit a Python i es pot instal·lar des de la següent adreça: https://nf-co.re/tools.

Per poder veure els diferents paquets simplement es pot fer un llistat:

```
$ nf-core list
```

Moltes vegades necessitem ser més específics en la nostra recerca i, per això, podem filtrar:

```
$ nf-core list rna
```

O fins i tot ordenar per estrelles:

```
nf-core list rna --sort stars
```

Cada repositori té la seva pròpia pàgina amb instruccions i documentació. La utilització d'aquests repositoris públics permet un aprenentatge més ràpid de les eines de Nextflow en el context específic de la bioinformàtica. Igualment, *nf-core* també proporciona un canal directe de preguntes mitjançant un compte a Slack (https://nf-co.re/join).

Resum

En aquest mòdul hem donat una breu pinzellada als Workflow Managers i en especial a Nextflow. Per augmentar el control dels entorns en els quals es computen les anàlisis és imprescindible utilitzar Workflow Managers, i, a poder ser, conjuntament amb entorns tipus Conda o contenidors, com hem vist en capítols anteriors. La utilització d'aquest tipus d'eines permet la portabilitat a qualsevol entorn de computació i la implementació de *pipelines* generats per altres grups d'una manera àgil i senzilla. La modularització dels processos permet que la seva actualització pugui ser dinàmica i que la introducció de codi generat per altres programadors no interfereixi en altres parts del procés. La versatilitat d'opcions dels Workflow Managers permet un control precís de tot el procés, de vegades en detriment de la seva corba d'aprenentatge. La diversitat de Workflow Managers que actualment tenim permet poder escollir l'ecosistema en què ens trobem més confortables, variant des d'entorns molt visuals tipus Galaxy a altres enterament programàtics. És important arribar a un compromís entre l'aprenentatge necessari per entendre el funcionament d'un *workflow* específic i les eines que ens permet controlar. Finalment, a l'hora de triar el vostre Workflow Manager s'ha de tenir en compte la potencial interoperabilitat. En entorns col·laboratius és imprescindible la reproductibilitat de processos i la compartició de codi per poder seguir els principis FAIR de les dades.

Activitats

1. Creeu un *script* Nextflow que transformi la paraula «HOLA» (en majúscules) tota en minúscules.

2. Modifiqueu a través de la crida de l'*script* la paraula «HOLA» per «CIAO».

3. Introduïu un nou procés que substitueixi *l'A* per un 4.

4. Creeu un *script* amb dos canals *of* i un procés que sumi els valors de cada canal i imprimeixi l'operació realitzada.

5. Busqueu i baixeu un *workflow* a través de *nf-core*.

Bibliografia

Bjørn Fjukstad i Lars Ailo Bongo (2017). A Review of Scalable Bioinformatics Pipelines. Data Science and Engineering, 2, p. 245-251. https://doi.org/10.1007/s41019-017-0047-z

Ed H. B. M. Gronenschild i altres (2012). The Effects of FreeSurfer Version, Workstation Type, and Macintosh Operating System Version on Anatomical Volume and Cortical Thickness Measurements. *PLoS ONE*, 7, e38234. Doi: 10.1371/journal.pone.0038234

Elise Larsonneur, Jonathan Mercier i Nicolas Wiart (2018). Evaluating Workflow Management Systems: A Bioinformatics Use Case. 2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), p. 2.773-2.775. Doi: 10.1109/BIBM.2018.8621141

Jeffrey M. Perkel (2019). Workflow Systems Turn Raw Data into Scientific Knowledge. *Nature*, 573, p. 149-150. Doi: 10.1038/d41586-019-02619-z

Jeremy Leipzig (2017). A Review of Bioinformatic Pipeline Frameworks. *Briefings in Bioinformatics*, 18, p. 530-536. Doi: 10.1093/bib/bbw020

Paolo Di Tommaso i altres (2017). Nextflow Enables Reproducible Computational Workflows. *Nature Biotechnology*, 35, p. 316-319. https://doi.org/10.1038/nbt.3820

Paolo Di Tommaso i altres (2015). The Impact of Docker Containers on the Performance of Genomic Pipelines. PeerJ, 3, e1273. https://doi.org/10.7717/peerj.1273

Philip Ewels i altres (2020). The nf-core Framework for Community-curated Bioinformatics Pipelines. *Nature Biotechnology*, 38, p. 276-278. Doi: 10.1038/s41587-020-0439-x

Taylor Reiter i altres (2021). Streamlining Data-intensive Biology with Workflow Systems. *Gigascience*, 10, giaa140. https://doi.org/10.1093/gigascience/giaa140