Crunchomics

Release 0.02

Han Rauwerda Wim de Leeuw

Jun 28, 2022

CONTENTS

CHAPTER

ONE

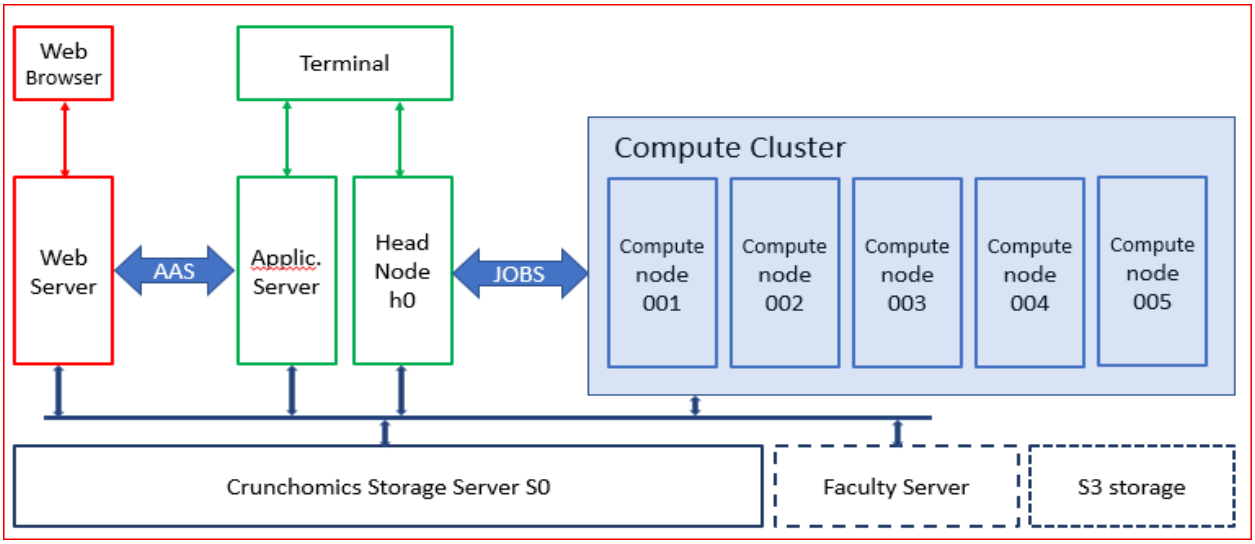
CRUNCHOMICS: THE GENOMICS COMPUTE ENVIRONMENT FOR SILS AND IBED

Cruchomics has 1 application server, a head node and 5 compute nodes. A storage system is mounted in which all users have a fast 25 GB SSD home directory and a 500 GB home directory. Volumes from the faculty server can be mounted on the system.

- CPU: AMD EPYC Rome 7302 32 cores, 64 threads, 3GHz
- Compute cluster: 160 cores, 320 threads
- Infiniband internal connection
- **Local storage compute cluster: 8TB (SSD/NVMe): /scratch**
  - /scratch is emptied after month of inactivity
- **Memory:**
  - Application server: 1024 GB
  - Head node and compute nodes: 512 GB
- **Storage: gross 504 TB operated in RAID-Z2: net approx. 220 TB.**
  - If one disk fails: no data loss
  - Snapshots taken: some protection against unintentional file deletions
  - No backups made!
- File systems are mounted on all nodes.
- OS: CentOS 7
- Help: w.c.deleeuw@uva.nl / j.rauwerda@uva.nl

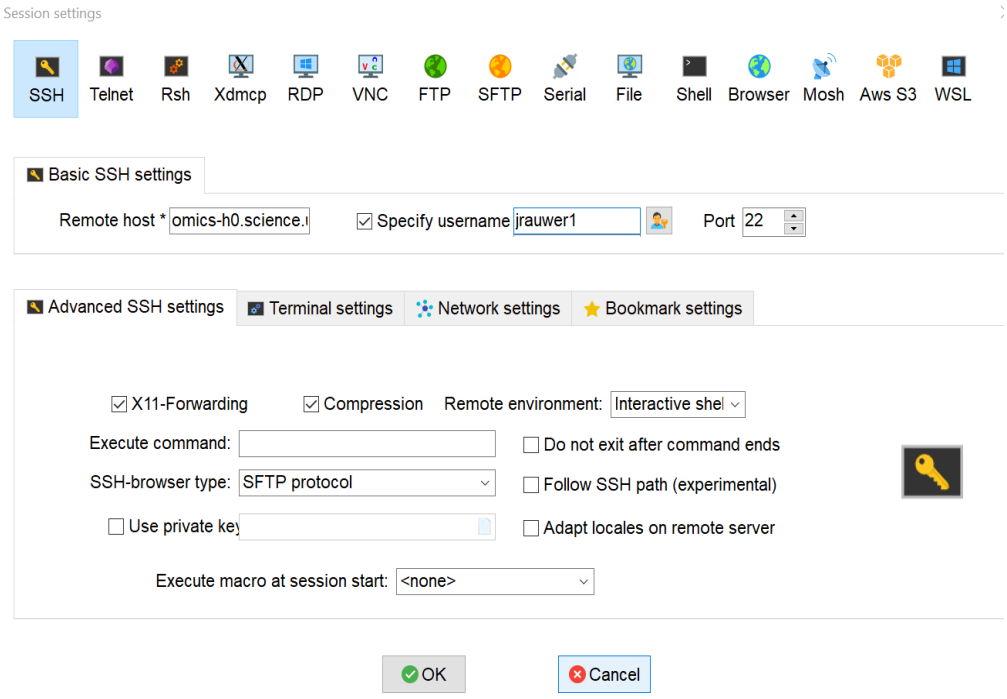
1.1 Log in on the Head Node

- Remark: if you are outside UvA, use VPN.
- **from your local (desktop or laptop) machine**
  - start puTTY or mobyxterm (windows)
  - start terminal program (mac)
  - start xterm or konsole (linux)



1.1.1 On Windows

- Connect via MobaXterm (includes X11 graphical output) or PuTTY.
- MobaXterm facilitates to remember passwords.
- **Via MobaXterm interface (address is omics-h0.science.uva.nl):**



1.1.1.1 Login Without a Password with RSA Keys

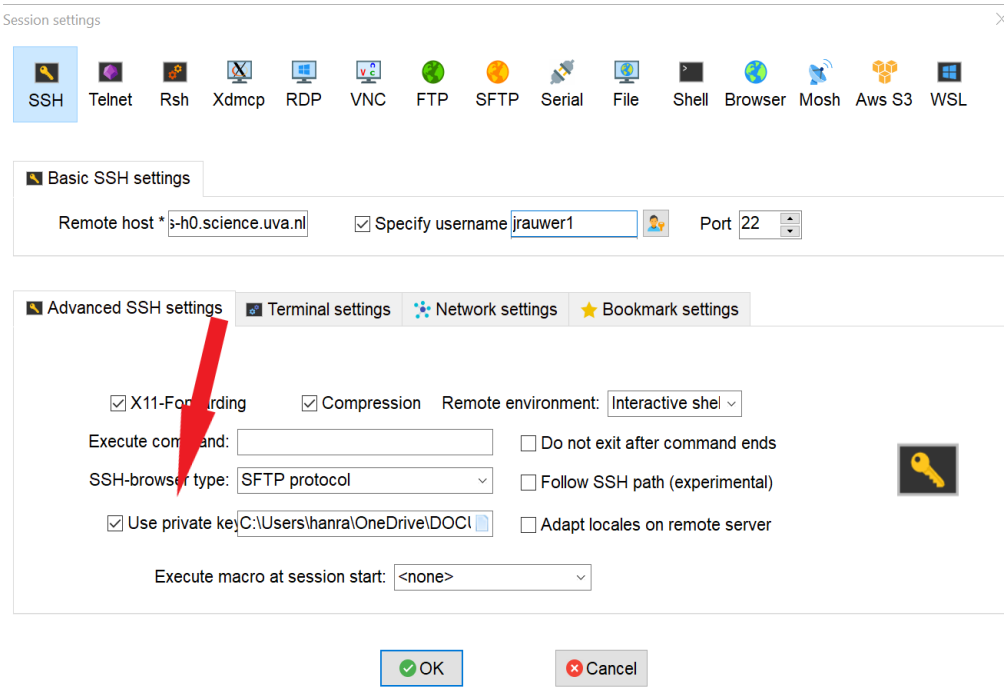
- An explanation on how to use SSH keys (existing ones or newly generated) can be found here: SSH keys with PuTTY
- **If you don't want to use the *Remember Password* mechanism in MobaXterm but still do not want to supply your password each time you login, you can configure MobaXterm also for use of SSH keys.**
  - Start a local terminal with MobaXterm and type:

```
if [  -f ~/.ssh/id_rsa.pub ] ;
        then  echo "Public key found.";
        else ssh-keygen -t rsa ;
fi
```

  - Type enter 3 times unless you want to use a passfrase for the key. Copy de public part key to the file .ssh/authorized_keys on omics-h0. Here you have to type the password a last time. In your local terminal type:

```
cat ~/.ssh/id_rsa.pub  | \
ssh uvanetid1@omics-h0.science.uva.nl \
'mkdir -p .ssh && cat >> .ssh/authorized_keys'
```

  - You can use the ssh key pair in MobaXterm by giving the location of the private key (it is in your local 'persistent' home directory, normally but depending on your windows version C:\Users\<YOUR_WINDOWS_LOGIN_NAME>\DOCUMENTS\MobaXterm\home\.ssh)



  -

  - Or: use the key generator mechanism in MobaXterm

1.1.2 On Linux and Mac

- Via a terminal:

```
#replace uvanetid1 with your uvanetid:
ssh -X uvanetid1@omics-h0.science.uva.nl
# -X for X11 graphical output
xclock
# if you see a clock xwindows works
```

1.1.2.1 Login Without a Password with RSA Keys

- Login using rsa keys: on the local system generate key if it doesn't exist:

```
if [  -f ~/.ssh/id_rsa.pub ] ; \
        then  echo "Public key found.";
        else ssh-keygen -t rsa ;
fi
```

- Type enter 3 times unless you want to use a passfrase for the key. Copy de public part key to the file .ssh/aurhorized_keys on omics-h0. Here you have to type the password a last time:

```
cat ~/.ssh/id_rsa.pub  | \
ssh uvanetid1@omics-h0.science.uva.nl \
'mkdir -p .ssh && cat >> .ssh/authorized_keys'
```

- Test if it works. Now you should be logged without being asked a password:

```
ssh uvanetid1@omics-h0.science.uva.nl
```

1.2 Preparing Your Account

1.2.1 Getting Your Environment Ready

- **You want to:**
  - use system wide installed software: `/zfs/omics/software/bin` is added to your path
  - have a python3 environment available + (genomics) specific packages such as *pandas* and *snakemake* installed: `madpy3` can be used as a command to activate the madpy3 environment
  - have a more informative prompt
  - have, in your ultrafast home directory a link to your 500 GB *personal* directory on the file storage.

A script to do the things mentioned above for you is here, type on your head node shell:

```
/zfs/omics/software/script/omics_install_script
```

This script will change the .bashrc and .bash_profile files such that the requirements mentioned above are fullfilled.

1.2.1.1 DIY

- If you want to make the adaptations yourself, here are some ideas:
- Create a shortcut (softlink) in your home directory (25GB) to your personal directory (500 GB):

```
ln -s /zfs/omics/personal/$USER ~/personal
```

- Update the PATH variable in your *.bashrc* file so you can use the software installed in `/zfs/omics/software`. This is software not installed by the package manager, such as R, bowtie2, samtools etc.:

```
export PATH=${PATH}:/zfs/omics/software/bin
```

- Some tools, such as snakemake, htseq, etc. need python3 and can be executed in a python virtual environment.
- Activate the virtual environment as follows:

```
which snakemake
#which: no snakemake
source /zfs/omics/software/v_envs/madpy3/bin/activate
which snakemake
#/zfs/omics/software/v_envs/madpy3/bin/snakemake
deactivate
```

1.3 Account

- **The quota for storage are:**
  - 25GB in your home directory
  - 500GB in /zfs/omics/personal/$USER
- The quota also apply to snapshots. Snapshots are made daily at 00.00.00 and kept for 2 weeks. This means that deleting files which are in a snapshot will not be available for another 2 weeks. It also means that if you accidentally remove a file it can be restored up to 2 weeks after removal.
- Data on Crunchomics is stored on multiple disks. Therefore, there is protection against disk failure. The data is not replicated. SILS users are encouraged to put their raw data on tape as soon as these are produced: SILS tape archive. Tape storage is duplicated.
- Help: w.c.deleeuw@uva.nl / j.rauwerda@uva.nl

1.4 The Crunchomics Application Server

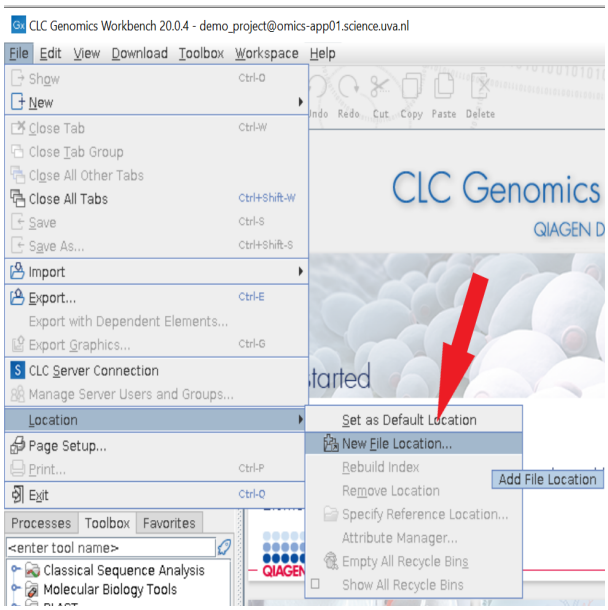
The address of the application server is *omics-app01.science.uva.nl* and you can log in to it in a similar fashion as you log in to the headnode. Your home and personal directory as well as other locations on the file system are also available on the application server.

1.4.1 CLC Genomics Workbench

- log in on the application server: address is *omics-app01.science.uva.nl* (**not** the headnode!!):

```
#in terminal type:
clcgenomicswb20
```

- **Remember, your home directory is 25G, so it is advisable to make your default location somewhere on /zfs/omics/personal/*uvanetid1*/**
  - **add a new folder in CLC genomics workbench (e.g. `/zfs/omics/personal/*uvanetid1*/CLC_personal`)**



  - **and make it the *default location***

1.4.2 Start RStudio

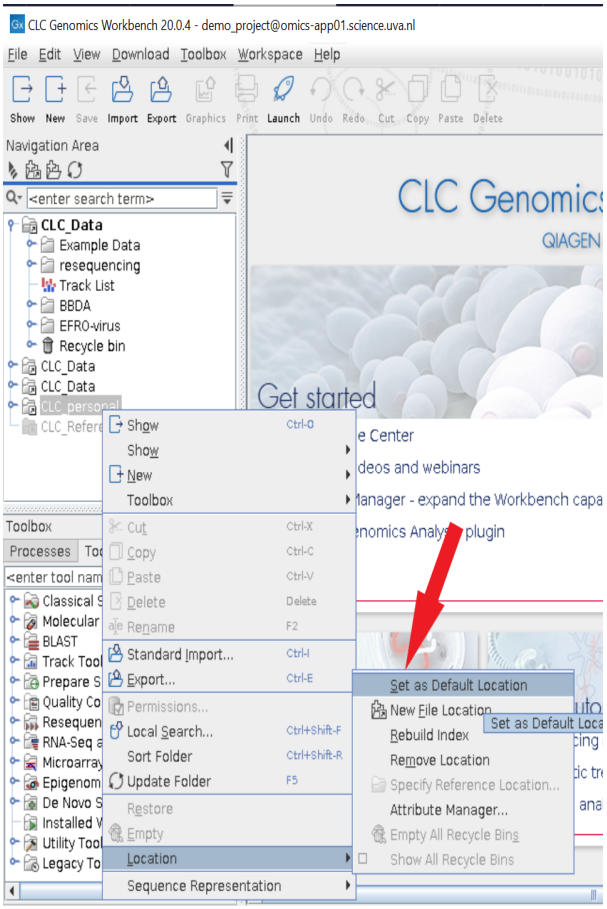
- Eventually, authentication to RStudio will be via Surfconnect. However, this Surfconnect access is not available yet. Until then you can login directly on the application server:
  - Login to RStudio on the application server with your UvAnetid and password at: *http://omics-app01.science.uva.nl/*.

1.5 The Crunchomics Compute Cluster

1.5.1 Interactive access

Interactive work on the head node *omics-h0.science.uva.nl* is possible but **only for small computational tasks. All other jobs should be executed via the SLURM queue!!**.

- **You can run interactive jobs on the head node**
  - small jobs


CLC Genomics Workbench 20.0.4 - demo_project@omics-app01.science.uva.nl
File Edit View Download Toolbox Workspace Help
Show New Save Import Export Graphics Print Launch Undo Redo Cut Copy Paste Delete
Navigation Area
<enter search term>
CLC_Data
Example Data
resequencing
Track List
BBDA
EFRO-virus
Recycle bin
CLC_Data
CLC_Data
CLC_personal
CLC Genomics
QIAGEN
Get started
Show Ctrl-O
Show
New
Toolbox
Cut Ctrl-X
Copy Ctrl-C
Paste Ctrl-V
Delete Delete
Rename F2
Standard Import... Ctrl-I
Export... Ctrl-E
Permissions...
Local Search... Ctrl+Shift-F
Sort Folder Ctrl+Shift-R
Update Folder F5
Restore
Empty
Location
Sequence Representation
Set as Default Location
New File Location
Rebuild Index
Remove Location
Specify Reference Location...
Attribute Manager...
Empty All Recycle Bins
Show All Recycle Bins
Toolbox
Processes
<enter tool nam
Classical S
Molecular
BLAST
Track Tool
Prepare S
Quality Co
Resequen
RNA-Seq a
Microarray
Epigenom
De Novo S
Installed V
Utility Tool
Legacy To

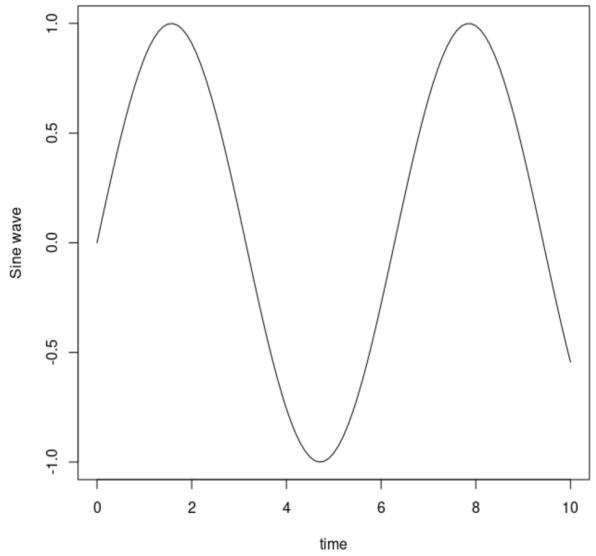
  - parameterization of a big job, pilot jobs
  - Only run jobs interactively when they take less than 12 hrs execution time on one cpu.

- **If you have a job for which a really large amount of memory is needed, such as a genome assembly, you can use the app server that has 1TB memory installed.**
  - Mark that the compute nodes have a 512 GB memory each, so many assembly jobs can better be put to the compute cluster.

1.5.1.1 Example of Interactive R Sessions

- connect to the head node (with X), start R
- a simple example in the kick-off meeting (in R):

```
t=seq(0,10,0.1)
y=sin(t)
plot(t,y,type="l", xlab="time", ylab="Sine wave")
```



Another example: * a file is read (the result of a BLAST alignment) and the ratio of the query length and subject length are plotted, together with x=y:

```
xclock&
wget https://surfdrive.surf.nl/files/index.php/s/9xIik2oVfjA8VVg/download \
--output-document blp.txt
```

- Next, in R

```
blp<-read.table("blp.txt", header=T)
head(blp)
x11(type="cairo")
library(ggplot2)
lbl<-paste("average ratio:",round(mean(blp$qlen/blp$slen),2))
lbl
```

(continues on next page)

(continued from previous page)

```
plot1<-ggplot(blp,aes(qlen,slen,colour=log(bitscore))) +
#scale_colour_gradientn(colours=rainbow(8)) +
scale_colour_gradient(low="lightgreen",high="darkgreen") +
geom_point(size=0.5, alpha=18/20, shape=20) +
coord_cartesian(ylim = c(0, 1000),xlim =  c(0,1000)) +
geom_abline(slope=1, intercept=0, alpha=0.8, colour="red") +
labs(x = "query length", y="subject length", title = "Blast result predicted proteins on␣
↪protein database") +
annotate("text", label = lbl, x = 800, y = 10)
plot1
#png("/zfs/omics/personal/jrauwer1/Crunchomics_intro/plot_blast_qlen_slen.png",width=700,
↪ height=700, type="cairo" )
#plot1
#dev.off()
```

1.6 Miniconda

- Conda is an open source package management system and environment management system. Conda allows you find and install packages in your own environment. You don't need administrator privileges.
- If you need a package that requires a different version of Python, you use conda as an environment manager. With just a few commands, you can set up a totally separate environment to run that different version of Python, while continuing to run your usual version of Python in your normal environment.

1.6.1 Installation

- basic install is 89M
- other packages will require additional space, so rather install it in `/zfs/omics/personal/<uvanetid1>` than in your `/home/<uvanetid1>`

1.6.1.1 Get miniconda

Download it

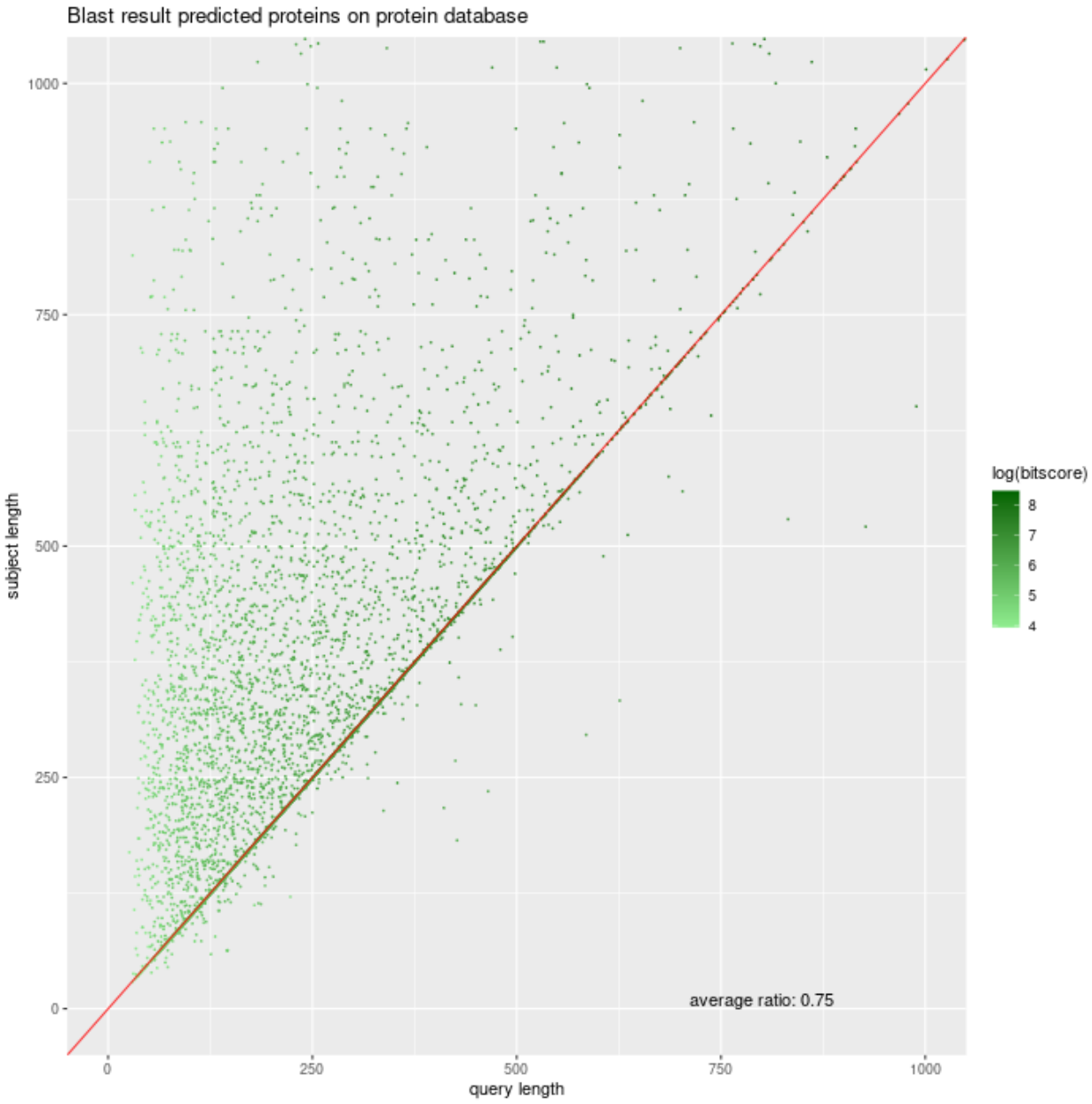
```
cd /zfs/omics/personal/${USER}
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
```

1.6.1.2 Install Miniconda

Run the install script (as you)

```
bash Miniconda3-latest-Linux-x86_64.sh
```

- Read the license information and answer yes….. After you answered yes, you will see a message like:
- Specify here `/zfs/omics/personal/${USER}/miniconda3` (or replace `\${USER}` with your uvanetid).
- When the installation has finished you are asked if you want to initialize conda. Answer *yes*:
- This finishes the installation:




```
Miniconda3 will now be installed into this location:
/home/jrauwer1/miniconda3

  - Press ENTER to confirm the location
  - Press CTRL-C to abort the installation
  - Or specify a different location below

[/home/jrauwer1/miniconda3] >>> /zfs/omics/personal/${USER}/miniconda3
```

```
  zlib               pkgs/main/linux-64::zlib-1.2.11-h7b


Preparing transaction: done
Executing transaction: done
installation finished.
Do you wish the installer to initialize Miniconda3
by running conda init? [yes|no]
[no] >>> yes
```

```
no change     /zfs/omics/personal/jrauwer1/miniconda3/lib/python3.8/site-packages/xontrib/conda.xsh
no change     /zfs/omics/personal/jrauwer1/miniconda3/etc/profile.d/conda.csh
modified      /home/jrauwer1/.bashrc

==> For changes to take effect, close and re-open your current shell. <==

If you'd prefer that conda's base environment not be activated on startup,
   set the auto_activate_base parameter to false:

conda config --set auto_activate_base false

Thank you for installing Miniconda3!
bash-4.2$
```

- After the installation, to have a shell with conda, leave your current one and start a new one or type:

```
source ~/.bashrc
```

- After you have done this, you will see *(base)* as a prefix in your shell prompt.
- So, each time you start a shell, conda will be activated, if you don't want that, type:

```
conda config --set auto_activate_base fals
```

- How much space does miniconda3 take?

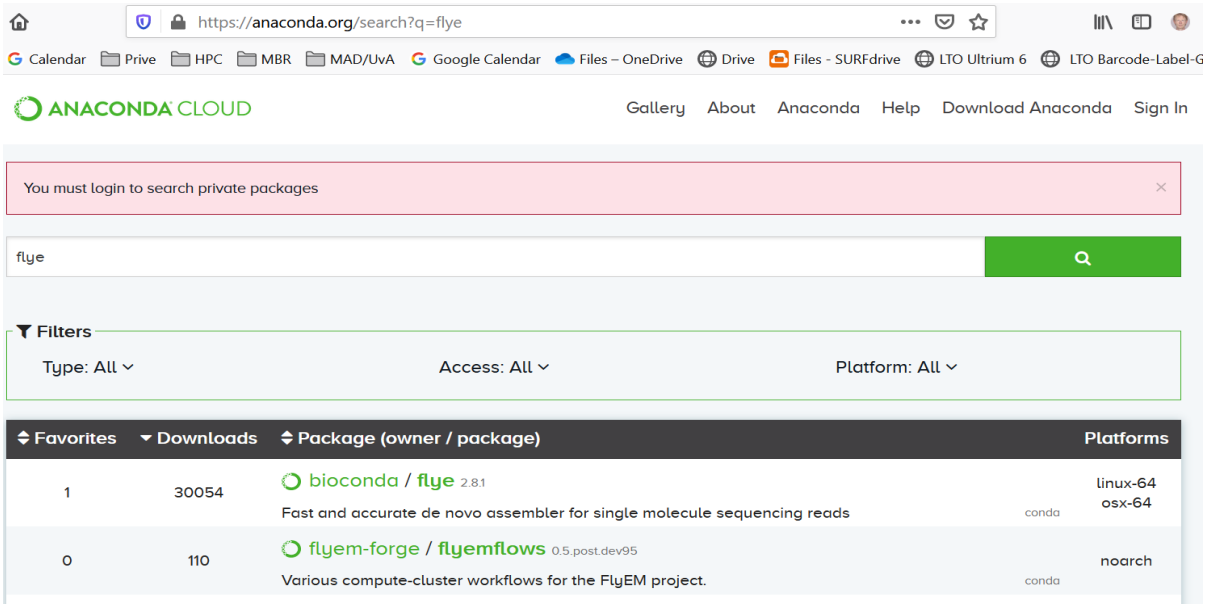
```
cd /zfs/personal/${USER}/
du -hs miniconda3/
#437M    miniconda3/
```

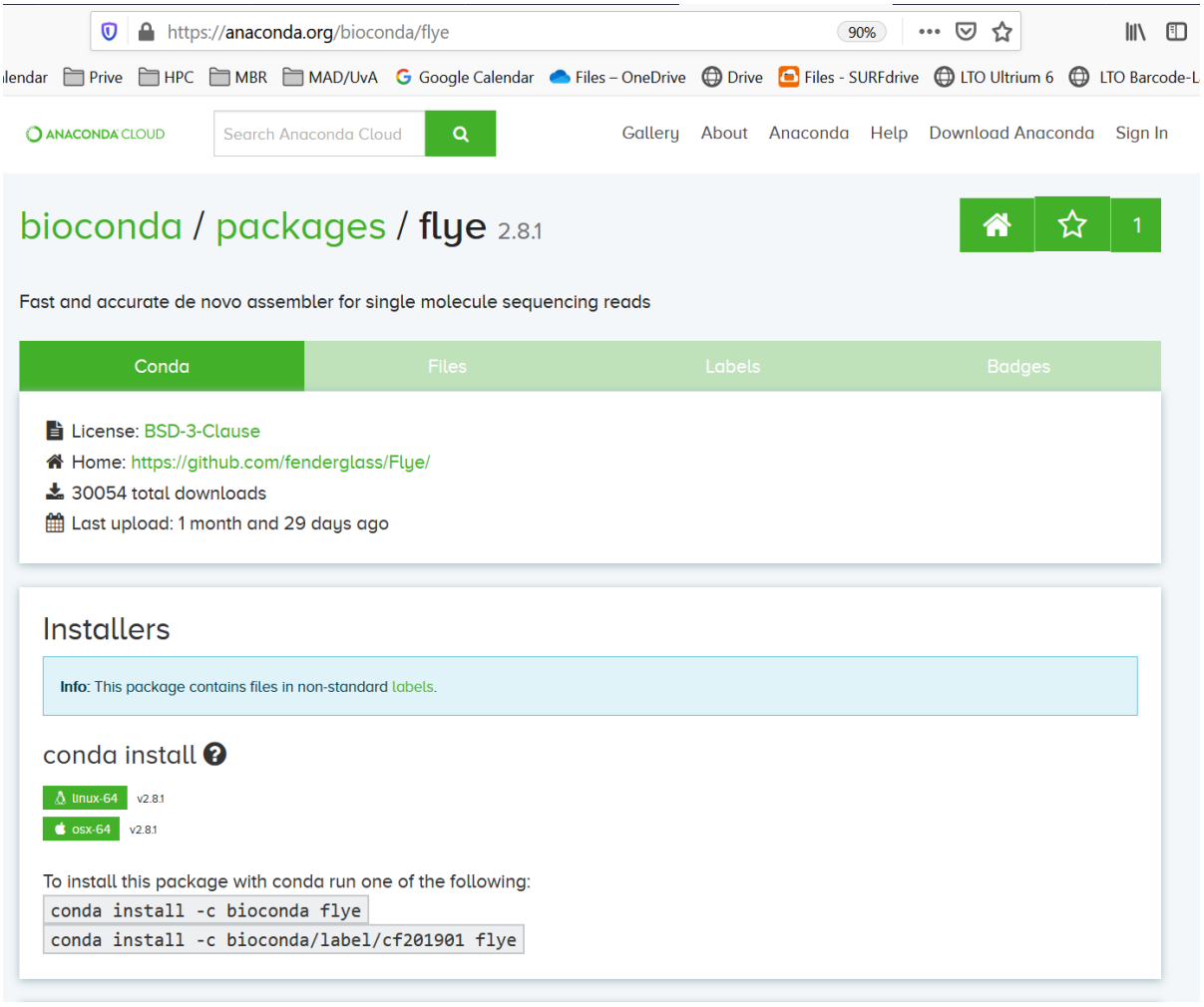
1.6.2 Installation of a Conda Package

- Suppose, you want to make a bacterial assembly with flye. With `which flye` you discover that flye is not installed on Crunchomics. Of course, you can ask the maintainers of Crunchomics to install flye. But you also could install flye as a conda package. You go to https://anaconda.org/ and search for *flye*
- There is a conda package available:
- When you click on flye you get a link with installation details:
- Suppose you don't want flye in your base environment. Therefore, you create a new environment that you call nptools. Install flye in the nptools evironment, deactivate base and activate nptools:

```
which flye  #indeed, no flye
conda create -n nptools
conda install -n nptools -c bioconda flye
conda deactivate
conda activate nptools
flye -h
#usage: flye (--pacbio-raw | --pacbio-corr | --pacbio-hifi | --nano-raw |
```

(continues on next page)


https://anaconda.org/search?q=flye
Calendar
Prive
HPC
MBR
MAD/UvA
Google Calendar
Files – OneDrive
Drive
Files - SURFdrive
LTO Ultrium 6
LTO Barcode-Label-G
ANACONDA CLOUD
Gallery
About
Anaconda
Help
Download Anaconda
Sign In
You must login to search private packages
flye
Filters
Type: All
Access: All
Platform: All
Favorites
Downloads
Package (owner / package)
Platforms
1
30054
bioconda / flye 2.8.1
Fast and accurate de novo assembler for single molecule sequencing reads
conda
linux-64
osx-64
0
110
flyem-forge / flyemflows 0.5.post.dev95
Various compute-cluster workflows for the FlyEM project.
conda
noarch


https://anaconda.org/bioconda/flye
90%
Prive
HPC
MBR
MAD/UvA
Google Calendar
Files – OneDrive
Drive
Files - SURFdrive
LTO Ultrium 6
LTO Barcode-L
ANACONDA CLOUD
Search Anaconda Cloud
Gallery
About
Anaconda
Help
Download Anaconda
Sign In
bioconda / packages / flye 2.8.1
1
Fast and accurate de novo assembler for single molecule sequencing reads
Conda
Files
Labels
Badges
License: BSD-3-Clause
Home: https://github.com/fenderglass/Flye/
30054 total downloads
Last upload: 1 month and 29 days ago
Installers
Info: This package contains files in non-standard labels.
conda install
linux-64 v2.8.1
osx-64 v2.8.1
To install this package with conda run one of the following:
conda install -c bioconda flye
conda install -c bioconda/label/cf201901 flye

(continued from previous page)

```
#             --nano-corr | --subassemblies) file1 [file_2 ...]
#             --genome-size SIZE --out-dir PATH
conda env list
conda deactivate
flye
#bash: flye: command not found
#
du -hs /zfs/personal/${USER}/miniconda3/
#820M    miniconda3/
```

- Look what is in your environment with:

```
conda list -n nptools
conda list -n base
```

1.6.2.1 Example: Using Conda Installation of Flye

```
conda deactivate
conda activate nptools
cd /zfs/omics/personal/${USER}/
mkdir -p ecoli
cd ecoli
wget https://zenodo.org/record/1172816/files/Loman_E.coli_MAP006-1_2D_50x.fasta
date
flye --nano-raw Loman_E.coli_MAP006-1_2D_50x.fasta --out-dir . --threads 4
date
```

1.7 Slurm Overview

- Jobs on the compute cluster are scheduled by a queue manager or cluster workload manager, called Slurm.
- **Slurm has three key functions.**
    1. It allocates exclusive and/or non-exclusive access to resources (compute nodes) to users for some duration of time.
    2. It provides a framework for starting, executing, and monitoring work (normally a parallel job) on the set of allocated nodes.
    3. It arbitrates contention for resources by managing a queue of pending work.
- Although Slurm offers several ways to restrict the use of resources to a maximum, currently Crunchomics is set up to give users a maximum amount of freedom. It is very easy to claim significant parts or even the whole cluster for extended periods of time. We expect users to use this freedom wisely and allow other users to do their work. Keep an eye on the overall system usage while running big jobs. Also free allocations which are not used. If you plan to run very big projects (10.000+ core-hours) you are encouraged to contact us beforehand to discuss distribution of the work in time. If we notice a user is using too much resources we will contact him or her. If necessary we can put restrictions on usage.

1.7.1 Additional help and information

This guide only covers a basic introduction/overview of Slurm with a focus on subjects which are specific to the Crunchomics cluster and bioinformatics problems and targeted at people who have some familiarity with the command line. Lots of information about slurm and the commands to use slurm is availabe from other sources:

- Detailed help information with the slurm commands is available, e.g. `sinfo --help`
- Nice tutorials and intros on the web, e.g. Slurm quick start.

1.7.2 Quick slurm demo

1.7.2.1 Information about the cluster

- **`sinfo` to get information about the cluster: node names, state, partition (queue)**
  - partition: the queues that are available
  - state: idle (available for jobs), mix (partly available), down etc. Slurm codes
  - node list: the names of the nodes omics-cn001 to omics-cn005

```
(base) bash-4.2$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
all*         up   infinite      5    mix omics-cn[001-005]
galaxy       up   infinite      5    mix omics-cn[001-005]
```

- **`squeue`: view information about jobs in the queue**
  - JOBID: every job gets a number. You can manipulate jobs via this number (e.g. with `scancel`)
  - ST: state, R = running, PD = pending, see: Slurm state codes.

```
bash-4.2$ squeue
      JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
       4168       all     bash jrauwer1  R      36:44      1 omics-cn001
       4175       all snakejob wdleeuw1  R      35:11      1 omics-cn003
       4176       all snakejob wdleeuw1  R      35:11      1 omics-cn003
       4177       all snakejob wdleeuw1  R      35:11      1 omics-cn003
       4178       all snakejob wdleeuw1  R      35:11      1 omics-cn004
       4179       all snakejob wdleeuw1  R      35:11      1 omics-cn004
       4180       all snakejob wdleeuw1  R      35:11      1 omics-cn004
       4181       all snakejob wdleeuw1  R      35:11      1 omics-cn005
       4182       all snakejob wdleeuw1  R      35:11      1 omics-cn005
       4183       all snakejob wdleeuw1  R      35:11      1 omics-cn005
       4184       all snakejob wdleeuw1  R       3:31      1 omics-cn001
       4185       all snakejob wdleeuw1  R       3:16      1 omics-cn001
```

1.7.2.2 Multithreading

Example of running a multithreaded program, which will be used to illustrate the use of slurm further below. We want to map a collection of reads (**READS**) to an an organism (**DB**) and write the result to **RESFILE**. (`/dev/null` means the data is discarded)

```
#You could run this program interactively on omics-h0.science.uva.nl
export READS="/zfs/omics/software/doc/Crunchomics_intro/DemoData/sample_1M.fastq"
export DB="/zfs/omics/software/doc/Crunchomics_intro/DemoData/db/NC_000913_ncbi"
```

(continues on next page)

(continued from previous page)

```
export RESFILE="/dev/null"
#
# single thread:
bowtie2 -t  -p 1  -x  $DB -U $READS >  $RESFILE
# multi threaded  use 4 threads:
bowtie2 -t  -p 4  -x  $DB -U $READS >  $RESFILE
```

1.7.2.3 srun

However, it is more efficient to run this mapping on the compute cluster using the same command in slurm with `srun` executed from the head node omics-h0.science.uva.nl:

```
# run bowtie as a single job with 16 cores:
srun -n 1 --cpus-per-task 16  bowtie2 -t -p 16  -x  $DB -U $READS  >  $RESFILE
# also possible: run an interactive  shell on a compute node, in this case for 10␣
↪minutes:
srun --pty -t 10 bash -i
```

- Allocation of the compute resource is for the duration of the command.

1.7.2.4 salloc

Often a job consists of a number of steps (jobsteps)

- Explicit allocation with `salloc` for interactive running of several jobs in row.

```
# allocate 16 cpu's on a single node, in this case for 30 minutes
salloc -n 1 -t 30 --cpus-per-task 16 bash
# the shell runs on the headnode srun is used to use the allocation
# srun uses the available cpu's in the allocation and using
srun bowtie2 -t  -p 16  -x  $DB -U $READS  >  $RESFILE
# with salloc we set 16 cpus-per-task.
# Slurm holds this information in the variable $SLURM_CPUS_PER_TASK,
# which we can use in the bowtie command:
srun bowtie2 -t  -p $SLURM_CPUS_PER_TASK  -x  $DB -U $READS  >  $RESFILE
#
# more SLURM variables in the environment:
env | grep SLURM
#
# it is not a good idea to ask for more threads than available in the allocation
srun bowtie2 -t  -p 64  -x  $DB -U $READS  >  $RESFILE
# the threads start to compete for the 16 availabe cores.
# If we ask for less threads than available
srun bowtie2 -t  -p 2  -x  $DB -U $READS  >  $RESFILE
# only 2 cores in the allocation  are used
exit
#  Release the allocated resources
```

1.7.2.5 sbatch

srun and salloc wait/block until resources are available. If the cluster is full you have to wait until resources become available. Sbatch is used to send the job to put the job in the queue and schedule it when the resources become available.

sbatch: create batch file called `test.sc` in a text editor on the headnode (e.g. vim, nano, gedit). It has the following content:

```
#!/bin/bash
#SBATCH --cpus-per-task 16
srun bowtie2 -t  -p $SLURM_CPUS_PER_TASK  -x  $DB -U $READS  >  $RESFILE
```

- *Note: the #SBATCH is not a comment but sets the cpus per task variable to 16. See below.*

Submit the created batch file to slurm using sbatch:

```
sbatch test.sc
```

In batch mode the standard output of the issued commands, which would normally appear in the terminal is written to a file. By default the name of this file is `slurm-<JOBID>.out` After a job is submitted it will run independent of the terminal, you can log out and come back later for the results.

The compute resources that are asked for and other job parameters can be specified by *parameters*. These parameters can be specified on the commandline (like in the salloc example above). For sbatch the parameters can be included in the batch script. The previously given `#SBATCH --cpus-per-task 16` is an example in which 16 cpus are asked for. Lines starting with `#SBATCH` are interpreted as parameters for the slurm job. Possible parameters can be found in the Slurm documentation: Sbatch

1.7.2.6 scancel

Use scancel to kill running jobs and remove waiting jobs from the queue

```
scancel [JOBID]
```

is used to cancel a particular job. All your jobs are killed/removed using:

```
scancel -u $USER
```

1.8 Slurm Jobs

1.8.1 Jobs and Job Steps

- A compute job can consist of several steps. For example: you download the files, you down sample and then you do an alignment. These steps are job steps and are invoked by `srun` from with in the batch script. Each srun command in a batch script can ask for its own set of resources as long as it fits in the allocation. In other words the srun commands in a batch script are bounded by the allocation for the batch script. For example an srun can not ask for more cpu's than asked for in the sbatch file.
- **Some useful srun flags are:**
  - **-c, --cpus-per-task=ncpus** number of cpus required per task
  - **-n, --ntasks=ntasks** number of tasks to run
  - **-N, --nodes=N** number of nodes on which to run (N = min[-max])

  - **-o, --output=out** location of stdout redirection
  - **-w, --nodelist=hosts** request a specific list of hosts

- Print the name of the node with the command `hostname`

```
srun hostname
#omics-cn004  this job was allocated to cn004
```

- Carry out this task four times:

```
srun -n4 hostname
#omics-cn004
#omics-cn004
#omics-cn004
#omics-cn004  again allocation on cn004
```

Note that the effect of the -n4 flag is that the program (hostname) is automatically started 4 times. For software which runs on multiple nodes and commucates between instances through mpi this is useful. In case of a multi threaded program it is usually not what you want.

- Now, ask for four nodes

```
srun -N4 hostname
#omics-cn002
#omics-cn003
#omics-cn001
#omics-cn004  now, cn001, cn002, cn003 and cn004 are allocated
```

- Ask for a specific host:

```
srun -n2 -w omics-cn002 hostname
#omics-cn002
#omics-cn002
```

- Output to a file (here: hn.txt that is stored in your current directory)

```
srun -N3 -n5 -o hn.txt hostname
cat hn.txt
#omics-cn001
#omics-cn001
#omics-cn002
#omics-cn002
#omics-cn003  #001 and 002 are used twice, 003 is used once
```

- A job consists in two parts: resource requests and job steps. Resource requests consist in a number of CPUs, computing expected duration, amounts of RAM or disk space, etc. Job steps describe tasks that must be done, software which must be run.
- The typical way of creating a job is to write a submission script. A submission script is a shell script, e.g. a Bash script, whose comments, if they are prefixed with SBATCH, are understood by Slurm as parameters describing resource requests and other submissions options. You can get the complete list of parameters from the sbatch manpage `man sbatch`.
- get hints for writing a job script at the script generator wizzard Script Generator Wizzard Ceci Ignore the cluster names and replace `#SBATCH --partition=defq` with `#SBATCH --partition=all`).

1.8.2 Batch jobs: sbatch

- **Make a text file with the content as in the box below and save it as batch1.sh:**
  - it writes the output to the file res.txt
  - it consists of one task
  - it allocates 10 minutes of compute time and 10 MB memory
  - I expect this script to run at least 15 seconds and to print 3 hostnames and 3 dates.
  - I can execute the script with `sbatch sbatch1.sh` and monitor the script with `squeue`

```
#!/bin/bash
#
#SBATCH --job-name=batch1
#SBATCH --output=res.txt
#
#SBATCH --ntasks=1
#SBATCH --time=10:00
#SBATCH --mem-per-cpu=10
#
srun hostname
srun sleep 5
srun date
srun hostname
srun sleep 5
srun date
srun hostname
srun sleep 5
srun date
```

- execute the script and monitor it

```
sbatch batch1.sh
squeue
```

- This is the output:

```
(base) bash-4.2$ sbatch batch1.sh
Submitted batch job 4963
(base) bash-4.2$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
              4963       all   batch1 jrauwer1  R       0:04      1 omics-cn001
(base) bash-4.2$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
(base) bash-4.2$ cat res.txt
omics-cn001
Mon Nov  2 17:35:32 CET 2020
omics-cn001
Mon Nov  2 17:35:39 CET 2020
omics-cn001
Mon Nov  2 17:35:46 CET 2020
(base) bash-4.2$
```

  - The job goes through the PENDING state (PD), then enters the RUNNING state (R) and finally goes to the COMPLETED state, or FAILED state.
  - Indeed 3 times hostname and 3 times date some 7 seconds apart

  - The job id issued was 4963

1.8.3 Parallel Jobs

- **Here, we will only discuss parallel jobs**
  - by running several instances of a single-threaded program (so-called embarrassingly parallel paradigm or a job array)
  - by running a multithreaded program (shared memory paradigm, e.g. with OpenMP or pthreads)
- Other types of parallel jobs: see Ceci - see Going parallel.
- **From this same website: Tasks are requested/created with the *–ntasks* option, while CPUs, for the multithreaded programs, are requested with the *–cpus-per-task* option. Tasks can be split across several compute nodes, so requesting several CPUs with the *–cpus-per-task* option will ensure all CPUs are allocated on the same compute node. By contrast, requesting the same amount of CPUs with the *–ntasks* option may lead to several CPUs being allocated on several, distinct compute nodes.**
  - Multithreaded programs run on one specific compute node: use the *–cpus-per-task* flag with these programs.

1.8.3.1 Multithreaded bowtie2 Example

- **Many genomics software use a multithreaded approach. We start with a bowtie2 example:**
  - We want to align 2 fastq files from the European Nucleotide Archive to the Mycoplasma G37 genome.
  - **Workflow:**
    * download the G37 genome to the /scratch directory of the node
    * build the genome index on this /scratch directory
    * download the fastq files to scratch
    * do the alignment
    * store the resulting sam files in your *personal* directory.
  - Our batch script is below (save it as align_Mycoplasma and run it with `sbatch align_Mycoplasma` and monitor it with `squeue`):

```
#!/bin/bash
#
#SBATCH --job-name=align_Mycoplasma
#SBATCH --output=res_alignjob.txt
#
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=8
#SBATCH --time=10:00
#SBATCH --mem-per-cpu=2000
#
cd /scratch
srun wget https://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/027/325/GCF_000027325.1_
↪ASM2732v1/GCF_000027325.1_ASM2732v1_genomic.fna.gz -P ./
srun bowtie2-build GCF_000027325.1_ASM2732v1_genomic.fna.gz MG37
srun wget ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR486/ERR486827/ERR486827_1.fastq.gz -P ./
```

(continues on next page)

(continued from previous page)

```
srun wget ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR486/ERR486827/ERR486827_2.fastq.gz -P ./
srun bowtie2 -x MG37 -1 ERR486827_1.fastq.gz -2 ERR486827_2.fastq.gz --very-fast -p
↪$SLURM_CPUS_PER_TASK -S /zfs/omics/personal/${USER}/result.sam
```

- the output of the job (res_alignjob.txt) is stored where you execute the sbatch. It contains the information that is normally written to your standard output (your screen). In this case, the progress of the download, the progress of the indexing and the alignment summary.
- the actual result of the alignment (the sam file) is written to your *personal* directory.
- the number of threads in the bowtie2 command (job step) is taken from the *SLURM variable* `$SLURM_CPUS_PER_TASK` that was given at the start of the job. You could have given any number up to 8 with the -p flag. When you issue a number >8 the job will still be executed with the number of threads defined by `$SLURM_CPUS_PER_TASK` (in this case 8).

1.8.4 Multithreaded Example in C

- The example below is a C code illustration of how treads are forked from a master thread. This idea can be used when you make your own parallelized code.
- Save the file below as omp_hoi.c:

```
/******************************************************************************
* FILE: omp_hoi.c
* DESCRIPTION:
*   OpenMP Example - Hello World - C/C++ Version
*   In this simple example, the master thread forks a parallel region.
*   All threads in the team obtain their unique thread number and print it.
*   The master thread only prints the total number of threads.  Two OpenMP
*   library routines are used to obtain the number of threads and each
*   thread's number.
* AUTHOR: Blaise Barney  5/99
* LAST REVISED: 04/06/05
******************************************************************************/
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#
int main (int argc, char *argv[])
{
int nthreads, tid;
/* Fork a team of threads giving them their own copies of variables */
#pragma omp parallel private(nthreads, tid)
{
        /* Obtain thread number */
        tid = omp_get_thread_num();
        printf("Hello World from Crunchomics thread = %d\n", tid);
        /* Only master thread does this */
        if (tid == 0)
        {
                nthreads = omp_get_num_threads();
                printf("Master thread says: number of threads = %d\n", nthreads);
        }
```

(continues on next page)

(continued from previous page)

```
  }  /* All threads join master thread and disband */
}
```

- and compile it:

```
gcc -fopenmp omp_hoi.c -o hoi.omp
```

- run it through the following sbatch script:

```
#!/bin/bash
#SBATCH --job-name=test_omp
#SBATCH --output=res_omp.txt
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=5
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
./hoi.omp
```

```
Submitted batch job 6919
(base) bash-4.2$ cat res_omp.txt
Hello World from Crunchomics thread = 0
Master thread says: number of threads = 5
Hello World from Crunchomics thread = 4
Hello World from Crunchomics thread = 1
Hello World from Crunchomics thread = 3
Hello World from Crunchomics thread = 2
(base) bash-4.2$
```

1.8.4.1 Job Arrays: Parallel example with Rscript

- You can use the sbatch `--array=<indexes>` parameter to submit a job array, i.e., multiple jobs to be executed with identical parameters. The indexes specification identifies what array index values should be used.
- Suppose we study 9 chromosomes of a certain organism (for which we have 9 files, 9 R-objects or something of the kind that we want to process).
- Below a sbatch script is shown that spawns 9 sub-tasks (the array starting with 0 to 8). With each sub-task an R script is run that uses the element of the vector `${CHROMS}` as indicated by the index of the array. Thus, 9 Rscript-processes are run, each for a chromosome:
- copy the code below in a file called parJobBatch.sh

```
#!/bin/bash
#
#SBATCH --job-name=ParTest
#SBATCH --ntasks=1
#SBATCH --array=0-8
#
CHROMS=("chr1" "chr2" "chr3" "chr4" "chr5" "chr6" "chrX" "chrY" "Mit")
#
srun Rscript parJob.R ${CHROMS[$SLURM_ARRAY_TASK_ID]}
```

- copy the code below in a file called parJob.R

```
args=commandArgs(trailingOnly=TRUE)
chromosome=args[1]
#
cat("Start work on",chromosome,"\n")
cat("working ....\n")
# put the code what you want to do with each chromosome here
Sys.sleep(20)
cat("Done work on",chromosome,"\n")
```

- The job is started with `sbatch parJobBatch.sh`
- The job 9 job steps each have an entry in the queue

```
(base) bash-4.2$ sbatch parJobBatch.sh
Submitted batch job 7080
(base) bash-4.2$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
              6866       all  Rscript  ealiche  R    5:51:16      1 omics-cn001
              6886       all     bash jrauwer1  R    4:16:53      1 omics-cn001
            7080_4       all   MyTest jrauwer1  R       0:02      1 omics-cn001
            7080_6       all   MyTest jrauwer1  R       0:02      1 omics-cn001
            7080_9       all   MyTest jrauwer1  R       0:02      1 omics-cn001
```

- **Result:**
  - 9 files each outputted by a different R process. With content such as:

```
cat slurm_[jobid]_0.txt
```

Where the number after the underscore is the jobstep number:

```
Start work on chr1
working ....
Done work on chr1
```

- Rscript was used here to illustrate the `--array` functionality, any program which has to be run for a number of inputs can be set up in this way.
- A list of files can be used instead of chromosomes. If there are 20 file to be processed the relevant part of the batch script would look something like:

```
#SBATCH --array=0-19
FILES=(/zfs/omics/personal/*)
srun program ${FILES[$SLURM_ARRAY_TASK_ID]}
```

1.8.4.2 Use Conda Environments on the Compute Nodes

- Run the flye assembly (see 6.2.1) on a compute node using sbatch.
- Remark: before you execute the sbatch command, activate the proper conda environmnent. In this case it is necessary to activate nptools because flye was installed in this environment. `activate conda nptools` The activation will be passed to the compute nodes.

```
#!/bin/bash
#
#SBATCH --job-name=ecoli_assemble
#SBATCH --output=res_ecoli_assembly.txt
#
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=8
#SBATCH --time=20:00
#SBATCH --mem-per-cpu=32000
#
SCRATCH=/scratch/$USER/ecoli
mkdir -m 700 -p $SCRATCH
cd $SCRATCH
srun wget https://zenodo.org/record/1172816/files/Loman_E.coli_MAP006-1_2D_50x.fasta
date
srun flye --nano-raw Loman_E.coli_MAP006-1_2D_50x.fasta --out-dir /zfs/omics/personal/$
↪{USER}/ecoli-batch --threads     $SLURM_CPUS_PER_TASK
date
```

- Save this file as assemble_ecoli.sh and run from the headnode `sbatch assemble_ecoli.sh`

1.8.5 Interactive Shells Continued

- For interactive work: use the head node.
- **In some cases it might be convenient to have shell access on the compute node, for instance to look at the memory and cpu allocation of a specific process.**
  - **limit the duration of this shell by issueing the -t <min>**
  - use the -w flag to go to the node you want your shell to live in.
  - have a shell on cn001 for 1 minute:

```
hostname
#omics-h0.science.uva.nl
srun -n 1  -t 1 --pty -w omics-cn001 bash -i
hostname
#omics-cn001
```

1.8.5.1 Example using an interactive shell

- Re-run the *Multithreaded bowtie2 example*. Configure to use 2 threads. Use squeue to find out the node on which the job runs. Then, from the head node (a shell for a minute on -in this case- cn001):

```
sbatch
srun -n 1  -t 1 --pty -w omics-cn001 bash -i
```

- I see with `squeue` that my alignment script is running as slurm job 7094 on compute node cn001. Hence, I start an interactive shell at compute node 001 (for a minute) and monitor with `top` that indeed there is a processor load of 200% (2 threads) used by the bowtie2-align-s program.

```
(base) bash-4.2$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
              6866       all  Rscript  ealiche  R    6:41:34      1 omics-cn001
              7094       all align_My jrauwer1  R       0:04      1 omics-cn001
(base) bash-4.2$ srun -n 1  -t 1 --pty -w omics-cn001 bash -i
(base) bash-4.2$ top
top - 17:05:10 up 103 days, 8 min,  2 users,  load average: 2.19, 1.31, 1.12
Tasks: 688 total,   1 running, 293 sleeping,   0 stopped,   0 zombie
%Cpu(s):  4.7 us,  0.1 sy,  0.0 ni, 95.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 52830649+total, 51415395+free, 12672360 used,  1480200 buff/cache
KiB Swap: 20971512 total, 20686388 free,   285124 used. 51264774+avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
23799 jrauwer1  20   0  222880  28860   4356 S 200.3  0.0   1:23.13 bowtie2-align-s
 9088 ealiche   20   0   10.0g   9.6g  11304 S 100.0  1.9 400:51.16 R
23834 jrauwer1  20   0  172980   5192   3884 R   0.7  0.0   0:00.09 top
19576 root      20   0       0      0      0 I   0.3  0.0   0:00.46 kworker/4:1-eve
23047 root      20   0       0      0      0 I   0.3  0.0   0:00.17 kworker/u129:4-
```

1.9 Using Slurm Indirectly

1.9.1 Using Slurm indirectly

Instead of controlling job preparation and submission directly via the command line it is possible to access slurm through packages in (programming) environments. A number of options are presented.

1.9.1.1 Slurm in R

Using the rslurm package an R session running on the headnode (omics-h0) that can run jobs on the cluster. The functions `slurm_call` and `slurm_apply` can be used to submit computationally expensive jobs on the compute nodes. The help provided in the package about the functions gives an example. To test demonstrate it on Crunchomics with some slight modifications

```
require("rslurm")
# Create a data frame of mean/sd values for normal distributions
pars <- data.frame(par_m = seq(-10, 10, length.out = 1000),
           par_sd = seq(0.1, 10, length.out = 1000))
#
# Create a function to parallelize
ftest <- function(par_m, par_sd) {
samp <- rnorm(2e6, par_m, par_sd)
c(s_m = mean(samp), s_sd = sd(samp))
}
#
sjob1 <- slurm_apply(ftest, pars,slurm_options=list(mem="1G"))
get_job_status(sjob1)
# wait until get_job_status(sjob1)$completed
# the job will take about a minute if resources are available.
res <- get_slurm_out(sjob1, "table")
all.equal(pars, res) # Confirm correct output
cleanup_files(sjob1)
```

Some notes while running it on Crunchomics:

- While running jobs the package creates a map in the current directory which is used to communicate data to the R processes on the nodes called *slurm_[jobname]*. The working directory changes to that directory so relative

file paths in the function will not work.

- The cleanup_files function can be used to remove this directory.
- By default the jobs will have 100MB of memory. Rather than giving up, the typical behaviour of R is to become very very slow if it does not have enough memory. Use `slurm_options=list(mem="1G")` in the slurm_call/slurm_apply to get enough(Change 1G to the needed amount) memory for your function.
- R has to run on the Slurm-cluster (Jobs can not be submitted from omics-app01)

1.9.1.2 Slurm in Python

A package called simple_slurm is available in the madpy3 environment. Check Simple Slurm pypi.org for package documentation and examples.

1.9.1.3 Slurm and Snakemake

Existing snakemake pipelines can be run on the cluster (almost) without change, using a parameter file (cluster.json) snakemake is told how jobs have to be submitted. Below is an example setup of a normal snakefile which maps a number of samples to a genome. This demonstration can be copied and run using the following commands:

```
cp -r /zfs/omics/software/doc/Crunchomics_intro/SnakeDemo .
cd SnakeDemo
ls
# 3 files and an empty directory are copied
#
# Snakemake has to be available use
# source /zfs/omics/software/v_envs/madpy3/bin/activate
# if which snakemake comes back without result
#
batch snakemake.cmd
```

Check the snakefile using

```
cat Snakefile
```

Snakefile is a typical snakemake input file to trim and map some samples, but the process should work for any Snakefile. The snakefile can be processed unchanged by snakemake utilizing the cluster using the a parameter file:

```
#cluster.json
{
"__default__" :
{
        "time" : "00:30:00",
        "n" : 1,
        "c" : 1,
        "memory"    : "2G",
        "partition" : "all",
        "output"    : "logs/slurm-%A_{rule}.{wildcards}.out",
        "error"     : "logs/slurm-%A_{rule}.{wildcards}.err",
},
"align_to_database" :
{
        "time" : "02:00:00",
```

(continues on next page)

(continued from previous page)

```
        "memory"    : "10G",
        "c" : 8
},
"trim_read" :
{
        "time" : "01:00:00",
        "c" : 2
}
}
```

In the parameter file the options values can be specified for how snakemake has to configure the slurm jobs. In it the number of cores, the amount of memory etc. is given for the rules. If the name of the rule is not given snakemake uses the values given in `__default__`

Using the `--cluster` and `--cluster-config` parameters snakemake is able replace the job execution commands with slurm batch jobs.

```
snakemake -j 8 --cluster-config cluster.json --cluster "sbatch  -p {cluster.partition} -
↪n {cluster.n}  -c {cluster.c} -t {cluster.time} --mem {cluster.memory} -o {cluster.
↪output} "
```

The command above can be wrapped into an sbatch script (the file: snakemake.cmd) In this way the snakemake program itself runs as a slurm job. Note that sbatch commands executed from within an sbatch job will run in a new and separate allocation independent of the allocation from which sbatch is run. While the snakemake program itself runs on a single core the batches it runs like the mapping tasks can use 8 cores.

The mapping of parameters defined in the cluster.json file to the sbatch command. Snakemake takes care that jobs are submitted using the parameters in the cluster.json file. By default the parameters in `__default__` are used. For rules that match a clause in the cluster.json file for example `align_to_database` changes from the default can be included for example the the number of cores (8 instead of 1) and the time (2 hour intead of 30 minutes). As mentioned in the introduction snakefiles can be run **almost** unchanged. Some rules do very little actual work and it make little sense to setup and schedule a job for it for example creating a directory. These rules can be declared as `localrules` at the top of the snakefile. This means snakemake will execute them directly instead of submitting them as a slurm job. Also depending on if and how thread counts are dealt with in the snakefile tweaking can improve the utilization of the allocated resources.

The snakemake parameter `-j` which normally limits the number of cores used by snakemake limits the number of concurrently submitted jobs regardless of the number of cores each job allocates. Setting -j to high values such as 999 as is done in some examples on the internet is usually not a good idea as it might claim the complete cluster.

Using snakemake in combination with the (super fast NVMe SSD) local scratch partition for temporary storage adds an extra challenge. The main mode of operation of snakemake is checking the existence of files. However, a snakemake process running on the headnode can not access the scratch on the nodes. A possibility is to configure snakemake (in cluster.json) to run on a single node and to combine that with running snakemake itself using sbatch in the selected node.

1.10 Debugging Slurm jobs

If things do not run as expected there are a number of tools and tricks to find out why.

- Initially `squeue` can be used to get the state of the job. Is it in the queue, running or ended. A job which request resources which are not available is queued until the resources become available. If a jobs ask for 6 nodes it will be queued but never run as there are only 5 compute nodes.
- Check the output file of the batchjob (slurm_[jobid].out or the name given using –output parameter). Use `cat slurm_[jobid].out` or `tail slurm_[jobid].out` to check what the job sent to its standard output.
- To see if the job runs as expected, an additional interactive job can be started on the node the job is running on using `srun -w [nodeX] --pty bash -i`. The commands `top` and `htop` are useful to see which programs are running and if the program is using the allocated resources as expected.
- If it is not possible to allocate a job on the node an option is to directly ssh into node. This is only possible if you have an active job on the particular node.
- A common problem is lack of memory. Memory is also limited in by the slurm allocation. The default memory allocation is 100MB. While this is fine for some, it is not enough for many jobs. Some jobs simply give up with an error if they don't have enough memory. Others try to make the best of it. This might result in jobs, which are expected to complete in a couple of hours to run for days. A telltale sign of this happening is that memory usage shown by top is close to or equal to the allocation asked for. While setting up jobs keep in mind the compute nodes have 512GB each. That is 8G for each of the 64 cores on average. Don't allocate what you do not need: this might be used by others. It is not unreasonable to allocate 128GB of memory for a 16-core job.

1.11 Use of Local NVMe Storage on Compute Nodes

Each compute node in the cluster has 7.3TB local and very fast NVMe storage available in /scratch. By storing temporary and/or much accessed data on the scratch disk, file access times can be improved significantly. It also reduces the load on the filer which will benefit all users. However as local storage it can be only accessed on a particular node. So either data has to be synchronized over all nodes, or jobs have to be fixed on particular nodes using the `-w flag`

- **You might remember that in the *Multithreaded bowtie2 example* we used the local /scratch disk to copy the genome index and the fastq files to. The alignment job was calculated (in this particular case on cn001).**
  - These files are **not** automatically removed when the job has finished. So multiple jobs can access and further process data in scratch.
  - To prevent /scratch from filling up, files which have not been accessed (read or written) for (currently) a month are removed. The time unused files are removed might be reduced if the file system fills up too quickely.
  - This policy makes it possible to reuse large files on the local /scratch disk on a compute node. For instance, look at the scratch of cn001 (this might look differently when you read this):

```
srun -n1 -w omics-cn001 ls -lh /scratch
```

```
(base) bash-4.2$ srun -n1 -w omics-cn001 ls -lh /scratch
total 51M
-rw-r--r--. 1 jrauwer1 Domain Users  19M Nov  3 12:00 ERR486827_1.fastq.gz
-rw-r--r--. 1 jrauwer1 Domain Users  23M Nov  3 12:00 ERR486827_2.fastq.gz
-rw-r--r--. 1 jrauwer1 Domain Users 162K Feb  4  2020 GCF_000027325.1_ASM2732v1_genomic.fna.gz
-rw-r--r--. 1 jrauwer1 Domain Users 4.2M Nov  3 12:00 MG37.1.bt2
-rw-r--r--. 1 jrauwer1 Domain Users 142K Nov  3 12:00 MG37.2.bt2
-rw-r--r--. 1 jrauwer1 Domain Users   17 Nov  3 12:00 MG37.3.bt2
-rw-r--r--. 1 jrauwer1 Domain Users 142K Nov  3 12:00 MG37.4.bt2
-rw-r--r--. 1 jrauwer1 Domain Users 4.2M Nov  3 12:00 MG37.rev.1.bt2
-rw-r--r--. 1 jrauwer1 Domain Users 142K Nov  3 12:00 MG37.rev.2.bt2
```

- Here we make use of the genome index on cn001 while doing an alignment with 2 new files (save the file below as align_Mycoplasma3.sh):

```
#!/bin/bash
#
#SBATCH --job-name=align_Mycoplasma
#SBATCH --output=res_alignjob.txt
#
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=8
#SBATCH --time=10:00
#SBATCH --mem-per-cpu=2000
#
WD="/scratch/$USER"
mkdir -p $WD
cd $WD
#
#test if the Mycoplasma genome is already there, download it and build the index if not.
fl="GCF_000027325.1_ASM2732v1_genomic.fna.gz"
if [ ! -f "${fl}" ]; then
        srun wget https://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/027/325/GCF_000027325.
↪1_ASM2732v1/GCF_000027325.1_ASM2732v1_genomic.fna.gz -P ./
        srun bowtie2-build GCF_000027325.1_ASM2732v1_genomic.fna.gz MG37
fi
#
#only download the 2 illumina files if they are not there
fl="ERR486828_1.fastq.gz"
if [ ! -f "${fl}" ]; then
        srun wget ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR486/ERR486828/ERR486828_1.fastq.
↪gz -P ./
fi
#
fl="ERR486828_2.fastq.gz"
if [ ! -f "${fl}" ]; then
        srun wget ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR486/ERR486828/ERR486828_2.fastq.
↪gz -P ./
fi
#
date
srun bowtie2 -x MG37 -1 ERR486828_1.fastq.gz -2 ERR486828_2.fastq.gz --very-fast -p
↪$SLURM_CPUS_PER_TASK -S /zfs/omics/personal/${USER}/result_ERR486828.sam
date
```

- execute with `sbatch -w omics-cn001 align_Mycoplasma3.sh`
- from `tail -n21 res_alignjob.txt`:
- I conclude the script succeeded.
- look at the /scratch directory:

```
srun -n1 -w omics-cn001 ls -lh /scratch
```

- Indeed, the index MG37 is reused (timestamp 12:00) and the two fastq files ending on 28 are newly downloaded (timestamp 18:07).

```
2020-11-03 18:07:34 (12.5 MB/s) - './ERR486828_2.fastq.gz' saved [27626304]

Tue Nov  3 18:07:35 CET 2020
410959 reads; of these:
  410959 (100.00%) were paired; of these:
    180127 (43.83%) aligned concordantly 0 times
    230152 (56.00%) aligned concordantly exactly 1 time
    680 (0.17%) aligned concordantly >1 times
    ----
    180127 pairs aligned concordantly 0 times; of these:
      155481 (86.32%) aligned discordantly 1 time
    ----
    24646 pairs aligned 0 times concordantly or discordantly; of these:
      49292 mates make up the pairs; of these:
        28040 (56.89%) aligned 0 times
        17361 (35.22%) aligned exactly 1 time
        3891 (7.89%) aligned >1 times
96.59% overall alignment rate
Tue Nov  3 18:07:52 CET 2020
(base) bash-4.2$
```

```
(base) bash-4.2$   srun -n1 -w omics-cn001 ls -lh /scratch
total 98M
-rw-r--r--. 1 jrauwer1 Domain Users  19M Nov  3 12:00 ERR486827_1.fastq.gz
-rw-r--r--. 1 jrauwer1 Domain Users  23M Nov  3 12:00 ERR486827_2.fastq.gz
-rw-r--r--. 1 jrauwer1 Domain Users  21M Nov  3 18:07 ERR486828_1.fastq.gz
-rw-r--r--. 1 jrauwer1 Domain Users  27M Nov  3 18:07 ERR486828_2.fastq.gz
-rw-r--r--. 1 jrauwer1 Domain Users 162K Feb  4  2020 GCF_000027325.1_ASM2732v1_genomic.fna.gz
-rw-r--r--. 1 jrauwer1 Domain Users 4.2M Nov  3 12:00 MG37.1.bt2
-rw-r--r--. 1 jrauwer1 Domain Users 142K Nov  3 12:00 MG37.2.bt2
-rw-r--r--. 1 jrauwer1 Domain Users   17 Nov  3 12:00 MG37.3.bt2
-rw-r--r--. 1 jrauwer1 Domain Users 142K Nov  3 12:00 MG37.4.bt2
-rw-r--r--. 1 jrauwer1 Domain Users 4.2M Nov  3 12:00 MG37.rev.1.bt2
-rw-r--r--. 1 jrauwer1 Domain Users 142K Nov  3 12:00 MG37.rev.2.bt2
```

- Obviously, this way of working is especially useful if there is a lot of IO. The size of the local scratch is net 7.3 TB.

```
srun -n1 -w omics-cn001 df -h /scratch
```

1.12 Docker / Singularity Containers

Containers are software environments which make it possible to distrubute applications along with all the dependencies needed and run these on all computers which have the container engine installed. **Docker** is a well know container platform. For security reasons the Docker engine is not available on Omics. **Singularity** is an alternative to Docker which is better suited for running in a multi-user plaform such as Crunchomics. Singularity is able to deal with Docker containers and many run without problems.

1.12.1 Singularity Hub

```
singularity pull singularity-images.sif shub://vsoch/singularity-images
singularity run lolcow.sif
singularity run lolcow.sif
```

1.12.2 Docker Hub

```
singularity build megahit docker://vout/megahit
cat /etc/*release
#CentOS Linux release 7.8.2003 (Core)
singularity shell megahit
#DISTRIB_ID=Ubuntu
#DISTRIB_RELEASE=18.04
#DISTRIB_CODENAME=bionic
exit
rm -rf ~/assembly
singularity run megahit -v
singularity run megahit -1 ERR486840_1.fastq.gz -2 ERR486840_2.fastq.gz -o ./assembly
awk '/^>/ {print}' assembly/final.contigs.fa | wc -l
#22

#Run this in a SLURM batch job:
sbatch batch_megahit.txt

#!/bin/bash
#SBATCH --job-name=assembly_gen.        # Job name

echo "Running megahit"

cd # go to home

rm -rf assembly

singularity run megahit -t 8 -1 ERR486840_1.fastq.gz -2 ERR486840_2.fastq.gz -o ./
↪assembly
awk '/^>/ {print}' assembly/final.contigs.fa | wc -l
```

CHAPTER

TWO

INDICES AND TABLES

- genindex
- search