

- Mac users, if you arrived in advance, how about installing Xquartz ?
 - <https://www.xquartz.org/>
 - (warning: needs a reboot)

This is not a

SKILLPILLS

Mini Course:
TERMINAL

Lecture III:

Interacting with Sango Deigo



November 2020

Presenter: Charles Plessy, using slides from previous years (Takeshi R. Tabuchi Y.)

Outside world
(a.k.a. the real life)

User external PC
222.14.251.35



SSH

login.oist.jp

0

Gateway

Internet

vpn.oist.jp

VPN

deigo.oist.jp

login1

1

login4

Login nodes

2

Intranet

Compute nodes

User workstation
10.2.80.230

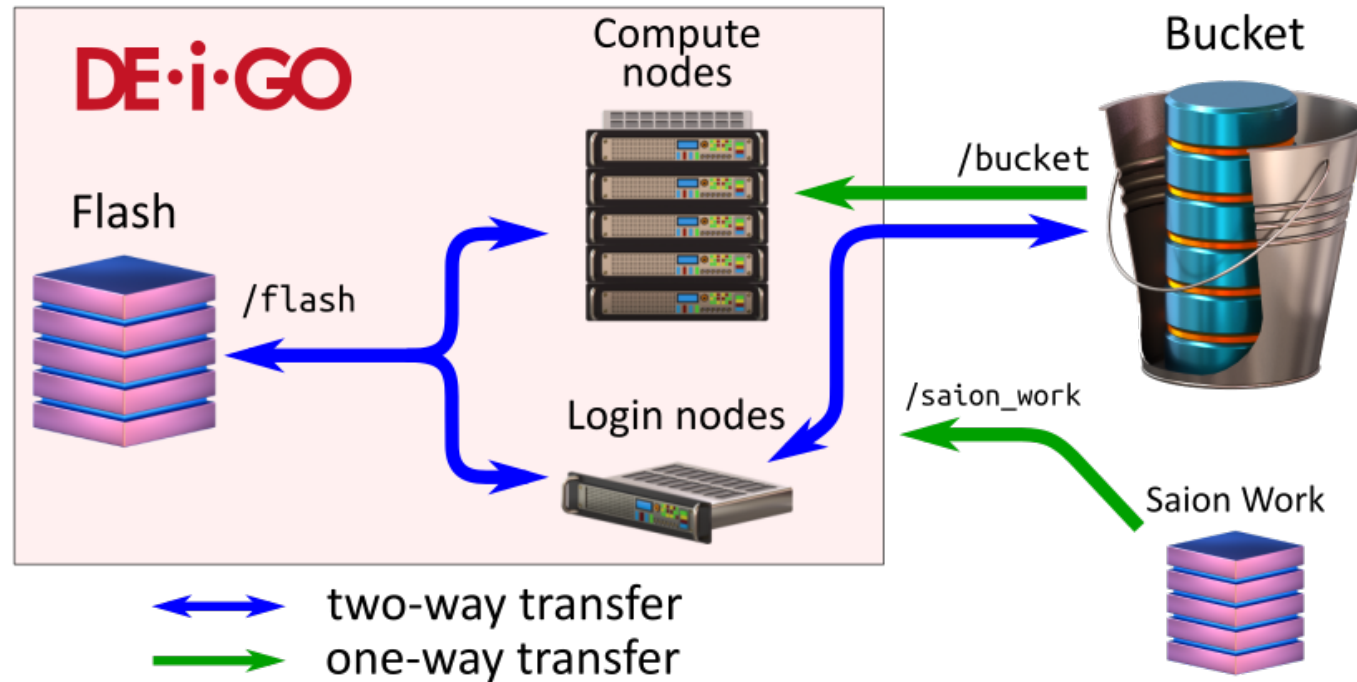


OIST network



OIST

OKINAWA INSTITUTE OF SCIENCE AND TECHNOLOGY GRADUATE UNIVERSITY



<https://groups.oist.jp/scs/getting-started>

Go up to RSD site
SCDA Top
About
Equipment Gallery
Getting started
Computing resources
Documentation
Systems status
Registration & Forms
Policies & legal notice
Calendar
Events
Posts

Contact Scientific
Computing Section
Email: it-help@oist.jp

Scientific Computing & Data Analysis Section

Getting started

HPC clusters at OIST

Accounts

1. Accessing Sango using SSH
2. Mandatory settings in your default BASH config file

Use the clusters

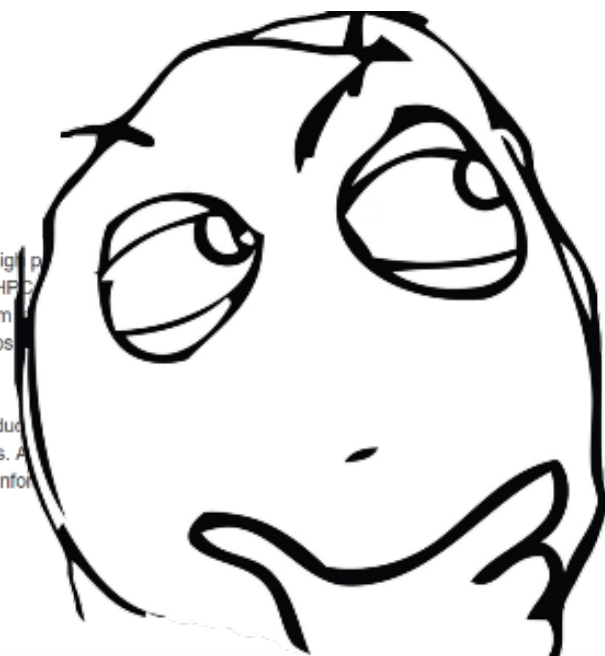
1. Transfer data in/to the clusters
2. Software available
3. Run computation on the clusters
4. Copy data from /bucket to /work, and vice versa
5. Manage your submitted computations
6. Scheduler output, error and log files

Best practices and user responsibilities

HPC clusters at OIST

OIST provides clusters, storage and software to support and promote the use of high performance computing activities. All university faculty, research staff and students are eligible to use the HPC clusters. HPC resources are shared between all users, a per unit and per user quota system is in place for the licensed software. Computations on the clusters are done by submitting jobs to the queue for processing resources.

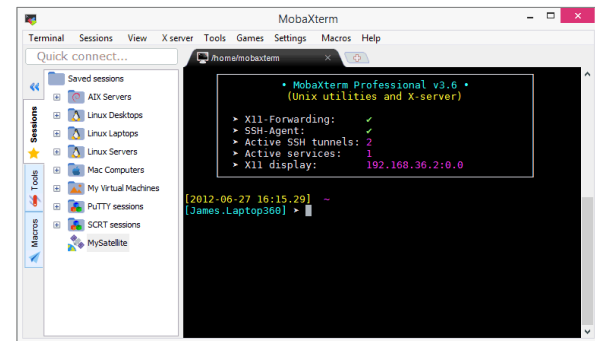
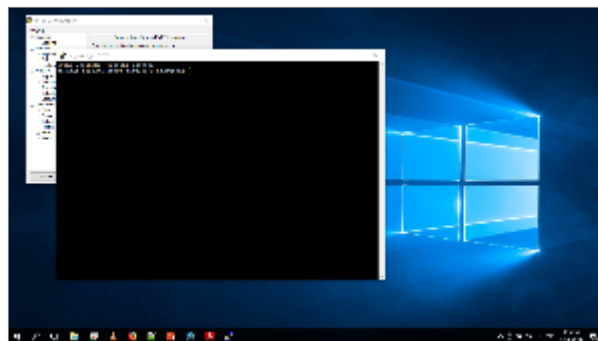
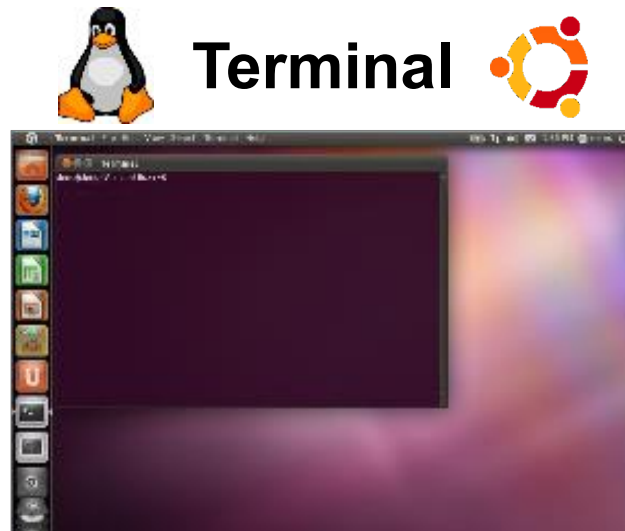
There are two HPC clusters at OIST, Sango and Tombo. Sango is OIST main production cluster used for testing, seminar, workshop, demo, and on-demand specific events. Access is granted without restriction. All OIST HPC clusters use Linux Operating System. For more information, please see the [computing resources page](#).



OIST

OKINAWA INSTITUTE OF SCIENCE AND TECHNOLOGY GRADUATE UNIVERSITY

Terminal



Secure Shell (SSH) is a protocol for operating network services securely over an unsecured network. Normally used for **remote login** by users to a network system.

Syntax `ssh [-options] [user@]server[:port] [commands]`

e.g. `$ ssh tida-name@login.oist.jp`
`$ ssh tida-name@deigo.oist.jp`

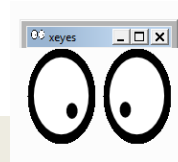
If you want to run programs with **graphical interface**:

```
$ ssh -Y tida-name@deigo.oist.jp
```

Only do this on a trusted network

Run xeyes to test your session

```
$ xeyes
```



If it does not work, you need a “X server”



-> MobaXterm <https://mobaxterm.mobatek.net/>



-> XQartz <https://www.xquartz.org/>

Place a configuration file in `~/.ssh/config` to set a default user name and default options

e.g.

```
Host deigo*  
  User tida-name  
  ForwardX11 Yes  
  Forward Agent yes
```

Only export X11 windows and SSH keys like this on a trusted network

Tired of SSH lagging? Try Mosh! <https://mosh.org/>

SSH key (public-private encryption) is used to uniquely identify you. The external OIST SSH servers are configured to use SSH key based authentication **instead of OIST password**.

<https://groups.oist.jp/it/ssh-0>

SSH access to OIST

OIST provides SSH servers to which you can connect from anywhere.

The external SSH servers are configured to use SSH key based authentication, not your OIST password.

- The external SSH servers share the same NFS home area as the tombo cluster and other SSH servers inside OIST.
- If you have added your SSH public key to your account on the tombo cluster, then you will be able to SSH to the SSH servers.
- The external SSH server is named:
 - login.oist.jp

Setup instructions

- Windows
- Mac
- Linux

Note: You will need to be on-campus at OIST, or connected via VPN to perform the initial setup procedure.



At anytime, you can change your passphrase with the command:

```
$ ssh-keygen -p
```

*Tip: when using your key, ssh ask for your pass**phrase**. If it asks for your pass**word**, something is going wrong. (It is SSH jargon, but yes, a passphrase can be a single word...)*

To save typing, you can ask your computer to memorize your passphrase during one session.

```
$ ssh-add
```

You can also forward your key to the (trusted!) server you are connecting to:

```
$ ssh -A tida-name@login.oist.jp
```

<https://groups.oist.jp/scs/deigo>

Partition	Total cores	Cores per node	Memory per node	Memory per user	Max cores per user	Max time per job
short	66552	36-128	500GB	6500G	4000	2h
compute	45568	128	500GB	7500G	2000	4 days
largejob	12800	128	500GB	— ¹	— ¹	— ¹
largemem	2424	36-40	500-750GB	— ²	— ²	— ²
bigmem	32	32	3T	—	—	—

1: largejob allocations are decided by the SCC.

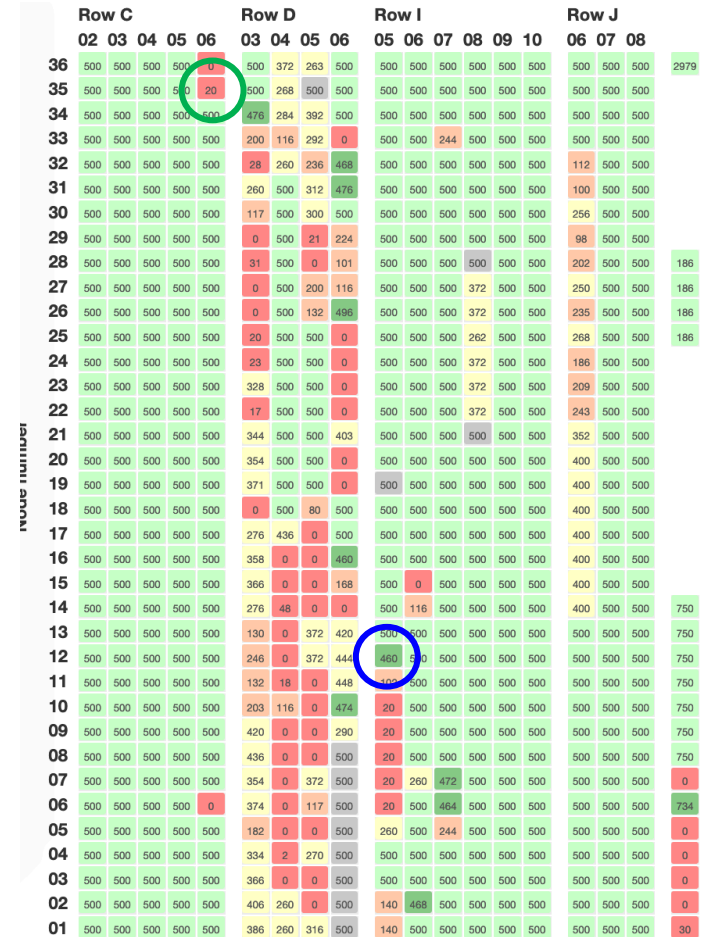
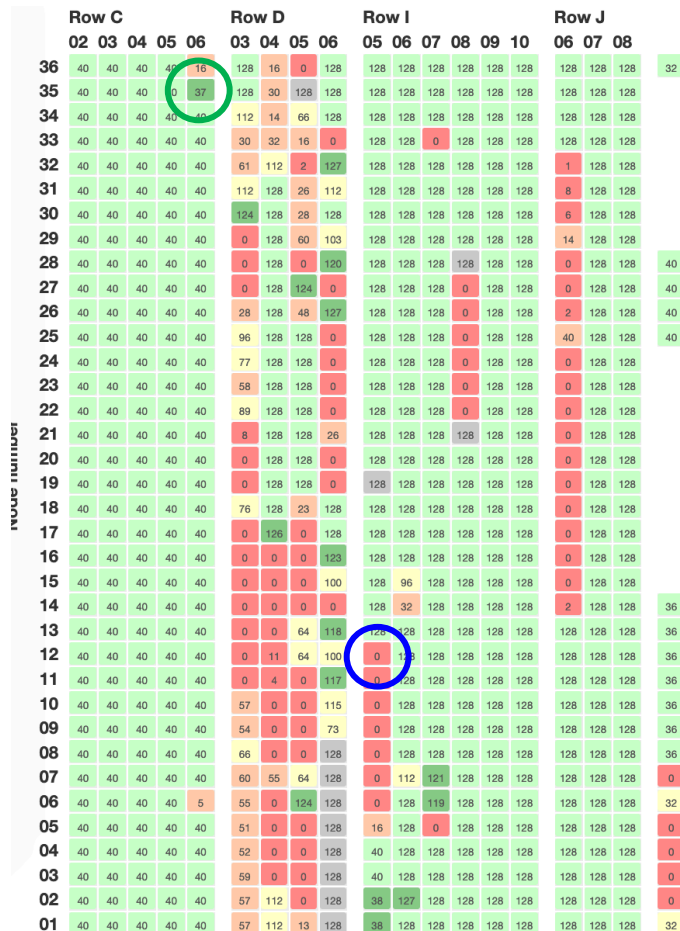
2: largemem limits you to ~5 nodes worth of resources in total.

“Compute” partition – nodes heatmap

<https://highsci.oist.jp>

Cores available

Memory available



☐ cpu temp. ☐ power ☐ hw spec. ☐ 10% ☐ 50% ☐ 90% ☐ down/drain

☐ cpu temp. ☐ power ☐ hw spec. ☐ 10% ☐ 50% ☐ 90% ☐ down/drain





KEEP CALM
AND
BE A
GOOD
HUMAN

- Remember Deigo is a shared resource: think about others!
- Don't run computational intense programs on the login nodes!
- Limit the resources you request (cores, memory and time) and the number of simultaneous jobs (no more than 50 at a time).
- Remove your temporary files.
- Reach for SCS (it-help@oist.jp or open hours 15h30-17h30, B648) if you have any question or problem, they will gladly help you.

Transfer data to/from Deigo

scp stands for secure cp (copy), which means you can copy files across ssh connection.

Copy from Local machine to Server

Syntax `scp [-options] localfile [user@server:]file`

e.g.

```
$ scp /local/path/data.tar.gz  
tida-login@deigo.oist.jp:~/data.tar.gz
```

Copy from Server to Local machine

Syntax `scp [user@server:]remotefile file`

Copy from Server 1 to Server 2

Syntax `scp [user1@server1:]file [user2@server2:]file`

rsync is faster, more flexible and versatile than `cp` and `scp`. Minimize the work by comparing the files that are already at the destination and only coping only the ones that have changed.

Syntax `rsync [-options] /source/path/ /destination/path/`

It also works over remote servers (ssh):

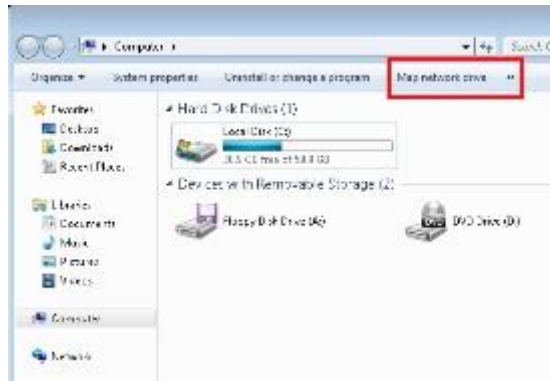
Syntax `rsync [-options] user@server:/source/ /destination/`

`rsync [-options] /source/ user@server:/destination/`

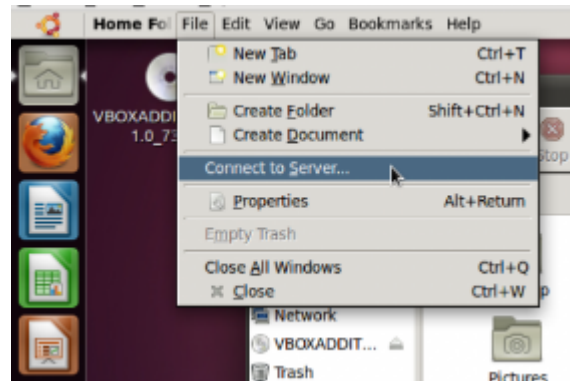
e.g. `$ rsync -a -V /local/path/myfolder/
tida-name@deigo.oist.jp:~/myfolder/`

* Options: (-a) Archive mode, (-V) Verbose

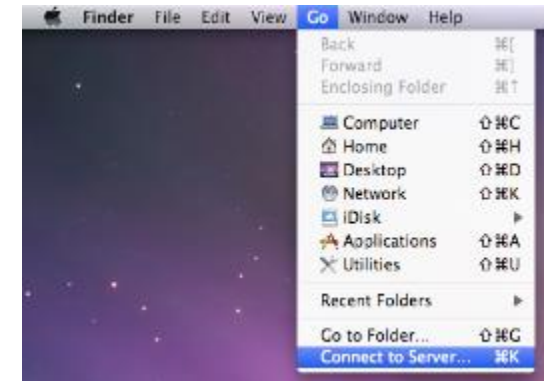
Using network drives



<https://groups.oist.jp/it/connecting-software-file-server-windows>



<https://groups.oist.jp/it/connecting-software-file-server-linux>



<https://groups.oist.jp/it/connecting-software-file-server-mac>

“Map network drive”

Username: OIST\tida-name

Password: *****

“Connect to Server...”

Domain: OIST

Username: tida-name

Password: *****



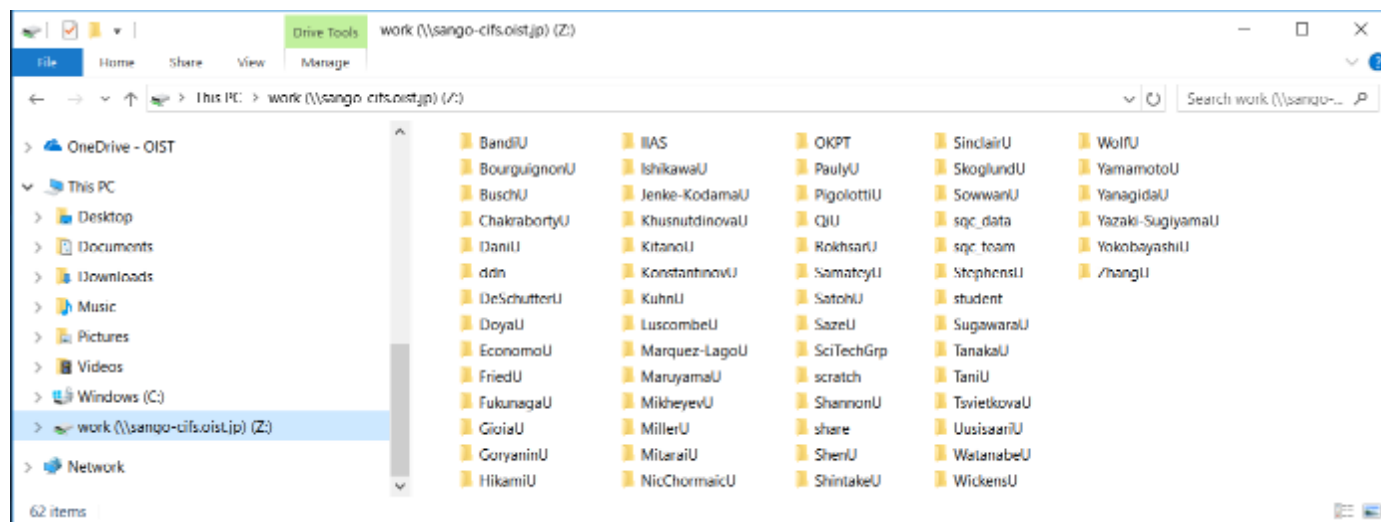
Accessing
“/work”
using
terminal

```
takeshi-tabuchi@sango-login2:/work
Using username "takeshi-tabuchi".
takeshi-tabuchi@sango.oist.jp's password:
Last login: Wed Apr 10 22:47:09 2010 from login01.oist.jp
*****
* Unauthorized access to this resource is prohibited.
* Okinawa Institute of Science and Technology.
*****

[takeshi tabuchi@sango login2 ~]$ cd /work/
[takeshi tabuchi@sango login2 work]$ ls
BandiU      DoyaU      ITAS      LuscombeU  OKPT      SazeU      SinclairU  SugawaraU  WolfU
BourguignonU  Economou  IshikawaU  Marquez-LagoU  PaulyU    SciTechGrp  SkoglundU  TanakaU    YamamotoU
BuschU      FriedU      Jenke-KodamaU  MaruyamaU  PigolottiU  scratch     SowwanU    TaniU      YanagidaU
ChakrabortyU  FukunagaU  KhusnutdinovaU  MikheyevU  QIU       ShannonU    sqc_data   TsvietkovaU  Yazaki-SugiyamaU
DaniU      GioiaU      KitanoU      MillerU     RokhsarU  share       sqc_team   UusisaariU  YokobayashiU
ddn         GoryaninU  KonstantinovU  MitaraiU    SamateyU  ShenU       StephensU  WatanabeU  ZhangU
DeSchutterU  HikamiU    KuhnU       NicChormaioU  SatohU    ShintakeU  student    WickensU

[takeshi-tabuchi@sango-login2 work]$
```

Mapping
“\work”
as a
network
drive



Only work **inside OIST network** (or using VPN).

Note: Sometimes it could lag and not update in real-time.



Licensed software:	<code>smb://software.oist.jp/licenced</code>	<code>\\software.oist.jp\licensed</code>
OIST /bucket drive:	<code>smb://bucket.oist.jp/bucket</code>	<code>\\bucket.oist.jp\bucket</code>
HPacquire drive:	<code>smb://hpacquire-ui.oist.jp/</code>	<code>\\hpacquire-ui.oist.jp</code>

Software available

To see the list of all available software on Sango:

Syntax `ml avail`
`ml spider software`

To load a software (and all its dependences/environment):

Syntax `ml software/version`

e.g. `$ ml ncbi-blast/2.10.0+`

To unload a software:

Syntax `ml unload software/name`

If you require a specific software that is not already installed in Deigo, contact it-help@oist.jp

Using Deigo to run computation jobs

Run computation jobs
directly on login node

**DON'T DO
THAT!**

Request resource
allocation for the job.
(advanced use)

salloc

Allocate resources
with the scheduler.

sbatch

Run a parallel task across
the allocated resources.

srun



Syntax `srun [options] [script] [arguments]`

To return, use: `exit`

If *srun* is invoked when no resources are allocated, it will send a requests for resource allocation:

e.g.

```
$ ml ncbi-blast/2.10.0+
$ srun -p compute -n 1 --mem=1g -o "%j.out"
  -e "%j.err" "blastn -query file.fa -remote
  -db nr -out result.out -evaluate 1e-30"
```

* Options: (-p) partition, (-n) number of tasks, (--mem=) total memory, (-o) output file, (-e) error log file.

To open an interactive session in a node:

e.g. `$ srun -p compute -c 1 --mem=1g --pty bash`

* Options: (`--pty`) connects your tty (terminal) session to a node.

To open an interactive session with a graphical interface (GUI):

e.g. `$ ssh -Y tida-login@deigo.oist.jp`
`$ ml matlab`
`$ srun -p compute -c 4 --mem=8g --x11 --pty matlab`

* Options: (`--x11`) forwards to the X11 server (graphic interface).



BASH script

(Unix Shell)



- Bash is a Unix shell and command language.
- It is the default login shell and command processor (interpreter) for most Linux distributions and macOS.
- All commands you can use in Linux terminal can be used in a BASH script and vice versa.

The first line of the file (header) indicates how the file is going to be interpreted and run, in this case by BASH shell:

```
1 #!/bin/bash
```

Bellow that, you can write your script in BASH language:

```
2 cd /your/working/directory/  
3 echo "Hello World"  
4 #etc, etc, etc
```

Save it as: **myscript.sh**

Important:

OS's represent differently the EOL (End of Line) instruction, set when pressing the “enter ↵” key.



LF (Left Feed) or `\n`



CR (Carriage Return) or `\r`



CR LF or `\r\n`

The Linux interpreter will return an error if the BASH script contains EOL different than **LF**.

If you write your script outside a Linux text editor be sure to convert all EOL into **LF**.



Important:

Linux is case sensitive.

In Bash script and Linux terminal lowercase and uppercase letters are different.



In linux, file extensions (like .sh) mean nothing, so to be able to execute the script you need to change its properties:

Syntax `$ chmod [options] permissions file`

e.g. `$ chmod u=rwx,g=rx,o=r myscript.sh`

or similarly:

```
$ chmod 754 myscript.sh
```

This means:

- “**u**ser” can “**r**ead”(4)+“**w**rite”(2)+“**e**xecute”(1)=7
- “**g**roup” can “**r**ead”(4)+“**n**ot write”(0)+“**e**xecute”(1)=5
- “**o**thers” can “**r**ead”(4)+“**n**ot write”(0)+“**n**ot execute”(0)=4

To execute the script simply type:

```
$ ./myscript.sh
```



Some people (rotation students, ...) belong to more than one group. The 'setgid' bit tells them to use the parent group ID for new directories.

Syntax `$ chmod g+s directory`

By default, we do not give write permissions to our group members. This can be changed by setting a different 'umask'.

```
$ umask 002
```

Put it in your startup script to make it automatic!

<http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html>

Some useful commands:

<code>./</code>	<code>head</code>	<code>let</code>
<code>../</code>	<code>less</code>	<code>read</code>
<code> </code>	<code>more</code>	<code>for; do; done</code>
<code>></code>	<code>seq</code>	<code>while; do; done</code>
<code>>></code>	<code>bc</code>	<code>if; then; else; fi</code>
<code>ls</code>	<code>cat</code>	<code>a=5</code>
<code>wc</code>	<code>grep</code>	<code>i=\$((\$a + 2))</code>
<code>echo</code>	<code>sed</code>	<code>i++</code>
<code>sort</code>	<code>tr</code>	<code>array=(“A” “B” “C”)</code>
<code>tail</code>	<code>awk</code>	<code>\${array[@]}</code>



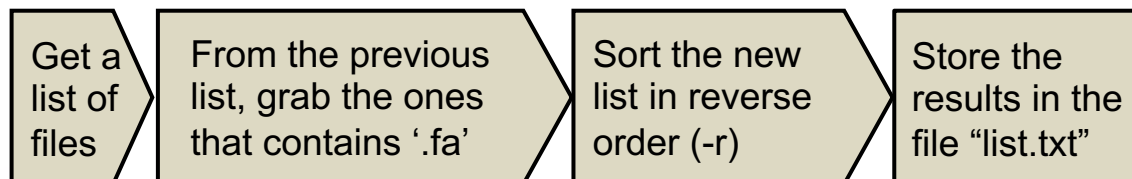
Pipelines (|) allows you to stream your work. It transfers the output of one command to the input of the next one:

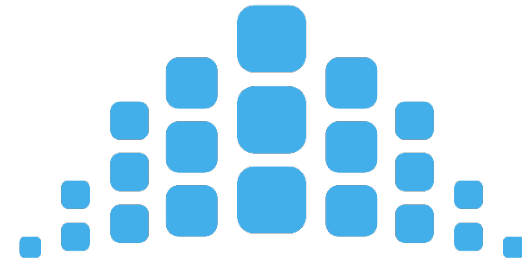
Syntax `command1 | command2 | command3`

Greater-than signs (> or >>) allows the output (stdout) of the previous command to be stored into a file, replacing the file's content (>) or just appending the data to the end (>>).

Syntax `command > file`

e.g. `$ ls | grep '.fa' | sort -r > list.txt`





Submit jobs to **slurm**

workload manager



To use the scheduler first you need to create a BASH script:

Scheduler instructions

```
1  #!/bin/bash
2
3  #SBATCH --job-name=derp_script
4  #SBATCH --partition=compute
5  #SBATCH --time=00:30:00
6  #SBATCH --mem-per-cpu=1G
7  #SBATCH --cpus-per-task=1
8  #SBATCH --ntasks=1
9  #SBATCH --mail-user=%u@oist.jp
10 #SBATCH --mail-type=BEGIN,FAIL,END
11 #SBATCH --output=job_%j.out
12 #SBATCH --error=job_%j.err
13
14 your | super | awesome | bash > script
```

Save the script as:

job_script_name.slurm



<https://groups.oist.jp/scs/getting-started>

Parameter	Value in the example	Explanation
<code>-J, --job-name=</code>	<code>test_ucsc</code>	name of the job as it will appear in the output of the <code>squeue</code> command
<code>-p, --partition=</code>	<code>compute</code>	name of the partition where the job will be executed. See " SLURM partitions in Sango " for more information about Sango partitions.
<code>-t, --time=</code>	<code>00:30:00</code>	limit on the total run time of the job. If not set the default job runtime value will be applied (see " SLURM partitions in Sango " for the list of default runtimes) Acceptable formats are: <code>days-hours:minutes:seconds</code> <code>days-hours:minutes</code> <code>days-hours</code> <code>hours:minutes:seconds</code> <code>minutes:seconds</code> <code>minutes</code>
<code>--mem-per-cpu=</code>	<code>1G</code>	minimum memory required per allocated core (the default value is 4g). This is a mandatory parameter when <code>--mem=</code> is not specified.
<code>--mem=</code>	<code>8g</code>	total real memory required per allocated node (the default value is 4g). This is a mandatory parameter when <code>--mem-per-cpu=</code> is not specified.
<code>-n, --ntasks=</code>	<code>1</code>	maximum number of tasks that the SLURM controller will launch on the allocated resources.
<code>-C, --cpus-per-task=</code>	<code>1</code>	number of CPUs (cores) per task that SLURM should allocate.
<code>--mail-user=</code>	<code>john-doe@oist.jp</code>	user to receive email notification of job state changes



To submit it to the scheduler use:

Syntax `sbatch [options] script [arguments]`

e.g. `$ sbatch job_script.slurm`

The scheduler will allocate resources according to your request, depending on the availability and priority (first-come-first-served & less demanding jobs = higher priority).

Scheduler will normally create 2 files (XXXX is the job ID):

`job_XXXX.out`

Stores every printed output
(e.g. *echo*) of the script.

`job_XXXX.err`

Contains all the error
messages.



To check the status of your jobs in the queue:

Syntax `queue [options]`

e.g. `$ queue -u ${USER}`

To cancel one (or more) job(s):

Syntax `scancel [options] [jobID[_arrayID][.stepID]...]`

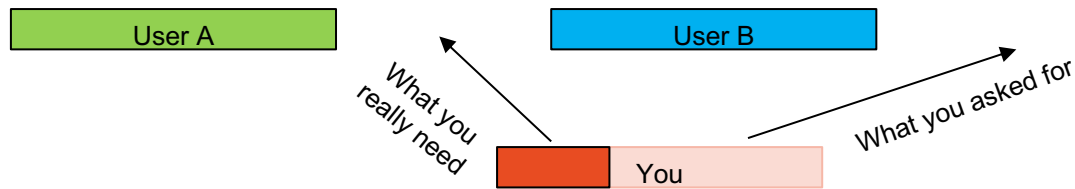
e.g. `$ scancel 1337_1 1337_2 1340 1345`

`$ scancel -t PENDING -u ${USER} -p compute`

`$ for i in {1337..1345}; do scancel $i; done`



- How much time can we use ?
 - The compute partition has a limit of time. Talk to the SCDA admins if you need to run a program for a long time ?
 - `--time` too short, and your job is cut, but
 - `--time` too long, and you wait.



- How much resources can we use ?
 - If many nodes are idle, we are wasting electricity, but,
 - if one person uses all the nodes, everybody is waiting.

Use script to submit several jobs

First create your .slurm jobs normally:

```
job_script_1.slurm  
job_script_2.slurm  
job_script_3.slurm
```

Now write your .sh script and then execute it:

Simple
way:

```
1  #!/bin/bash  
2  
3  sbatch job_1.slurm  
4  sbatch job_2.slurm  
5  sbatch job_3.slurm
```

Slightly
better
way:

```
1  #!/bin/bash  
2  
3  for i in job_*.slurm ; do  
4      sbatch $i  
5  done
```



Method 2: sbatch array

```
1  #!/bin/bash
2
3  #SBATCH --job-name=ArrayJob
4  #SBATCH --output=job_%j_%A-%a.out
5  #SBATCH --error=job_%j_%A-%a.err
6  #SBATCH --partition=compute
7  #SBATCH --mem=1g
8  #SBATCH --array=1-6%3
9
10 Arr=(0.05 0.075 0.1 0.25 0.5 1.0)
11
12 echo ${Arr[$(( $SLURM_ARRAY_TASK_ID
    % ${#Arr[@]} ))]}
```

This command will simply print one of the elements of the array “Arr” depending on the ID of the current task.

This will run an array of 6 jobs (indexes 1 to 6), but limited to 3 simultaneous jobs.

```
$SLURM_JOBID=1337
$SLURM_ARRAY_JOB_ID=1337
$SLURM_ARRAY_TASK_ID=1
```

```
$SLURM_JOBID=1338
$SLURM_ARRAY_JOB_ID=1337
$SLURM_ARRAY_TASK_ID=2
```

...

* If you include *srun* in the script, a new job step will be created to run it within the task.

```
%j ≈ $SLURM_JOBID
%A ≈ $SLURM_ARRAY_JOB_ID
%a ≈ $SLURM_ARRAY_TASK_ID
%s ≈ $SLURM_ARRAY_TASK_STEP
```



Method 3: auto-create the jobs from template

SKILLPILLS

Create your template:



Create your script:

```
1 #!/bin/bash
2
3 #SBATCH --job-name=Job_VAR
4 #SBATCH --output=job_%j.out
5 #SBATCH --error=job_%j.err
6 #SBATCH --partition=compute
7 #SBATCH --mem=4g
8
9 a=VAR
10
11 grep ">" $a.fa > $a.out
```

```
1 #!/bin/bash
2
3 arr=("f_1" "f_3" "f_4")
4
5 for i in ${arr[@]}; do
6     cat template.slurm |
7     sed s/'VAR'/$i/g >
8     job_$i.slurm
9     sbatch job_$i.slurm
10 done
11
```

Save it:

template.slurm

Save and execute:

script.sh

Create your script:

```
1  #!/bin/bash
2
3  arr=("f_1" "f_3" "f_6")
4  for i in ${arr[@]}; do
5      echo '#!/bin/bash' > job_${i}.slurm
6      echo '#SBATCH --job-name=Job_'${i} >> job_${i}.slurm
7      echo '#SBATCH --output=job_%j.out' >> job_${i}.slurm
8      echo '#SBATCH --error=job_%j.err' >> job_${i}.slurm
9      echo '#SBATCH --partition=compute' >> job_${i}.slurm
10     echo '#SBATCH --mem=4g' >> job_${i}.slurm
11     echo 'grep ">" '${i}'.fa > '${i}'.out' >> job_${i}.slurm
12     echo '' >> job_${i}.slurm
13     sbatch job_${i}.slurm
14 done
```

Save and execute: **script.sh**

