# Centro de Computación de Alto Rendimiento

# CCAR_UNED

## CASPER – Manual de usuario

Versión 1.3

#SOMOS2030

https://ccar.uned.es

**CLUSTER CASPER – Manual de Usuario (v.1.3)**



*Centro de Computación de Alto Rendimiento CCAR*

Julio 2024

# Sumario

# Introducción

El Centro de Computación de Alto Rendimiento de la UNED (**CCAR UNED**) tiene como misión proveer de la infraestructura y los recursos necesarios para investigadores que requieran el uso de herramientas de computación de alto rendimiento (HPC).

Su uso está destinado exclusivamente a labores de investigación y explotación de recursos computacionales en el ámbito de las Ciencias. El **CCAR** ofrece servicios de asesoría, adquisición de equipos e infraestructuras, soporte y mantenimiento de hardware y software de altas prestaciones.

# Estructura y datos técnicos

**CASPER** es un clúster del **CCAR** orientado a investigadores, cuenta con un nodo maestro (*front-end* o nodo de *login*) que administra toda la comunicación entre los nodos de cálculo: envía trabajos, gestiona las cuotas de disco de los usuarios y permite instalar fácilmente software, sirviéndolo a los nodos de cálculo mediante instalación en los propios discos o a través de directorios compartidos por NFS.

El software en el que se basa **CASPER** es OpenHPC v2.6, una suite enfocada a entornos HPC instalada sobre Rocky 8.9, una distribución GNU/Linux 100% compatible con Red Hat Enterprise Linux (RHEL). **CASPER** usa OpenPBS (v23.06) como gestor de colas.

**CASPER** dispone de dos redes de comunicación dedicadas. La primera de ellas (192.168.10.0/24) es una red de 10 GbE BaseT gestionada por switches Netgear M4300, que sirve el espacio de usuario (el directorio `$HOME`) a todo el clúster. El sistema de almacenamiento en red está basado en BeeGFS (v7.4.1), y cuenta con un servidor de gestión, dos servidores de metadatos con discos SSD en configuración RAID1 y dos servidores de almacenaje con discos SAS en configuración RAID6. En total, se dispone de 128 TB para almacenamiento. La segunda red de comunicación (172.16.10.0/24) es una red 1 GbE gestionada por switches HPE 1960S, que permite la comunicación y el envío de trabajos entre el *front-end* y los nodos de cálculo, así como el uso de MPI en cálculos inter-nodos.

El resto de **CASPER** lo conforman los nodos de cálculo (c-x-x), organizados en diferentes colas de trabajo, dependiendo de las especificaciones de los nodos y los grupos de investigación asociados. En función del tipo de trabajo a realizar, y de la adscripción de un usuario a un grupo de trabajo concreto, es recomendable utilizar una cola u otra. En general, se desaconseja trabajar con nodos de cálculo concretos, siendo preferible referirse siempre a la cola.

# Cómo trabajar con CASPER

**CASPER** es el clúster de supercomputación del **CCAR** orientado a investigadores, está pensado para ejecutar trabajos que tengan un alto coste computacional o requieran una paralelización intensiva. Dispone de nodos monoprocesador de alta frecuencia y nodos multiprocesadores de hasta 128 hilos para cálculos en paralelo. Además, dispone de más de 30 tarjetas gráficas para computación mediante GPUGP.

## Módulos

Dada la diferente casuística existente, **CASPER** se organiza en módulos de trabajo, permitiendo que el usuario active, bajo demanda, aquellos módulos que necesite para computar.

Esencialmente, al activar un módulo cargamos en memoria un conjunto de variables que enlazan a programas o compiladores determinados y enlazamos a sus librerías correspondientes. Podemos, por ejemplo, cargar el módulo IntelMPI para que por defecto el compilador sea el propietario de Intel (`icc` o `ifort`), las llamadas a mpiexec sean a través de Intel y carguemos en memoria (en `$LD_LIBRARY_PATH`) el conjunto de librerías asociadas a esta suite.

Pueden obtenerse todos los módulos disponibles a través del comando `module avail`:

*Código*

```
[username@c-X-Y ~]$ module avail

------------------------ /opt/ohpc/pub/modulefiles -----------------------------
OpenFOAM/2016     eigen/3.4.0       lammps-intel/2Aug23  prun/2.2
R/4.2.1           gaussian/16       libfabric/1.19.0     python/3.11.1
Rust/1.74.0       gnu12/12.3.0  (L) ohpc                 siesta-intel/rel-max-3
autotools         gnu9/9.4.0        os                   singularity/3.7.1
charliecloud/0.15 hwloc/2.7.2       papi/5.7.0           ucx/1.15.0
cmake/3.24.2      intel/2023.2.1    papi/6.0.0       (D) valgrind/3.19.0
cp2k/2023.1       lammps-gnu12-openmpi4/2Aug23           pmix/4.2.9
```

```
  Where:
   D:  Default Module

If the avail list is too long consider trying:

"module --default avail" or "ml -d av" to just list the default modules.
"module overview" or "ml ov" to display the number of modules for each name.

Use "module spider" to find all possible modules and extensions.
Use "module keyword key1 key2 ..." to search for all possible modules matching
any of the "keys".
```

Al cargar un módulo concreto (por ejemplo, un compilador), el listado de módulos disponibles aumenta, mostrando todos los programas que hace uso del módulo ya cargado:

*Código*

```
[username@c-X-Y ~]$ module load gnu12/12.3.0
[username@c-X-Y ~]$ module avail

------------------------ /opt/ohpc/pub/moduledeps/gnu12 -----------------------
gsl/2.7.1         metis/5.1.0         netcdf-fortran/4.6.0    pdtoolkit/3.25.1
hdf5/1.10.8       mpich/3.4.3-ofi     netcdf/4.9.0            plasma/21.8.29
impi/2021.10.0    mvapich2/2.3.7      openblas/0.3.21         py3-numpy/1.19.5
likwid/5.2.2      netcdf-cxx/4.3.1    openmpi4/4.1.6          scotch/6.0.6

------------------------ /opt/ohpc/pub/modulefiles ----------------------------
OpenFOAM/2016     eigen/3.4.0       lammps-intel/2Aug23  prun/2.2
R/4.2.1           gaussian/16       libfabric/1.19.0     python/3.11.1
Rust/1.74.0       gnu12/12.3.0  (L) ohpc                 siesta-intel/rel-max-3
autotools         gnu9/9.4.0        os                   singularity/3.7.1
charliecloud/0.15 hwloc/2.7.2       papi/5.7.0           ucx/1.15.0
cmake/3.24.2      intel/2023.2.1    papi/6.0.0       (D) valgrind/3.19.0
cp2k/2023.1       lammps-gnu12-openmpi4/2Aug23           pmix/4.2.9

  Where:
   D:  Default Module
   L:  Module is loaded

If the avail list is too long consider trying:

"module --default avail" or "ml -d av" to just list the default modules.
"module overview" or "ml ov" to display the number of modules for each name.

Use "module spider" to find all possible modules and extensions.
Use "module keyword key1 key2 ..." to search for all possible modules matching
any of the "keys".
```

Una vista pormenorizada de los módulos, incluyendo una descripción y las dependencias, puede obtenerse con **`module spider`**:

*Código*

```
[username@c-X-Y ~]$ module spider
----------------------------------------------------------------------------
The following is a list of the modules and extensions currently available:
----------------------------------------------------------------------------
EasyBuild: EasyBuild/4.8.2
Software build and installation framework
Gaussian: Gaussian/16
OpenFOAM: OpenFOAM/2016
OpenFOAM is a free and open-source software used for CFD simulation. To
compile OpenFOAM.
R: R/4.1.2, R/4.2.1, R/4.3.1
R is a language and environment for statistical computing and graphics (S-Plus
like).
Rust: Rust/1.74.0
```

Con el comando **`module spider módulodeseado`**, se puede obtener información acerca del programa y del compilador necesario para su carga:

*Código*

```
[username@c-X-Y ~]$ module spider lammps
----------------------------------------------------------------------------
lammps: lammps/2Aug2023-inteloneapi
----------------------------------------------------------------------------
Description:
Lammps package management
You will need to load all module(s) on any one of the lines below before the
"lammps/2Aug2023-inteloneapi" module is available to load.
intel/2023.2.1
```

**IMPORTANTE**: En muchas ocasiones es necesario, antes de cargar un módulo, cargar previamente la cadena de herramientas (*toolchain*) con la que se ha compilado el módulo de interés (gnu12, Intel, etc.), tal y como se indica al hacer **`module spider módulodesado`**.

El listado de módulos activos para un usuario en una sesión determinada puede obtenerse con **`module list`**:

*Código*

```
[username@c-X-Y ~]$ module list

Currently Loaded Modules:
1) autotools   2) prun/2.2   3) gnu12/12.3.0   4) hwloc/2.7.2   5) ucx/1.15.0
6) libfabric/1.19.0   7) openmpi4/4.1.6   8) ohpc
```

Los módulos pueden activarse en el espacio de trabajo de un usuario con **`module load`** y desactivarse con **`module unload`**:

*Código*

```
[username@c-X-Y ~]$ module list
Currently Loaded Modulefiles:
   1) autotools
[username@c-X-Y ~]$ module load gnu12/12.3.0
[username@c-X-Y ~]$ module list
Currently Loaded Modulefiles:
  1) autotools   2) gnu12/12.3.0
[username@c-X-Y ~]$ module unload autotools
[username@c-X-Y ~]$ module list
Currently Loaded Modulefiles:
   1) gnu12/12.3.0
```

## Colas de trabajo

**CASPER** posee un total de 932 hilos de computación accesibles al sistema, repartidos en cinco colas de trabajo diferentes (un listado actualizado de colas e hilos puede encontrarse en el sitio web **https://ccar.uned.es/hpc**):

- alpha: Tres nodos de computación híbridos en CPU/GPU, con un total de 36 hilos de computación por CPU y 128 GB de RAM por nodo. Esta cola cuenta con tres GTX 1080 Ti y cuatro RTX 2080 Ti como GPUs.
- beta: Seis nodos de computación híbridos CPU/GPU, con 288 hilos de computación por CPU y 128 GB de RAM por nodo. Esta cola cuenta con nueve GTX 1080 Ti y seis RTX 3080 como GPUs.

- gamma: Trece nodos de computación multinúcleos, con un total de 528 hilos de computación por CPU y 128 GB de RAM por nodo.
- delta: Un nodo multinúcleo con 64 hilos de computación por CPU y 192 GB de RAM.
- epsilon: Un nodo multinúcleo con 64 hilos de computación por CPU y 192 GB de RAM. Este nodo cuenta con dos GTX 1080 Ti.

Los usuarios, al ser dados de alta, adscriben su pertenencia a un grupo de trabajo específico. Según el grupo de trabajo al que pertenezca el usuario, podrá acceder a un número específico de recursos.

## Establecer el método de trabajo

Todos los trabajos se envían mediante un sistema de colas (PBS) gestionado por OpenPBS. El usuario, tras iniciar sesión en **CASPER**, puede subir código a su directorio de usuario (a través de `scp`), editarlo (están instalados los editores vim, vi y nano), compilarlo, etc. Todo ello desde el *front-end* del sistema. Debe evitarse el uso de los nodos de cálculo para este tipo de tareas.

**IMPORTANTE:** Solo en caso de querer optimizar la compilación de un código para una arquitectura específica (CPU o GPU), puede ser recomendable realizar estas tareas haciendo uso de una sesión interactiva en un nodo de cálculo. En general, todo el software se encuentra ya optimizado para cada nodo de cálculo, por lo que este paso no suele ser necesario.

Una vez preparado el código, el procedimiento general de trabajo consiste en enviarle a OpenPBS (el gestor de recursos) una petición con los recursos solicitados y las instrucciones a ejecutar. Para asegurarse de la correcta ejecución del código de forma desatendida, es recomendable hacer antes una reserva interactiva en un nodo de cálculo, seleccionando la cola donde queramos ejecutar nuestro trabajo (`-q COLA`):

*Código*

```
[username@casper ~]$ qsub -I -q alpha
-------------------------------------------------------------------------
qsub: waiting for job 2200.casper to start
qsub: job 2200.casper ready

[username@c-X-Y ~]$
```

Una vez dentro del nodo es recomendable comprobar qué módulos deben cargarse para la correcta ejecución y compilación del software, así como establecer las instrucciones concretas y el orden correcto en el que deben realizarse.

Comprobada la correcta ejecución desde una sesión interactiva, podemos salir de ella con el comando **`exit`** y, desde el *front-end*, crear un *script* de trabajo para su ejecución desatendida.

Este script (por ejemplo, **`trabajo.job`**) especifica las instrucciones para OpenPBS y los comandos a ejecutar. Las instrucciones específicas de OpenPBS comienzan con la sentencia **`(#PBS`**). Los comentarios (ignorados por OpenPBS) contienen dos almohadillas (**`##`**):

*Código*

```
[username@casper ~]$ cat trabajo.job

#!/bin/bash

################################################
##  Configuración del trabajo                 ##
################################################

## Nombre del trabajo
#PBS -N prueba

## Nombre del log de salida
#PBS -o prueba.out

## Nombre del log de error
#PBS -e prueba.err

## Especifica a que cola se mandará el trabajo
#PBS -q gamma

## Reserva de recursos
## select     = número de nodos (se recomienda usar 1)
## ncpus      = número total de hilos a usar
## mpiprocs   = número de hilos de MPI en paralelo (se recomienda usar ncpus)
## ompthreads = número de hilos de OMP en paralelo (se recomienda usar 1)
##              Debe cumplirse la relación ncpus = mpiprocs * ompthreads
## diskspace  = espacio en disco necesario (se especifica en mb, gb, tb, etc.
##              Si se requieren menos de 100 GB, se usará un disco SSD de alta
##              velocidad, si se requiere más espacio, se usará un disco HDD de
##              alta capacidad.

#PBS -l select=1:ncpus=4:mpiprocs=4:ompthreads=1:diskspace=50gb
```

```
## Asigna el directorio temporal de trabajo en función del espacio requerido
source /usr/local/jobs/scratch.sh

###############################################
##  Copia de archivos al nodo de trabajo      ##
###############################################

## Copia los ficheros necesarios al directorio temporal
cp $PBS_O_WORKDIR/f1    $TMPDIR
cp $PBS_O_WORKDIR/f2    $TMPDIR
cp -R $PBS_O_WORKDIR/d1 $TMPDIR

# Ir al directorio temporal del nodo de trabajo
cd $TMPDIR

###############################################
##  Ejecución del trabajo                     ##
###############################################

## Cargar los módulos necesarios
## module unload MODULO_NO_DESEADO
## module load MODULO_DESEADO

## Trabajo en serie
## ./MiPrograma

## Trabajo en paralelo con OpenMP
## ./OMPTest

## Trabajo en paralelo con MPI
## El número de hilos debe coincidir con la variable mpiprocs
## mpirun -np 4 ./MPIexample

###############################################
##  Copia de archivos al $home del usuario    ##
###############################################

## Copia todos los archivos generados en el Directorio temporal
## al directorio desde el que se envió el trabajo
OUTDIR=$PBS_O_WORKDIR/$PBS_JOBID
mkdir -p $OUTDIR
cp -r * $OUTDIR
```

**IMPORTANTE:** Los comandos usados para copiar ficheros a y desde el directorio temporal al directorio del usuario son ejemplos que deben personalizarse en función de las necesidades.

En caso de querer usar una combinación de hilos con MPI y OpenMP puede usarse la línea:

```
#PBS -l select=1:ncpus=8:mpiprocs=4:ompthreads=2:diskspace=50gb
```

Esta combinación realizará una reserva de 8 hilos de computación, utilizando 2 hilos de OpenMP por cada uno de los 4 hilos de MPI disponibles.

En caso de querer especificar un nodo en concreto se puede complementar la reserva con el comando **host**:

```
#PBS -l select=1:ncpus=8:mpiprocs=4:ompthreads=2:host=c-0-2:diskspace=50gb
```

Una vez configurado el archivo de envío **trabajo.job**, este debe enviarse a OpenPBS con el comando **qsub**:

*Código*

```
[username@casper ~]$ qsub trabajo.job
76.casper
```

Lo que nos devuelve un identificador del trabajo en ejecución, **76.casper**. Si, por algún motivo, se quiere cancelar la ejecución del trabajo, se usa el comando **qdel**, indicando el identificador del trabajo (en este ejemplo, 76):

*Código*

```
[username@casper ~]$ qdel 76
```

## Comprobar el estado de un trabajo

Una vez mandado un trabajo a la cola de envío, este queda en estado **Q** (en cola, *queue*) hasta que el gestor encuentra disponibilidad para lanzarlo. En ese momento pasa a un estado **T** (*transferring*), hasta que el nodo de cálculo toma el control del trabajo y lo ejecuta, entrando en estado **R** (*running*). Los distintos estados pueden comprobarse ejecutando el comando **qstat**:

*Código*

```
[username@casper]$ qstat
Job id            Name             User             Time Use S Queue
----------------  ---------------- ----------------  -------- - -----
214.casper        prueba           username          00:00:00 R GAMMA
```

`qstat` muestra, de izquierda a derecha:

- Job ID — El número identificador del trabajo.
- Name — Nombre del trabajo.
- User — El usuario que ha lanzado el trabajo.
- Time Use — Tiempo asignado a dicho trabajo.
- S — El estado del trabajo
  - E — El trabajo ha finalizado.
  - H — El trabajo está en espera (por el usuario o por un administrador).
  - Q — El trabajo está en cola o apto para ejecutarse.
  - R — El trabajo está ejecutándose.
  - T — El trabajo se está transfiriendo al nodo de trabajo.
  - W — El trabajo está en espera para ejecutarse.
  - S — El trabajo está suspendido.
- Queue — La cola donde está corriendo el trabajo.

**IMPORTANTE:** Si por algún motivo el trabajo no pudiera ejecutarse, quedaría en estado `Q` indefinidamente, indicando con `qstat -f`, al final, la razón (en "`comment =`"). Si la razón fuera, por ejemplo, que el usuario no tiene permisos para lanzar trabajos en la cola que ha especificado, el trabajo no podría lanzarse y aparecería el siguiente mensaje:

*Código*

```
[username@casper]$ qsub job.mpi
qsub: Unauthorized Request
```

Para una visión pormenorizada del estado de **CASPER** puede usarse el comando `queue`, el cual muestra información de las colas disponibles con sus respectivos nodos:

*Código*

```
[username@casper ~]$ queue

queuename                Used/Tot
```

```
--------------------------------------------------------------------------------
ALPHA@c-0-2             0/12
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
GAMMA@c-1-11            0/40
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
GAMMA@c-1-12            0/48
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
GAMMA@c-1-13            0/48
--------------------------------------------------------------------------------
```

Como se puede ver en el código, el comando nos mostrará una visión general de las colas de trabajo disponibles y los nodos totales del clúster. También podremos observar el número de hilos en uso y total del nodo (0/12,0/40, 0/48, etc.).

Si un usuario decide enviar varios trabajos, puede saber dónde se encuentran con el comando **`qstat -n1`**:

*Código*

```
[username@casper]$ qstat -n1

casper:

                                                                  Req'd  Req'd   Elap
Job ID          Username Queue    Jobname       SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
244.casper      username GAMMA    prueba      83369.   1  16    --    --  R 00:30 c-1-13/1*16
248.casper      username GAMMA    prueba      40241.   1  48    --    --  R 00:18 c-1-12/0*48
```

En este ejemplo, los trabajos están corriendo en la cola GAMMA, en los nodos c-1-13 y c-1-12, habiendo reservado 16 hilos para el trabajo 244 y 48 para el trabajo 248.

# Resumen de comandos

## Comandos de módulos

| | |
|---|---|
| `module avail` | Ver los módulos disponibles |
| `module list` | Ver los módulos cargados actualmente |
| `module load + módulo` | Cargar un módulo |
| `module unload + módulo` | Descargar un módulo |
| `module spider` | Ver módulos disponibles de forma detallada |

## Comandos de trabajos

| | |
|---|---|
| `qsub nombre_trabajo` | Lanzar un trabajo |
| `qdel ID_trabajo` | Cancelar un trabajo |
| `qstat / qstat -n1` | Ver el estado de un trabajo |
| `pbsnodes -a` | Ver información de las colas |
| `queue` | Ver estado colas de trabajo |