計算機利用の手引き

■ **[最終更新日時] 2024年1月30日**

English versionもあります。

よくある質問集

目次

- 接続
  - RCCSのコンピューターへのログイン
  - ssh公開鍵とウェブページ用パスワードの登録
  - ログインシェル
- RCCSシステムの全体像
- キューイングシステム関連
  - キューイングシステムの全体像
  - キュー構成
  - CPU点数とキュー係数
  - リソース集計
  - ユーザ別 リソース制限設定／表示
  - ジョブの投入
  - ジョブの状態表示
  - ジョブの取り消し
  - ジョブのホールドとリリース
  - 実行済みジョブの情報取得
- ビルドと実行
  - ビルドのコマンド
  - 並列プログラムの実行方法
  - Environment Modules
- パッケージプログラム
- その他のRCCS固有コマンド
  - ジョブ関連
  - ジョブの実行開始時間予測(waitest)
  - 資源使用状況表示
  - ジョブスクリプト用ユーティリティコマンド
  - 計算ノード上のファイル操作
- 問い合わせ

接続

RCCSのコンピューターへのログイン

RCCSのスパコンで計算を流すためにはログインサーバー(ccfep.ims.ac.jp)にログインする必要があります。

- ログインにはsshの公開鍵認証を使用します。利用申請が許可され、アカウントが有効化されたらクイックスタートガイドを参考に公開鍵を登録してください。
- メンテナンス中はログイン出来ません。→定期メンテナンス情報（メンテナンス日は原則毎月第一月曜日）
- ログインサーバーへのログインは、日本に割り当てられたIPv4アドレスを持つホストまたは許可されたホストからでなければなりません。→日本国外からの接続について

ssh公開鍵とウェブページ用パスワードの登録

sshの公開鍵と秘密鍵のペアを最初に作成しておきます。作成方法が不明な方はインターネットなどで調べてください。
クイックスタートガイドのページにも情報があります。

初めて登録する場合もしくはウェブページ用パスワードを忘れた場合

1. 専用ウェブページの「登録案内メールの発行」をウェブブラウザーで開きます。
2. 利用申請書で申請したメールアドレスを入力後、「登録案内メール送信」ボタンを押します。
3. 自動送信されたメールの中に記載されているURLをウェブブラウザーで開きます。
4. ユーザー限定ページにアクセスするために、新たに設定したいパスワードを2ヶ所に入力します。
5. あらかじめ用意したsshの公開鍵をペーストなどして入力します。

6. 「保存」ボタンを押します。

ウェブページ用パスワードを使う場合

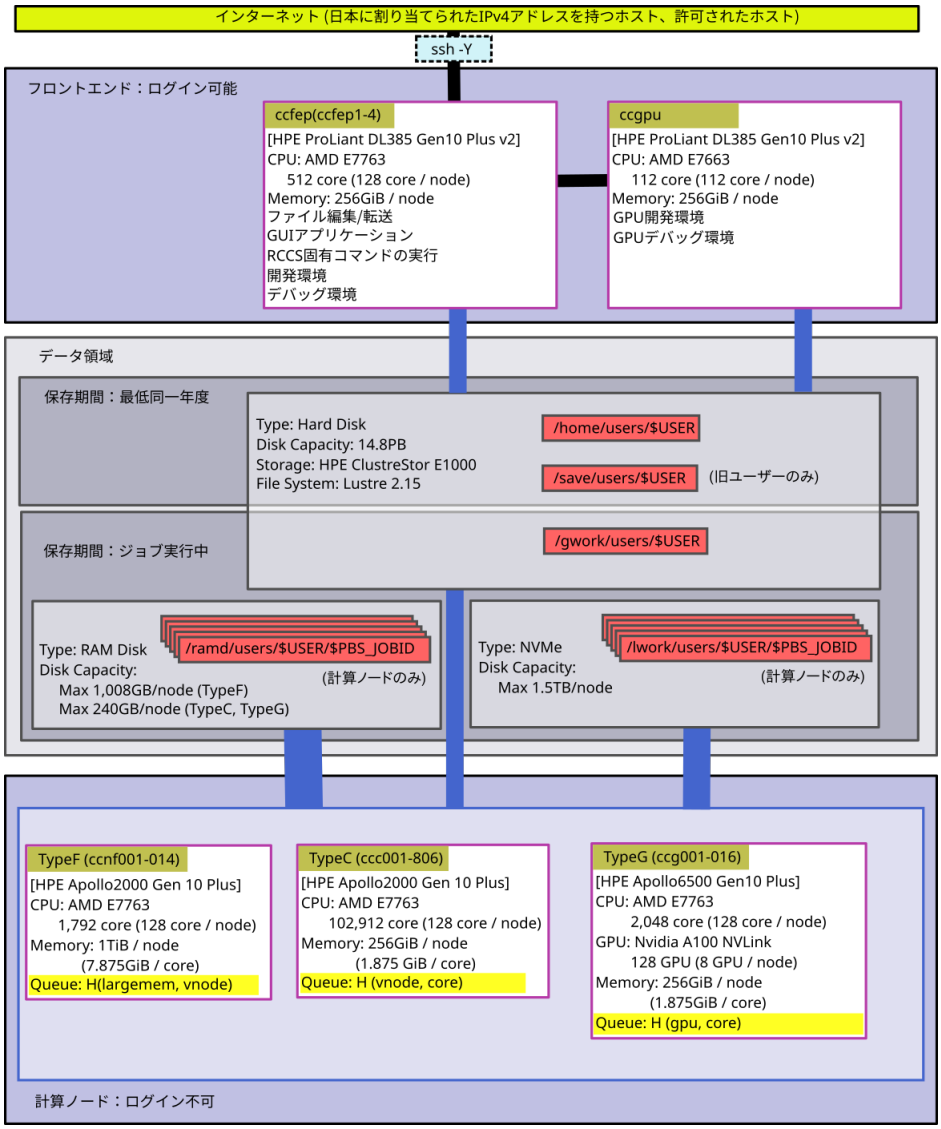
1. 専用ウェブページ(https://ccportal.ims.ac.jp/account/)をウェブブラウザーで開き、ユーザー名とパスワードを入力し、「ログイン」ボタンを押します。
2. 画面右上の「アカウント情報」を開きます。
3. 「編集」タブを押します。
4. パスワードを変更するは、現在のパスワードと新たに設定したいパスワードを入力します。
5. あらかじめ用意したsshの公開鍵をペーストなどして入力します。
6. 「保存」ボタンを押します。

ログインシェル

- /bin/bash、/bin/tcsh、/bin/zshを利用できます。
- 変更はssh公開鍵と同じウェブページで行ないます。変更が反映されるまで時間がかかる場合があります。
- .loginや.cshrcはカスタマイズしても構いませんが、十分な注意が必要です。

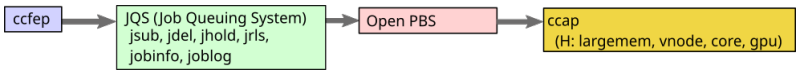
RCCSシステムの全体像

- いわゆる会話処理ができるのは、ccfep (4ノード) です。ビルドやデバッグにご利用下さい。
- 保存期間が異なるディスクが、/home、/save、/gwork、/lwork(演算サーバーのみ)、/ramd(演算サーバーのみ)の5種類あります。
- アクセス速度は、下図の太い線の方が高速です。
- /gwork, /lwork, /ramdは計算途中の一時ファイルを置くのに使います。ジョブ終了後に削除されます。
- /lworkはNVMe SSDです。容量は約1.5TBあり、コアあたり11.9GB使えます。
- /ramdは容量が240GBもしくは1,008GB程度のラムディスクです。プログラムで使うメモリー量とラムディスクで使う容量の総計はキューイングシステムによって管理されています。
- /homeと/saveは現在全く同様に扱われています。容量制限についても合算値で判断されます。(歴史的な経緯で名前だけが異なっています。)
- /tmp、/var/tmp、/dev/shmの一時ディレクトリーの使用を禁止します。一時ディレクトリーを使用しているジョブは見つけ次第削除しますので予めご了承ください。
- インターコネクトにはInfiniBandが使われています。



キューイングシステム関連

キューイングシステムの全体像



キュー構成

ノードジョブは 64 コアの vnode (仮想ノード)を利用単位とします。(各ノードは 2 つの vnode に分けられます。)

■ 全利用者が利用可能なキュー

キュー名 (jobtype)	計算ノード	メモリー	利用単位	ジョブあたりの制限	1グループの制限			vnode 数 (コア数)
					割当点数	コア数/GPU数	ジョブ数	
H (largemem)	TypeF	7.875GB/core	vnode ノード	1～14vnode (64～896コア)	720万点以上 240万点以上 72万点以上 24万点以上 24万点未満	9,600/64 6,400/42 4,096/28 3,200/12 768/8	1,000	28 vnode (1,792 コア)
H (vnode)	TypeC	1.875GB/core	vnode ノード	1～50vnode (64～3,200コア)				1,248 vnode 以上 (79,872 コア以上)
H (core)	TypeC	1.875GB/core	core	1～63コア				200 vnode 以上 (12,800 コア以上)
H (gpu)	TypeG	1.875GB/core	core	1～48GPU 1～16コア/GPU				32 vnode (2,048 コア 128 GPU)

- core 単位のジョブ(ncpus<64 及び GPU ジョブ)と jobtype=largemem については別途コア数制限が課されます。jobinfo -s で制限値を確認可能です。

- ジョブの最大時間は、定期メンテナンスまでとなります。ただし、1週間を越えるジョブが実行できるノードは全体の半分程度とします。
- largemem 以外の jobtype については指定したリソースより判定可能であるため、省略可能です。
- TypeC の 80 ノード(160 vnode)は vnode ジョブと core ジョブ共存領域となります。
- 短期の vnode ジョブが largemem 用の領域で実行される場合があります
- 短期の core ジョブが gpu 用ノードで実行される場合があります。
- グループ制限を判断する点数には追加点数を含みません。

■ 別途申請が必要なキュー

キューの設定は下記の通りです。

キュー名	計算ノード	制限時間	メモリー	1ジョブあたりのコア数	グループ制限
専有利用	TypeC	7日間単位	1.875GB/core	応相談	許可されたコア数

CPU点数とキュー係数

CPU点数は、CPUやGPUを使うことによって減ります。
消費点数はシステム毎に設定されているCPUキュー係数とGPUキュー係数により求められます。

jobtype	CPUキュー係数	GPUキュー係数
largemem	60 点 / (1 vnode * 1時間)	-
vnode	45 点 / (1 vnode * 1時間)	-
core	1 点 / (1コア * 1時間)	-
gpu	1 点 / (1コア * 1時間)	60 点 / (1GPU * 1時間)

- 会話処理ノードの内、ccfepはCPU時間でCPU点数が消費されます。
- 他のシステムは、経過時間でCPU点数が消費されます。
- 実際の費用は無料です。

現在の使用CPU点数、残りのCPU点数を知るためには、showlim -cコマンドを使います。

リソース集計

- キューイングシステムで実行されたジョブの集計とディスク使用量の集計は10分毎に行います。
- 会話処理の集計は、毎日2:20に行ないます。
- CPU点数を使いきると、グループ内利用者全員の全ての実行中ジョブは削除され、新たなジョブ投入が抑止されます。
- ディスク使用量が制限値を越えると、新たなジョブ投入が抑止されます。

ユーザ別 リソース制限の設定／表示

ウェブブラウザーでリソース制限設定ページにアクセスします。
- 代表利用者のみがリグループ内のメンバー個々に対してグループに割り当てられたリソース量を限度に制限値を設定できます。
- 一般利用者はリソース制限の値を確認することができます。
- リソース制限の種類は、CPU数、点数、ディスク容量です。

ジョブの投入

下記の２つの方法があります。

- jsubコマンドにスクリプトファイルを指定してジョブを投入する。
- g09sub / g16subコマンドにgaussianのインプットファイルを指定してジョブを簡単に投入する（gaussian専用）。

以下はjsubを使ってジョブを投入する方法です。

ジョブスクリプトの作成

※ゼロからジョブスクリプトを作成するよりも、センター側で用意したサンプルをベースに適宜修正した方が効率的かもしれません。
これらの方法についてはクイックスタートガイドページのジョブ投入ガイドの項目が参考になるかと思います。

■ ヘッダー部の書き方

スクリプトの最初にバッチコマンドを記述する必要があります。csh, bash (/bin/sh), zsh でのジョブ投入が可能です。

意味	ヘッダー部	重要度
シェル	(csh の場合) #!/bin/csh -f (bash の場合) #!/bin/sh (zsh の場合) #!/bin/zsh	必須 (どれか一つ)

使用CPU数	#PBS -l select=[*Nnode*:]ncpus=*Ncore*:mpiprocs=*Nproc*:ompthreads=*Nthread*[:jobtype=*Jobtype*][:ngpus=*Ngpu*]
	• Nnode: vnode数またはノード数
	• Ncore: vnodeまたはノードあたりの確保するコア数
	  ◦ largemem, vnode: 64(vnode単位)または128(ノード単位)
	  ◦ core: 63以下のコア数
	• Nproc: vnodeあたりのプロセス数
	• Nthread: プロセスあたりのスレッド数
	• Jobtype: largemem
	  ◦ 2022年度の更新でlargemem以外の場合は省略できるようになりました。
	• Ngpu: 使用するGPUの数
		必須

時間制限	#PBS -l walltime=*72:00:00*	必須
ジョブの開始前後にメールで通知	#PBS -m abe	オプション
ジョブの再実行抑止	#PBS -r n	オプション
バッチジョブ投入ディレクトリへの移動	cd ${PBS_O_WORKDIR}	推奨

「使用CPU数」行の書き方の例

例**1: 5** ノードまるごと使う場合**(640 (128*5)** コア**, 320 (64*5) MPI)**

```
#PBS -l select=5:ncpus=128:mpiprocs=64:ompthreads=2
```

例**2: 64** コアごとに **10** 個確保するパターン**(640 (64*10)**コア**, 320(32*10) MPI)**

```
#PBS -l select=10:ncpus=64:mpiprocs=32:ompthreads=2
```

例**3:16** コアジョブ**(16 MPI)**

```
#PBS -l select=1:ncpus=16:mpiprocs=16:ompthreads=1
```

例**4:16** コア **(16 OpenMP), 1 GPU** ジョブ

```
#PBS -l select=1:ncpus=16:mpiprocs=1:ompthreads=16:ngpus=1
```

注意: ノードあたりの GPU 数は 8 で、GPU あたりの CPU コア数(ncpus/ngpus)は 16 以下である必要があります。

例**5:64** コア、大容量メモリ**(**約 **500 GB)**

```
#PBS -l select=1:ncpus=64:mpiprocs=32:ompthreads=2:jobtype=largemem
```

このジョブタイプだけは jobtype=largemem の指定が必須です。

ジョブの投入コマンド

ジョブスクリプトが準備できたら、jsubコマンドを用いてジョブを投入します。

```
ccfep% jsub [-q HR[0-9]] [-gXXX] [-W depend=(afterok|afterany):JOBID1[:JOBID2...]] script.csh
```

- サーバ負荷を減らすため、短期間に大量のジョブを実行されている場合ペナルティが課されます。
  - 一日に数千ジョブ、のような規模で実行するような場合は、ジョブをまとめるようお願いします。
- 2022年度の更新で-q H は不要になりました。
- 計算物質科学スパコン共用事業利用枠としてジョブ投入する場合は、-gオプションをつけます（XXXは計算物質科学スパコン共用事業利用枠のグループ名）。
- ジョブの依存関係を-Wオプションで設定できます。正常終了後に実行させる場合はafterok、異常終了後でも実行させる場合はafteranyを指定します。依存関係のあるジョブIDをコロンで区切って指定します。

一連のジョブ実行(ステップジョブ)

--step もしくは --stepany オプションをつけることで、比較的簡単にジョブを順番に実行することができます。

■ ステップジョブ**1:** 複数のジョブを順番に実行する。前のジョブが異常終了した場合には以後のジョブは破棄。

```
ccfep% jsub [-q HR[0-9]] [-gXXX] --step [-W depend=(afterok|afterany):JOBID1[:JOBID2...]] script.csh script2.csh ...
```

■ ステップジョブ**2:** 複数のジョブを順番に実行する。前のジョブが終了したら次のジョブを実行。

```
ccfep% jsub -q (PN|PNR[0-9]) [-gXXX] --stepany [-W depend=(afterok|afterany):JOBID1[:JOBID2...]] script.csh script2.csh ...
```

実行例:

```
ccfep% jsub --stepany job1.csh job2.csh job3.csh
```

ジョブスクリプトで利用可能な変数を定義する

-v オプションを利用して変数を定義することができます。
変数の名前と値は (変数名)=(値) のコンマ区切りの文字列で指定します。

```
ccfep% jsub -v INPUT=myfile.inp,OUTPUT=myout.log script.sh
```

script.sh 中で $INPUT や $OUTPUT がそれぞれ myfile.inp や myout.log に置き換えられます。
-v を複数指定した場合、最後の一つのみが有効になります。ご注意ください。
また、ジョブスクリプトの select= 中の ncpus や jobtype 等の値を置き換えることもできません。

ジョブの状態表示

```
ccfep% jobinfo [-h HOST] [-q QUEUE] [-c|-s|-m|-w] [-n] [-g GROUP|-a] [-n]
```

表示する情報の選択

以下のオプションで表示する情報を選択します。複数同時に指定することはできません。

- -c 最新のジョブの状態を表示します(利用 GPU 数やグループの情報等、一部情報が利用できません)
- -s キューのサマリーを表示
- -m ジョブのメモリ情報を表示
- -w ジョブをサブミットしたディレクトリーを表示
- -n 各計算ノードの状態を表示する

他ユーザーのジョブ表示について

- -g 同一グループ内の全ユーザの情報を表示
  - 複数のグループに属している場合、-g GROUP名 でグループ名を指定することもできます。
- -a 全てのユーザの情報を表示
  - 他ユーザーのジョブ名等の情報は隠蔽されます。

キュー指定関連オプション

何も指定しなければ H 及び専有利用キュー(HR[0-9])のジョブ全てが対象となります。そのため、通常は指定する必要ありません。

- -h HOST: ホスト名で指定します(現在は cclx だけが有効です)
- -q QUEUE: キュー名で指定します

実行例: サマリーの表示

利用中/キュー待ち中/ホールド中ジョブの数やCPU数、GPU数を表示します。制限値についても表示されます。
また、各 jobtype の混雑状況も表示されます。

```
ccfep% jobinfo -s

User/Group Stat:
-------------------------------------------------------------------------------
 queue: H                    | user(***)            | group(***)
-------------------------------------------------------------------------------
   NJob (Run/Queue/Hold/RunLim) |    1/    0/    0/-    |    1/    0/    0/6400
   CPUs (Run/Queue/Hold/RunLim) |    4/    0/    0/-    |    4/    0/    0/6400
   GPUs (Run/Queue/Hold/RunLim) |    0/    0/    0/-    |    0/    0/    0/  48
```

```
  core (Run/Queue/Hold/RunLim) |    4/    0/    0/1200 |    4/    0/    0/1200
-----------------------------------------------------------------------------
note: "core" limit is for per-core assignment jobs (jobtype=core/gpu*)

Queue Status (H):
-----------------------------------------------------------------------
     job        | free   |     free     | # jobs  |  requested
     type       | nodes  | cores (gpus) | waiting | cores (gpus)
-----------------------------------------------------------------------
week jobs
-----------------------------------------------------------------------
1-4 vnodes       |    705 |  90240      |       0 |      0
5+  vnodes       |    505 |  64640      |       0 |      0
largemem         |      0 |      0      |       0 |      0
core             |    179 |  23036      |       0 |      0
gpu              |      0 |      0 (0)  |       0 |      0 (0)
-----------------------------------------------------------------------
long jobs
-----------------------------------------------------------------------
1-4 vnodes       |    325 |  41600      |       0 |      0
5+  vnodes       |    225 |  28800      |       0 |      0
largemem         |      0 |      0      |       0 |      0
core             |     50 |   6400      |       0 |      0
gpu              |      0 |      0 (0)  |       0 |      0 (0)
-----------------------------------------------------------------------
Job Status at 2023-01-29 17:40:12
```

出力上部の core (Run/Queue/Hold/RunLim) はコア単位で利用する場合の制限値です。
上記出力の場合は最大 1200 コアまで利用できます。jobtype=vnode や jobtype=largemem での利用状況には影響を受けません。

> 実行例: 個々のジョブの状態を見る

-c オプションを指定すると、ジョブの最新の状態を確認できます。(以前と同様に -l を指定しても問題ありません)

```
ccfep% jobinfo -c

-------------------------------------------------------------------------------
Queue   Job ID Name           Status CPUs User/Grp       Elaps Node/(Reason)
-------------------------------------------------------------------------------
H       9999900 job0.csh       Run      16  zzz/---    24:06:10 ccc047
H       9999901 job1.csh       Run      16  zzz/---    24:03:50 ccc003
H       9999902 job2.sh        Run       6  zzz/---     0:00:36 ccc091
H       9999903 job3.sh        Run       6  zzz/---     0:00:36 ccc091
H       9999904 job4.sh        Run       6  zzz/---     0:00:36 ccc090
...
H       9999989 job89.sh       Run       1  zzz/---     0:00:11 ccg013
H       9999990 job90.sh       Run       1  zzz/---     0:00:12 ccg010
-------------------------------------------------------------------------------
```

-c を指定しない場合は、数分ほど古い情報になる場合がありますが、GPU 数や jobtype などの情報も確認できます。

```
ccfep% jobinfo
-------------------------------------------------------------------------------
Queue   Job ID Name           Status CPUs User/Grp       Elaps Node/(Reason)
-------------------------------------------------------------------------------
H(c)    9999900 job0.csh       Run      16  zzz/zz9    24:06:10 ccc047
H(c)    9999901 job1.csh       Run      16  zzz/zz9    24:03:50 ccc003
H(c)    9999902 job2.sh        Run       6  zzz/zz9     0:00:36 ccc091
H(c)    9999903 job3.sh        Run       6  zzz/zz9     0:00:36 ccc091
H(c)    9999904 job4.sh        Run       6  zzz/zz9     0:00:36 ccc090
...
H(g)    9999989 job89.sh       Run     1+1  zzz/zz9     0:00:11 ccg013
H(g)    9999990 job90.sh       Run     1+1  zzz/zz9     0:00:12 ccg010
-------------------------------------------------------------------------------
```

例: ジョブの作業ディレクトリを表示する

どこでジョブを実行したのかがわからなくなった場合には -w オプションを追加することで、
ジョブの作業ディレクトリ(PBS_O_WORKDIR)を表示することができます。

```
ccfep% jobinfo -w
--------------------------------------------------------------------------------
Queue   Job ID Name          Status Workdir
--------------------------------------------------------------------------------
H      9999920 H_12345.sh      Run    /home/users/zzz/gaussian/mol23
H      9999921 H_23456.sh      Run    /home/users/zzz/gaussian/mol74
...
```

-c とは併用できません。

ジョブの取消

あらかじめjobinfoコマンドで、取り消したいジョブのRequest IDを調べておきます。その後、

```
ccfep% jdel RequestID
```

とします。

ジョブのホールドとリリース

キュー待ち状態のジョブを実行されないように留めておく(ホールドする)ことができます。
あらかじめjobinfoコマンド等でジョブIDを調べておいた上で以下のコマンドを実行することでジョブをホールドできます。

```
ccfep% jhold RequestID
```

ホールドしたジョブを解放するには、

```
ccfep% jrls RequestID
```

とします。

実行済みジョブの情報取得

ジョブの終了日時、経過時間、CPU 点数等の情報を joblog コマンドで得ることができます。

```
ccfep% joblog [-d 日数] [-oitem1[,item2[,...]]]
```

期間指定をしない場合には今年度実行したジョブの情報を表示します。以下のオプションも利用できます。

- -d ndays: 直近 ndays 以内に終了したジョブについて表示します。
  - 直近 7 日間に終了したジョブの場合: -d 7
- -y year: year 年度のジョブについて表示します。
  - 2021 年度に終了したジョブの場合: -y 2021
- -f YYYY[MM[DD[hh[mm]]]] -t YYYY[MM[DD[hh[mm]]]]: -f で指定した日時から -t で日時で指定した間に終了したジョブについて表示します。(一部省略可能)
  - 2021年の7-8月に実行されたジョブの情報を表示する場合: -f 202107 -t 202108

表示される項目を-oオプションを使ってカスタマイズすることができます。*item*には以下のキーワードを指定することができます。

- queue: キュー名
- jobid: ジョブID
- user: ユーザー名
- group: グループ名
- node: 計算に使われた最初のノード名
- Nodes: 計算に使われた全ノード名
- type: ジョブタイプ
- start: ジョブの開始時刻（YYYY/MM/DD HH:MM）
- Start: ジョブの開始時刻（YYYY/MM/DD HH:MM:SS）

- finish: ジョブの終了時刻（YYYY/MM/DD HH:MM）
- Finish: ジョブの終了時刻（YYYY/MM/DD HH:MM:SS）
- elaps: 経過時間
- cputime: 全CPU時間
- used_memory: 使用したメモリー量
- ncpu: 予約したCPU数
- ngpu: 予約したGPU数
- nproc: MPIのプロセス数
- nsmp: プロセスあたりのスレッド数
- peff: 並列化効率
- attention: 非効率なジョブかどうか
- command: ジョブ名
- exit_status: ジョブの終了コード
- point: ジョブが使用したCPU点数
- standard: jobid,type,finish,elaps,ncpu,ngpu,point (デフォルト)
- all: 全て

■ 例**1:** 最近 **10** 日以内に終わったジョブの **ID,** 開始日時、終了日時、点数を表示

```
ccfep% joblog -d 10 -o jobid,start,finish,point
```

■ 例**2: 2020**年度に実行したジョブの **ID,** 終了日時、点数、実行ディレクトリを表示

```
ccfep% joblog -y 2020 -o jobid,finish,point,Workdir
```

■ 例**3:** 最近 **2** 日以内に終了したジョブの全ての情報を表示

```
ccfep% joblog -d 2 -o all
```

ビルドと実行

ビルドのコマンド

- gcc, aocc, NVIDIA HPC SDK を用意しています。
- インテル oneAPI については、ライブラリ等は導入していますが、共用領域にコンパイラは用意していません。インテルコンパイラが必要であれば oneAPI Base Toolkit や oneAPI HPC Toolkit をご自身のホームディレクトリに導入してください。
- gcc についてはシステム標準のもの(8.5)に加えて gcc-toolset の新しいバージョン(9.2,10.3,11.2)も導入しています。module load gcc-toolset/11 のように実行すれば使えるようになります。

各種ライブラリ、MPI環境等の導入状況についてはパッケージプログラム一覧のページをご覧下さい。

oneAPI 環境の導入と csh で oneAPI 環境を読み込む方法について

Intel oneAPI Base Toolkit はこちらからダウンロードできます。コンパイラ、MKL, MPI などが含まれています。Linux 向け Online か Offline 版をご利用ください。
Fortran のコンパイラが必要な場合は HPC Toolkit も導入する必要があります。こちらよりダウンロードしてください。

bash の場合も以下の方法はそのまま使えますが、素直に ~/intel/oneapi/setvars.sh を読み込んだ方が楽です。
コンパイラや MKL 等の個別パッケージだけを読み込みたい場合は各コンポーネント中の env ディレクトリにあるスクリプトを読み込むこともできます。
(コンパイラーを読み込む場合: source ~/intel/oneapi/compiler/latest/env/vars.sh)

csh での設定読み込みはいくつか方法が考えられますが、ここでは module 化する方法を紹介します。oneAPI は ~/intel 以下に導入済とします。

```
$ cd ~/intel/oneapi
$ sh modulefiles-setup.sh
$ cd modulefiles/
$ module use .
$ module save
```

これを実行すると、oneapi の module を作成し、module の検索パスにそのディレクトリを登録する(module use . の箇所)ことになり

ます。
最後に module save でその設定を保存しています。
RCCS ではログイン時に save した module 環境を restore するようになっているため、次回以降のログイン時には自動的に設定されます。
例えばコンパイラを読み込みたいのであれば、module load compiler/latest を実行することになります。
バージョンの詳細等については module avail コマンド等でご確認ください。

module save の前に compiler 等を読み込んでから save すれば、次回ログイン後はすぐにインテルのコンパイラが使えます。

```
$ cd modulefiles/
$ module use .
$ module load compiler/latest
$ module load mkl/latest
$ module load mpi/latest
$ module save
```

module save した環境を破棄したい場合は module saverm を実行してください。
なお、環境を保存してしまうと、システム側のデフォルト設定の変更に追随できなくなりますので、そこはお気をつけください。

並列プログラムの実行方法

ジョブスクリプト中、select 行で MPI プロセスの数を mpiprocs に、OpenMP のスレッド数を ompthreads に正しく設定して下さい。
例: 合計 12 CPU で 4 MPI プロセス、3 OpenMP スレッドの場合は ncpus=12:mpiprocs=4:ompthreads=3 となります。

センター側で導入したアプリケーションのサンプル(/apl/(アプリ名)/samples 以下に有り)も参考にして下さい。

■ ジョブスクリプトでの **OpenMP** スレッド指定

jsub で実行する場合、ompthreads で指定した値が自動的に指定されます。
スクリプト内で OMP_NUM_THREADS 環境変数で手動指定しても問題ありません。
当然ですが、jsub で実行しない場合(フロントエンドノードでのテスト場合等)は OMP_NUM_THREADS 環境変数を設定する必要があります。

■ **MPI**でのホスト指定

jsub で実行する場合、MPI が使うホストリストのファイル名が PBS_NODEFILE 環境変数に入ります。
センター側で導入している MPI 環境(Intel MPI, OpenMPI)ではこの環境変数を自動的に読み込むため、
いわゆる machine file 指定を省略して実行できます。

例: 4 MPI * 3 OpenMP ハイブリッド並列の例

```
#!/bin/sh
#PBS -l select=1:ncpus=12:mpiprocs=4:ompthreads=3:jobtype=core
#PBS -l walltime=24:00:00
cd $PBS_O_WORKDIR
mpirun -np 4 /some/where/my/program options
```

- OpenMP のスレッド数は ompthreads で指定された値が使われます(OMP_NUM_THREADS=3 を指定したと同義)
- センター側で導入した MPI 環境の場合 machine file は通常省略できます。
- PBS_NODEFILE 環境変数を machine file に指定しても動作は同じです。

Environment Modules

- ログインシェルが csh の場合、ジョブスクリプトでは module コマンドを使う前に source /etc/profile.d/modules.csh を実行してください。
  - ログインシェルが /bin/bash で csh のジョブスクリプト使う場合も source /etc/profile.d/modules.csh が必須です。
  - ログインシェルが /bin/tcsh で sh のジョブスクリプトを使う場合は . /etc/profile.d/modules.sh が必須です。(. は source と同義です)
- スクリプト内で実行する場合は、-s をつけて余計な出力をさせないようにした方が無難です。(例: module -s load openmpi/3.1.6)
- module save コマンドで現在の状況を保存することができます。保存した状況は次にログインした時に自動で読み込まれます。
- Intel oneAPI の setvars.sh を .bashrc 等で読み込んでいる場合、sftp (含 WinSCP 等)での接続ができなくなる場合があります。

    - source ~/intel/oneapi/setvars.sh >& /dev/null のように出力を捨てればひとまず接続はできるはずです。
    - ($PS1 が存在する時だけ setvars.sh を読み込むようにしても大丈夫です。.bash_profile に移動させるだけでも大丈夫かもしれません)
- 詳細については、こちらのページをご覧ください。

パッケージプログラム

- 各システムにインストールされている最新の一覧は、パッケージプログラム状態一覧で参照できます。
  module avail コマンドでも確認できます(キーボードの q を押すか、ページの一番下まで動かせば終了できます)。
- ジョブスクリプトのサンプルは、ccfep:/apl/*アプリケーション名*/samples/ディレクトリーを御覧下さい。
- アプリケーションの実体は、各システムの/apl/*アプリケーション名*/にあります。
- センターでビルドしたアプリケーションの構築法はアプリケーションライブラリーの構築方法を御覧下さい。
- csh 用の設定スクリプトが用意されないソフトが増えてきています。module コマンドならば対応できますので、module コマンドの活用をご検討ください。

ソフトウェア導入の要望

下記の項目を全てご記入の上、rccs-admin[at]ims.ac.jp宛(迷惑メール対策のため、@を[at]に置換しています)に送信してください。
有料ソフトウェアの場合、導入できないことがあります。

- 導入を希望するソフトウェアの名前、バージョン
- ソフトウェアの概要と特長
- 共同利用システムに導入を希望する必要性
- 開発元のURL

その他のRCCS固有コマンド

ジョブ関連

jobinfo, jsub, jdel, jhold, jrls, joblog については上に説明があります。

Gaussian専用ジョブ投入ツール

ジョブは通常上記の jsub コマンドで投入しますが、Gaussian だけは g16sub という専用コマンドが用意されています。
Gaussian 09 用の g09sub というコマンドもあります。使い方については g16sub と基本的に同じです。
Gaussian のインプットを直接与えれば、自動でジョブスクリプトを生成、投入します。

■ g16の場合

```
ccfep% g16sub [-q "QUE_NAME"] [-j "jobtype"] [-g "XXX"] [-walltime "hh:mm:ss"] [-noedit] \
        [-rev "g16xxx"] [-np "ncpus"] [-ngpus "n"] [-mem "size"] [-save] [-mail] input_files
```

- デフォルトでは 8 コアを使用します。
- walltimeのデフォルトは72時間です。ジョブの実行時間より多少多めに時間を設定してください。
- "g16sub"と入力すると各オプションの意味や使用例が表示されます。
- 元となるインプット中の %nproc や %cpu、%mem の情報は g16sub や g09sub に上書きされます。それらについては -np や -mem 等のオプションで指定して下さい
    - -noedit を使えば %mem の上書きを抑制することはできますが、非推奨です。
    - メモリ量については自動的に上限に近い値が指定されます。値を減らしたい時のような特殊ケースを除けば、ユーザ側で指定する必要はありません。
-
- largemem を使う場合は -j largemem の指定が必要です。
- -np 64 や -np 128 を指定した時には自動的に jobtype=vnode になります。

■ 基本形(8コア、72 時間)

```
[user@ccfep somewhere]$ g16sub input.gjf
```

■ コア数、制限時間を変える場合(16 コア、168 時間)

```
[user@ccfep somewhere]$ g16sub -np 16 --walltime 168:00:00 input.gjf
```

formchk の実行についてはこちらの FAQ を参考にしてください。

ジョブの実行開始時間予測(waitest)

ccfep では waitest コマンドで実行開始時間が予測できます。各ジョブが walltime 一杯まで実行されるという前提の元で実行開始時間を予測します。
そのため、細かな誤差を無視すれば waitest の出力は最悪予測になります。
一方でジョブ種類に依存する優先度や振り分けの設定や救済措置等により後続のジョブに順番を追い抜かれる場合もあるので、確実性はあまり高くありません。
jobinfo -s で出力されるキューの空き状態なども確認した上でご利用ください。

基本形

待ち状態にあるジョブの実行開始時間予測:

```
$ waitest [jobid1] ([jobid2] ...)
```

指定したジョブスクリプトを投入した場合の開始時間予測:

```
$ waitest -s [job script1] ([jobscript2] ...)
```

実行例1: キュー待ち中のジョブ ID を指定する場合

(ターミナル上では緑や赤の色付けはありません)

```
[user@ccfep2]$ waitest 4923556
Current Date   : 2023-02-15 14:32:30
2023-02-15 14:32:30 ...
2023-02-15 16:40:44 ...
2023-02-15 22:26:07 ...
2023-02-16 00:43:43 ...
2023-02-16 03:03:11 ...
2023-02-16 05:58:00 ...
2023-02-16 11:34:12 ...
Job 4923556 will run at 2023-02-16 13:03:11 on ccc500.
Estimation completed.
```

実行例2: 実行前のジョブスクリプトを指定する場合

```
[user@ccfep2]$ waitest -s vnode4N1D.sh vnode1N1D.sh

Job Mapping "vnode4N1D.sh" -> jobid=1000000000
Job Mapping "vnode1N1D.sh" -> jobid=1000000001

Current Date   : 2023-09-06 16:43:10
2023-09-06 16:43:10 ...
2023-09-06 18:43:42 ...
2023-09-06 21:19:19 ...
Job 1000000001 will run at 2023-09-06 21:39:34 on ccf013.
2023-09-06 22:02:09 ...
2023-09-07 01:02:14 ...
2023-09-07 03:34:18 ...
Job 1000000000 will run at 2023-09-07 05:28:07 on ccc428,ccc571,ccc356,ccc708.
Estimation completed.
```

(優先度や振り分けの関係で、大規模ジョブが小規模ジョブより入りやすいことがあります。)

実行例3: 一般的なサイズのジョブについての予測情報

一般的なジョブの種類については定期的に予測を行っており、その結果は以下のコマンドで確認できます。

```
[user@ccfep2]$ waitest --showref
```

資源使用状況表示

```
ccfep% showlim (-cpu|-c|-disk|-d) [-m]
```

- -cpu|-c: 使用した使用点数と割当点数の表示
- -disk|-d: 使用しているディスク容量と上限容量の表示

- -m: 所属全メンバー毎の使用状況と上限値の表示

■ 例**1:** 自身及びグループの利用/割り当て **CPU** 点数の表示

```
ccfep% showlim -c
```

■ 例**2:** グループ全体及びグループの各メンバの利用/割り当て **CPU** 点数の表示

```
ccfep% showlim -c -m
```

■ 例**3:** グループ全体及びグループの各メンバの利用/割り当てディスク容量表示

```
ccfep% showlim -d -m
```

ジョブスクリプト用ユーティリティーコマンド

コマンドの実行時間の制限

- RCCSで提供していたps_walltimeコマンドは廃止されました。Linuxで標準的なtimeoutコマンドを使用してください。

ジョブの統計情報の表示

- RCCSで提供していたjobstatisticコマンドは廃止されました。ジョブ終了後にjoblogコマンドを使用してください。

計算ノード上のファイル操作

remshコマンドを使うと、計算ノードのramdiskのようにフロントエンド(ccfep)から直接アクセスできないファイルへアクセスできます。

remsh *hostname command options*

- hostname: ccc???, ccg???, ccnf??? のようなホスト名です。
- command: 実行するコマンド。ls, cat, cp, find, head, tail のいずれかを指定できます。
- options: コマンドのオプションです。

例: ユーザzzzによる計算ノードcccXXXでのramdisk操作

```
remsh cccXXX ls /ramd/users/zzz
```

```
remsh cccXXX tail /ramd/users/zzz/99999/fort.10
```

ジョブを実行しているノード名はjobinfoコマンドより確認できます。

問い合わせ

https://ccportal.ims.ac.jp/contact
をご参照下さい。