

# 鹏城云脑快速使用指南

## 目录

- 鹏城云脑快速使用指南 ..... 1
- 鹏城云脑快速使用指南 ..... 2
- 一、云脑简介 ..... 2
- 二、登录界面 ..... 2
- 三、提交任务 ..... 3
  - 提交 DEBUG 任务详细介绍..... 3
- 四、多机通信环境配置 ..... 8
- 五、上传和下载数据临时解决方案 ..... 11
- 六、附件：Git 快速使用基本命令文档..... 16

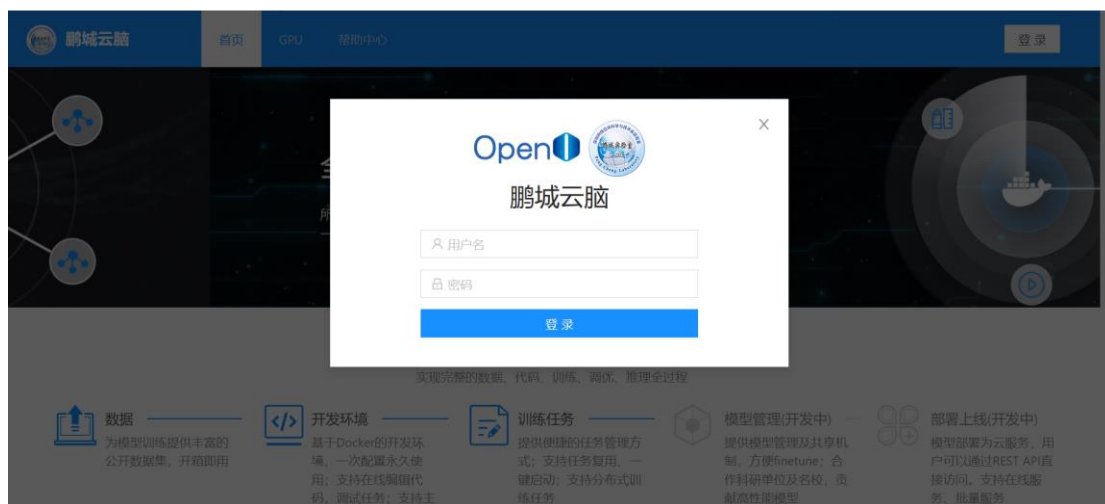
# 鹏城云脑快速使用指南

## 一、云脑简介

云脑平台是基于 OpenI-Octopus 开源项目的集群管理工具和资源调度平台。支持在 GPU 集群中运行 AI 任务作业，平台提供了一系列接口，能够支持主流的深度学习框架，如 pytorch、tensorflow、pandlepandle、mxnet、caffe、cntk 等。这些接口同时具有强大的可扩展性：添加一些额外的脚本或者 Python 代码后，平台即可支持新的深度学习框架（或者其他类型的工作）。用户可以方便的通过界面向云脑平台提交 GPU、CPU 等任务，云脑平台将在后台根据用户的资源需求对任务进行调度，选择合适的服务器节点并将任务以 docker 的形式启动。用户可方便的在平台上 debug 代码，运行任务，上传、下载数据集及模型。

## 二、登录界面

网址：<http://cloudbrain.pcl.ac.cn/#/openi/v2/home>



随后前往控制台，可看到多个模块，其中：

提交任务：用户可在此界面按需提交任务，以下会详细说明；

任务列表：已提交任务列表，用户可从任务列表中点击再次提交；

镜像列表：其他用户配置好环境后保存的镜像，供用户参考。

# 三、提交任务

类型：有三种选择，分别为 DEBUG\_CPU、DEBUG 和 DGX。其中：

**DEBUG\_CPU**：只含 CPU 模式，不限使用时长；

**DEBUG**：最常用模式，单次限时 2 小时，2 小时后自动关闭，如有需要请在关闭前保存镜像，用户可使用 Jupyterlab 代码编辑器调试程序；

**DGX**：在 DEBUG 模式下调试好程序后，即可切换至 DGX 模式运行程序，该模式下不限时长。

## 提交 DEBUG 任务详细介绍

### 1. 基本信息

任务名可任意取，类型选择 debug，重复次数默认为 0，镜像可以在镜像列表选取也可以使用 dockerhub，阿里云等公共镜像资源。

基本信息

任务名

example

326947

重复次数

0

类型

DEBUG

镜像

192.168.202.74:5000/user-images/deepo:v2.0

子任务列表

子任务名

CPU数

GPU数

内存(MB)

共享内存(MB)

副本数

最小副本成功数

最小副本失败数

InfiniBand设备

启动命令


操作

快速通道


### 2. 添加（子任务）

子任务名也可任意取，但建议跟任务环境相关，debug 模式下，其他选项均为默认值，不可更改。


## 编辑子任务

子任务名 


test

副本数 

1

最小副本成功数 


1

最小副本失败数 


1

CPU数 


2

GPU数 

1

内存(MB) 

16384

共享内存(MB) 

8192

启动命令 


```
pip3 install jupyterlab==1.1.4;service ssh stop;jupyter lab --no-browser --ip=0.0.0.0 --allow-root --notebook-dir="/userhome" --port=80 --NotebookApp.token="" --LabApp.allow_origin="self https://cloudbrain.pcl.ac.cn"
```

☐ InfiniBand设备 

### 3. 点击确定后，出现本次任务基本配置信息


可以添加多个子任务，子任务列表中详细描述了各子任务具体配置，如 test 中：  
CPU 数量为 2，GPU 数量为 1。一般添加单个子任务即可。

基本信息


任务名 

example


328947

重试次数 

0

类型 

DEBUG

镜像 

192.168.202.74:5000/user-images/deeppcv2.0

子任务列表

子任务名	CPU数	GPU数	内存(MB)	共享内存(MB)	副本数	最小副本成功数	最小副本失败数	InfiniBand设备	启动命令	操作
test	2	1	16384	8192	1	1	1	false	pip3 install jupyterlab==1.1.4;service ssh stop;jupyter lab --no-browser --ip=0.0.0.0 --allow-root --notebook-dir="/userhome" --port=80 --NotebookApp.token="" --LabApp.allow_origin="self https://cloudbrain.pcl.ac.cn"	<div>编辑</div> <div>删除</div> <div>提交</div>
anothertest	2	1	16384	8192	1	1	1	true	pip3 install jupyterlab==1.1.4;service ssh stop;jupyter lab --no-browser --ip=0.0.0.0 --allow-root --notebook-dir="/userhome" --port=80 --NotebookApp.token="" --LabApp.allow_origin="self https://cloudbrain.pcl.ac.cn"	<div>编辑</div> <div>删除</div> <div>提交</div>

+ 添加

快速浏览

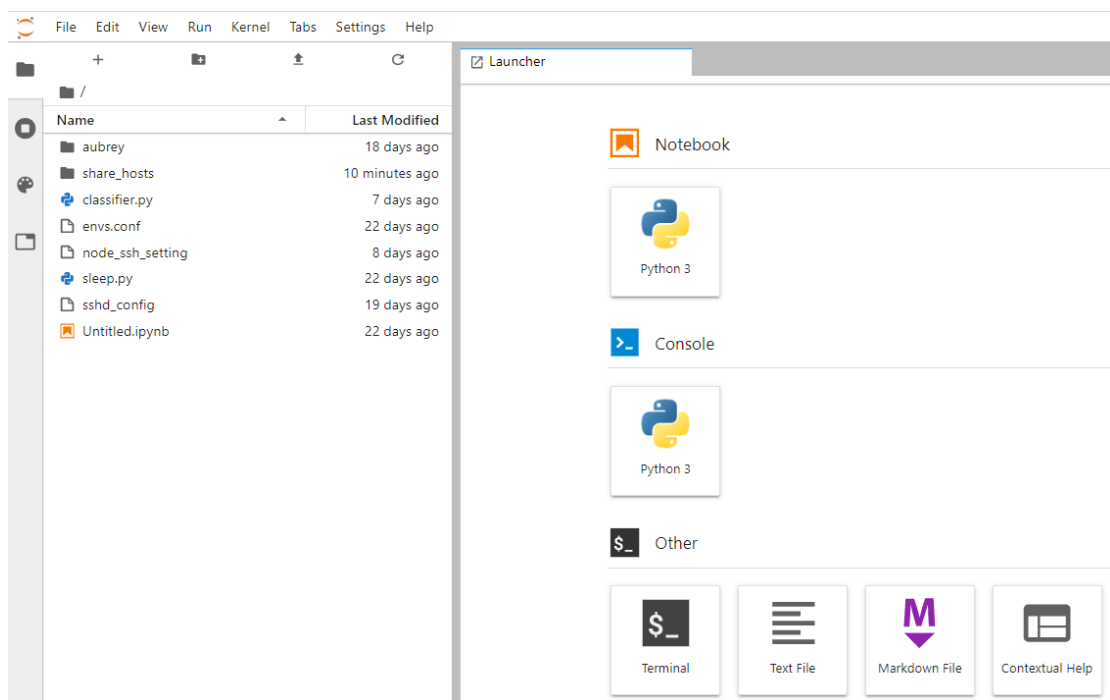
### 4. 点击提交后，可在任务列表中查看当前任务运行状况

Debug 模式下运行状态有：waiting、running、stopped 等

任务名	用户	状态	开始	时长	重试	停止	再次提交
example326947	liuqy01	Waiting	2020-07-10 09:53:11	1m 6s	0	停止	再次提交
aubrey387312	liuqy01	Stopped	2020-07-09 17:53:16	1m 36s	0	停止	再次提交
aubrey862290	liuqy01	Stopped	2020-07-03 10:31:29	13m 27s	0	停止	再次提交
aubrey678536	liuqy01	Stopped	2020-07-02 10:01:50	39m 49s	0	停止	再次提交

## 5. 进入调试界面

点击任务名，如上图中的 **example326947**，即可查看具体任务配置和运行简况等信息，等任务状态显示为 running 时，点击调试可进入调试界面（建议使用 Google Chrome 浏览器，其他浏览器不一定支持）：



## 6. 点击 Terminal，输入 bash，即可开始终端界面操作

```

root@20de6a700c254011ea0a9dc099f632e03dbe-individual-0-0-0: X
# bash
root@20de6a700c254011ea0a9dc099f632e03dbe-individual-0-0-0:/# ls
bin boot dev etc gdata home lib lib64 media mnt model-hub opt proc root run/sbin srv sys userhome usr var
root@20de6a700c254011ea0a9dc099f632e03dbe-individual-0-0-0:/# pwd
/
root@20de6a700c254011ea0a9dc099f632e03dbe-individual-0-0-0:/#

```

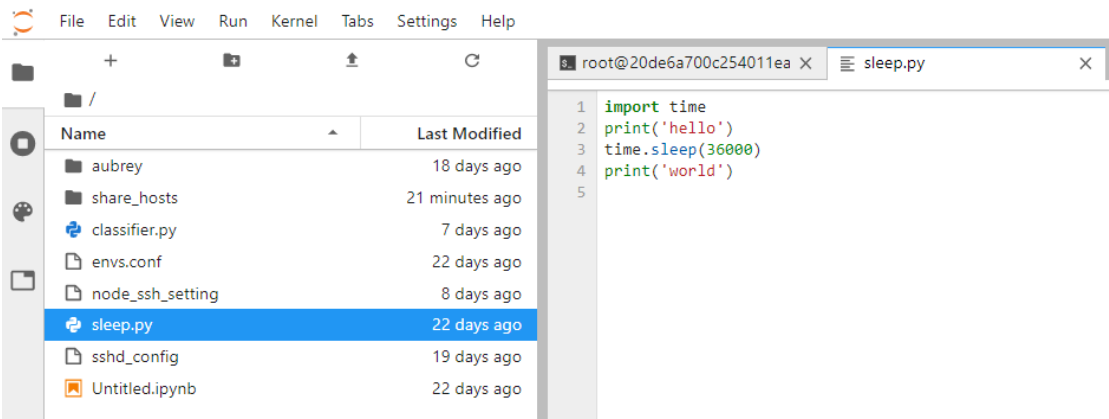
## 7. 保存镜像

调试完毕后，在任务列表界面，点击镜像提交可保存镜像。下次提交任务时选取保存的镜像，就可以直接使用配置好的环境。在任务列表界面，可以停止或者再次提交任务。



### 注意事项:

在 JupyterLab 界面 vim 文本编辑器不太好用，可以将需要编辑的文本移动至 /userhome 路径，然后双击该文本即可进行编辑，编辑完成后再拷贝至原路径。



### DEBUG\_CPU 任务说明

DEBUG\_CPU 任务只含 CPU 模式，不限使用时长，点击任务终端后，先按快捷键 r，然后输入 bash 即可进入终端操作界面。

任务配置 运行简况 **任务终端**

任务名	类型	状态	开始	时长	重试	停止	再次提交
aubrey111338	debug_cpu	Running	2020/07/10 16:31:28	6m 18s	0	停止	再次提交

子任务名	序号	容器名	状态	操作
individual	0	bc724fa5131b4e076ef775914af193c8173e000f3dba610fd11d260db7ae8ca	Running	指标 详情

## DGX 任务说明

DGX 任务与 DEBUG 任务最大的区别在于提交 DGX 任务时，各参数选项可自由选择，并且 DGX 任务不限时（DEBUG 任务单次只能运行两小时），但 DGX 任务没有提供终端调试界面。

The screenshot displays the DGX task configuration interface. On the left, the 'Basic Information' section includes a task name field (aubrey), a type dropdown (DGX), and a sub-task list table. The sub-task list table has columns for sub-task name, CPU count, GPU count, memory, shared memory, replica count, minimum successful replica count, minimum failed replica count, InfiniBand settings, and start command. The right panel, titled 'Edit Sub-task', allows for configuring specific sub-tasks like 'worker', setting the replica count to 2, and defining resource requirements such as 2 CPUs, 2 GPUs, 16384 MB memory, and 64 MB shared memory. It also includes a field for the start command and an option for InfiniBand settings.

其中副本数  $n$  表示该子任务创建的容器数量，默认为 1，可以理解为以子任务中的配置（CPU 数为 2，GPU 数为 2，内存 16384MB 等）启动  $n$  个相同配置的容器。

启动命令：用户在容器启动后需要执行的一系列指令，如切换至指定路径，运行环境配置脚本，以及执行程序命令等。

## 四、多机通信环境配置

目标：抓取启动的多个容器的 IP 地址，并配置好多节点之间的 ssh 无密码访问

### 1. 启动 ssh 服务

首先需要在 DEBUG 模式下确保镜像已经安装 ssh 服务端和客户端，ubuntu 系统

具体操作可参考：<https://www.cnblogs.com/asyang1/p/9467646.html>

`ps -e | grep ssh` 命令可查看当前镜像是否已经安装和启动 ssh 服务；

`apt-get install openssh-client` 命令可安装 ssh 客户端；

`apt-get install openssh-server` 命令可安装 ssh 服务端；

`/etc/init.d/ssh start` 命令可启动 ssh 服务。

### 2. 生成密钥

可参考：<https://www.jianshu.com/p/057afbda5741>

`ssh-keygen` 命令生成密钥，在用户根目录下的 .ssh 文件中生成私钥 `id_rsa` 和公钥 `id_rsa.pub`；

### 3. 备份密钥

将生成的 `id_rsa` 和 `id_rsa.pub` 拷贝至云脑平台共享存储目录下，如：

```
cd /userhome; mkdir -p aubrey/ssh_key;
```

```
cp ~/.ssh/id_rsa* /userhome/aubrey/ssh_key
```

### 4. 云脑平台脚本实现

云脑平台容器启动后，会在 `/etc/hosts` 文件中生成对应的虚拟 IP 地址（IP 地址与子任务名间有映射关系），可在此文件中抓取 IP 地址，同时也可在容器启动后将上述生成的密钥分发至各节点根目录下的 .ssh 路径并修改相应权限，具体脚本实



现如下（脚本命名为 node\_ssh\_setting）：

#将秘钥分发至各节点根目录下.ssh 路径，并修改相应权限；

```
cp /userhome/aubrey/ssh_key/id_rsa.pub /root/.ssh/
```

```
cp /userhome/aubrey/ssh_key/id_rsa /root/.ssh
```

```
cd /root/.ssh
```

```
cat id_rsa.pub >> authorized_keys
```

```
chmod 600 authorized_keys
```

#从/etc/hosts 中抓取 IP 地址，生成节点列表，worker 为提交 DGX 任务时标注的

#子任务名；

```
cat /etc/hosts |grep work| awk '{print $1}' > /userhome/aubrey/node/`date
```

```
+"%Y%m%d"`_nodelist 2>&1
```

```
cd /userhome/aubrey/node/
```

```
sed -i 's/$/& slots=4/g' `date +"%Y%m%d"`_nodelist
```

最后两行代码，是为 mpi 通信进行节点环境配置而加的，用户可注释掉或者

据自己需求修改，目的是在每个节点上分配 4 个进程（slots=4）

例如，我提交 DGX 任务时选的副本数为 10，启动命令中添加执行此脚本命令

（/etc/init.d/ssh start; cd /userhome; ./node\_ssh\_setting; python sleep.py），  
则生成的节点列表如下：

```
root@5ba298d00c27b011ea08cb10936f6b89b786-individual-0-0-0:/userhome/aubrey/node# ls
20200702_nodelist
root@5ba298d00c27b011ea08cb10936f6b89b786-individual-0-0-0:/userhome/aubrey/node# cat 20200702_nodelist
10.13.56.94 slots=4
10.155.117.19 slots=4
10.104.56.107 slots=4
10.253.213.246 slots=4
10.110.211.95 slots=4
10.246.110.31 slots=4
10.146.60.9 slots=4
10.26.236.26 slots=4
10.45.206.40 slots=4
10.160.4.40 slots=4
root@5ba298d00c27b011ea08cb10936f6b89b786-individual-0-0-0:/userhome/aubrey/node#
```

5. 多节点执行任务

实现抓取 IP 地址并配置节点间 ssh 无密码访问后，用户可通过以下方式在这 10 个节点上进行操作：

（1）通过启动命令执行

直接在启动命令中通过脚本或命令行的方式运行任务，但需要注意在运行多节点通信任务前设置等待，比如等待 2 分钟，因为各容器启动时间不确定，如果在容器全部启动完成前就执行多节点通信任务，任务会失败；如下图所示，启动 10 个副本，部分容器启动成功 running 状态，部分容器启动较慢 waiting 状态。

任务名	类型	状态	开始	时长	重试	停止	再次提交
aubrey718839	dgx	Running	2020/07/10 15:47:33	44s	0	停止	再次提交

子任务名	序号	容器名	状态	操作
worker	0		Waiting	指标 详情
worker	1		Waiting	指标 详情
worker	2	b5a28c6951875b0e9541583fd531f3fd6d47e30a4f43c9b85ff75f7bd18129f	Running	指标 详情
worker	3	392e7e540d36393143242724a95971693e949114c6c16338b7a286a58cfb0d57	Running	指标 详情
worker	4		Waiting	指标 详情
worker	5		Waiting	指标 详情
worker	6		Waiting	指标 详情
worker	7	208667737def1411eb4ac4221ad04c4fadb0c1f503d7dd21bc1db04c22be4a	Running	指标 详情
worker	8	59ae671ea635543f0e0518d60bce6221aa0be6c0b59428691d4d43d773b831	Running	指标 详情
worker	9	656c55b0296d1c6a2ca980bd19d9e6a2f3ddb895d9f814f03702caa8a424aa05	Running	指标 详情

## （2）后台终端执行

可从 DEBUG 终端界面，通过虚拟 IP 地址进入其他容器，然后再去执行分布式训练多机 mpi 通信等任务。在上述过程中，我们已经实现通过脚本配置启动的 10 个节点间的 ssh 无密码访问，并且抓取了这 10 个节点的 IP 地址列表（可以在 DGX 模式下先让这 10 个节点运行 sleep 任务，目的是为了占用这 10 个节点），现在只需将备份在共享存储下的秘钥拷贝至 DEBUG 任务终端相应路径中，具体操作执行 node\_ssh\_setting 中的相关命令。这相当于将新启动的 DEBUG 节点也添加至 ssh 无密码访问环境中，即 11 个节点间可以 ssh 无密码访问，然后就可以进入运行 sleep 程序的节点，并在节点上执行多机通信任务。

通过运行以下命令可实现 DEBUG 终端与前述 10 节点间的 ssh 无密码访问：

```
cp /userhome/aubrey/ssh_key/id_rsa.pub /root/.ssh/  
cp /userhome/aubrey/ssh_key/id_rsa /root/.ssh  
cd /root/.ssh  
cat id_rsa.pub >> authorized_keys  
chmod 600 authorized_keys
```

## 五、上传和下载数据临时解决方案

上传和下载数据临时解决方案，操作流程是将数据从本地的 git 仓库（win 系统中要上传的数据）传到自己注册在鹏城实验室的（<https://git.pcl.ac.cn/>）git 远程仓库，然后在终端（鹏城云脑提交任务启动的容器）通过 git clone/pull 将数据下载下来，详细操作流程如下所示。

## 1. 登录鹏城实验室的 git

网址 <https://git.pcl.ac.cn/>，没有注册的用户请先注册一个用户，创建一个远程仓库，用来上传数据。以上传在 windows 系统下载的 mkl2019 为例：

### 创建仓库

所有者 \*

 liuqy

仓库名称 \*

mkl

好的存储库名称使用简短、深刻和独特的关键字。

可见性

☐ 将仓库设为私有

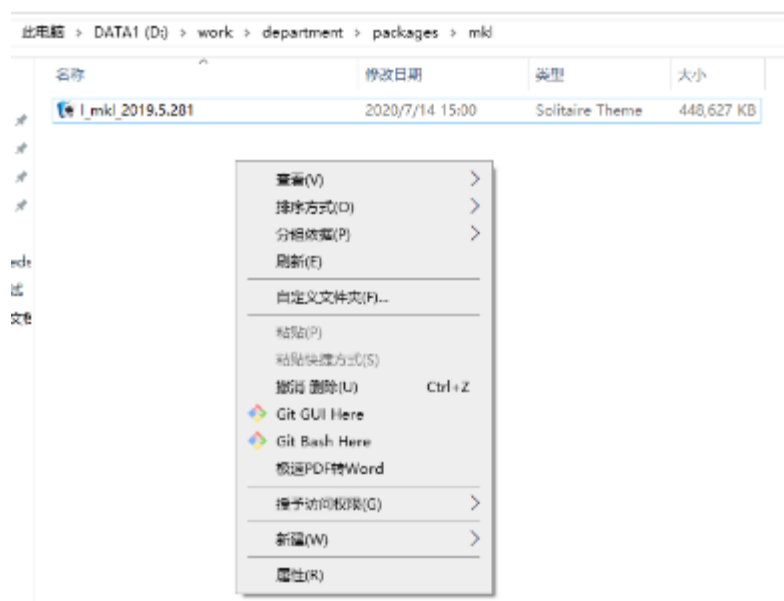
只有课题所有人或拥有权利的课题成员才能看到。

仓库描述

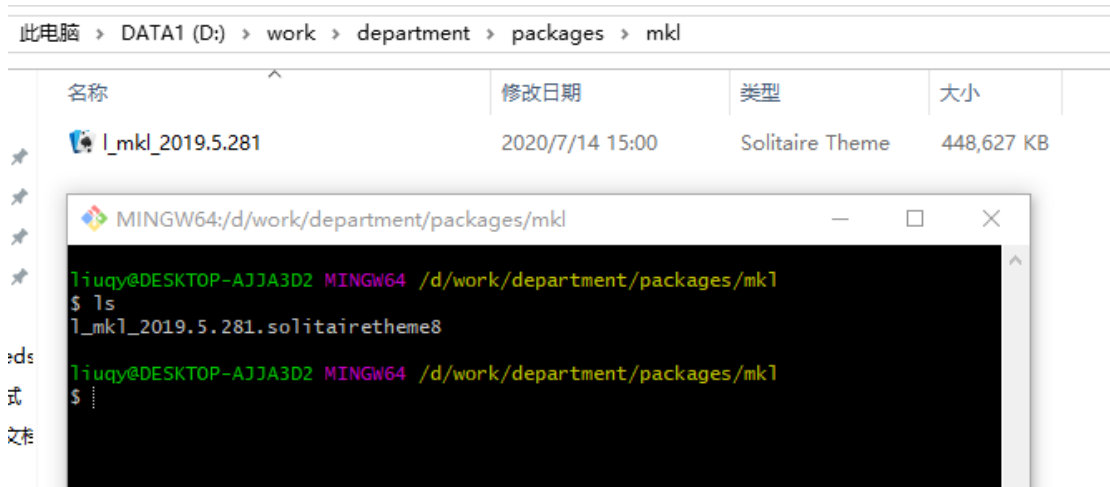
mkl2019

## 2. 配置本地仓库

(1) 在 windows 系统下载和安装 git (<https://git-scm.com/downloads>)，然后移动到你想要上传数据目录，单击鼠标右键：



选择 Git Bash Here:



(2) 然后设置用户名和邮箱(--global 为全局参数表明本地所有 Git 仓库都会使用这个配置)

```
git config --global user.name "yourname"
```

```
git config --global user.email your\_email@youremail.com
```

可用 git config --list 命令查看是否配置成功:

```
liuqy@DESKTOP-AJJA3D2 MINGW64 /d/work/department/packages/mkl
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager
user.name=liuqy
user.email=liuqy01@pcl.ac.cn
liuqy@DESKTOP-AJJA3D2 MINGW64 /d/work/department/packages/mkl
```

(3) 配置 ssh 秘钥，将生成的秘钥添加到远程仓库（鹏城 git）：

ssh-keygen; （会在 ~/.ssh 路径下生 id\_rsa id\_rsa.pub）

cat id\_rsa.pub

```
liuqy@DESKTOP-AJJA3D2 MINGW64 ~/.ssh
$ cat id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQgQDeu0wNdT0cLvMkzS/j63fp/LGpbcEGD6BqwW4+L4G1
U1Z0HVjEXkETRYbhhi2VmHkuD8JDILmuZKPs2isX/q1fzUaa704Zz6tgNA0DYC46MGTMegLmgWidamMR
1L9/nL1nkyv8egATa25keLg3PZxZgTbDnVkvKJWNJTf0R24gWjvG81Zdgn5kPiFrGZ0oEASVsThJJhF1
0ZKzdciiw+cvyEGtEcNfjEcCR7vmtXHEb0ovwJ1cPLYO/C/rj+iw6LcstaWBWjR1P+rfdAAL+9Bjm5X
nGTZfTcWk0/9no8Va0oSQXcl0eQ8mKP+7yjlj/qmUfmxuPUGeoMfint1Gk1Ls21QXrPSFedQsZe25
01B8yZUbuPYi9sVxA+pT4g+kgDWR7NKyadyGotm1Ne+Eh58eEicjJovyOR+aFMZ+4ahhe3ByJhf/01S+
A45Pg5yQrZ/c8yUhqCZQz0AiXJ0L8NZcLizGzz+090Avat7PznyZz3DY/Mn0FDWZsLL5Y58= liuqy01
@pcl.ac.cn

liuqy@DESKTOP-AJJA3D2 MINGW64 ~/.ssh
$
```

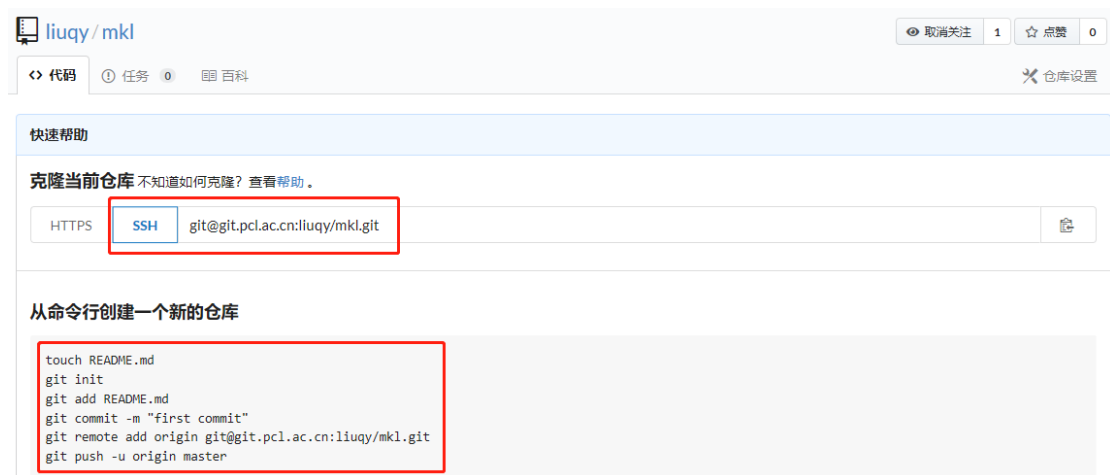
然后将 id\_rsa.pub 添加至远程仓库（鹏城 git）



(4) 按如上步骤配置好基础设置后，回到鹏城 git 新建的 mkl 仓库，选 ssh 通信

（http 会限制上传数据大小），然后按如下命令行提示在本地（win 路径）Git Bash

Here 界面依次执行：

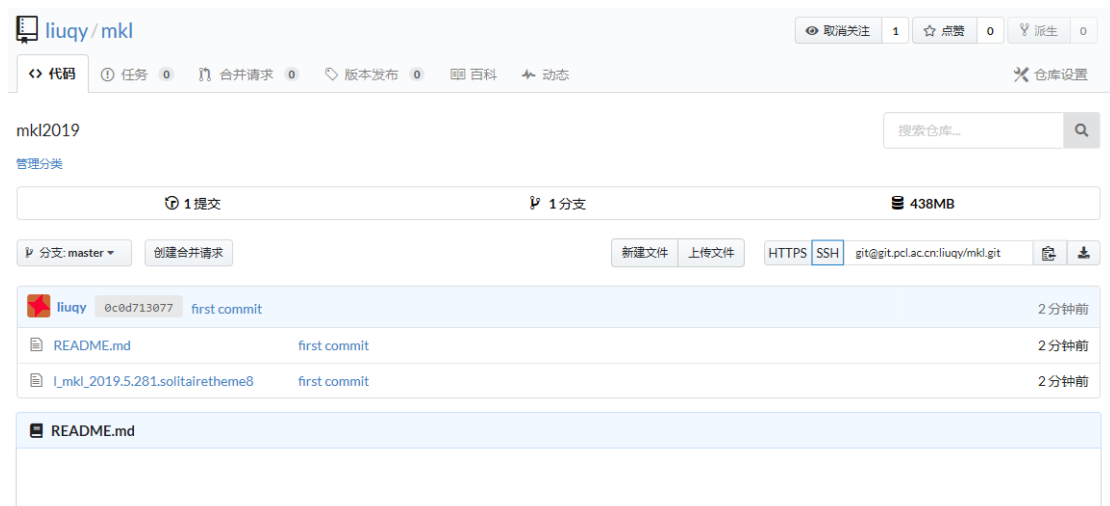


将 `git add README.md` 替换成 `git add *`即可。

```
liuqy@DESKTOP-AJJA3D2 MINGW64 /d/work/department/packages/mkl (master)
$ git push -u origin master
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 6 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 438.17 MiB | 9.11 MiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
remote: . Processing 1 references
remote: Processed 1 references in total
To git.pcl.ac.cn:liuqy/mkl.git
* [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.

liuqy@DESKTOP-AJJA3D2 MINGW64 /d/work/department/packages/mkl (master)
$
```

执行完命令后，就将本地仓库（win 端数据）上传到了远程仓库（鹏城 git）：



### 3. 鹏城云脑与本地仓库（win 端）间数据传输

至此，鹏城云脑和本地仓库（win 端）间就可以通过远程仓库（鹏城 git）进行数据交互。

下载数据：

`git clone git@git.pcl.ac.cn:liuqy/mkl.git`

其他常用 git 指令：

`git pull` （更新本地仓库，使之与远程仓库一致）

另附 git 快速使用指南如下：

## 六、附件：Git 快速使用基本命令文档

刘琼瑶 (2019.06.19)

### 一 . git clone 项目详细流程

#### 1. 下载已有项目

`git clone https://github.com/hpc-app/relion-3.0_beta`

```
[liuqy@admin419 sourcecode]$ git clone https://github.com/hpc-app/relion-3.0_beta
Cloning into 'relion-3.0_beta'...
remote: Enumerating objects: 1338, done.
remote: Counting objects: 100% (1338/1338), done.
remote: Compressing objects: 100% (814/814), done.
remote: Total 1338 (delta 596), reused 1258 (delta 516), pack-reused 0
Receiving objects: 100% (1338/1338), 2.60 MiB | 70.00 KiB/s, done.
Resolving deltas: 100% (596/596), done.
[liuqy@admin419 sourcecode]$
```

#### 2. 查看状态(刚下载项目，查看时会显示如下)

`git status`

```
[liuqy@admin419 relion-3.0_beta]$ git status
# On branch master
nothing to commit, working directory clean
[liuqy@admin419 relion-3.0_beta]$
```

#### 3. 查看配置信息

`git config --list`

```
[liuqy@admin419 relion-3.0_beta]$ git config --list
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
remote.origin.url=https://github.com/hpc-app/relion-3.0_beta
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
branch.master.remote=origin
branch.master.merge=refs/heads/master
[liuqy@admin419 relion-3.0_beta]$
```

#### 4. 设置用户信息

`git config --global user.name "liuqy"`

`git config --global user.email liuqy@sugon.com`

`git config --list`



```
[liuqy@admin419 relion-3.0_beta]$ git config --global user.name "liuqy"
[liuqy@admin419 relion-3.0_beta]$ git config --global user.email liuqy@sugon.com
[liuqy@admin419 relion-3.0_beta]$ git config --list
user.name=liuqy
user.email=liuqy@sugon.com
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
remote.origin.url=https://github.com/hpc-app/relion-3.0_beta
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
branch.master.remote=origin
branch.master.merge=refs/heads/master
[liuqy@admin419 relion-3.0_beta]$
```

##### 5. 修改文件后再查看状态，例如，我新编辑一个 example\_for\_beginner 文件

```
[liuqy@admin419 relion-3.0_beta]$ git status
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       example_for_beginner
nothing added to commit but untracked files present (use "git add" to track)
[liuqy@admin419 relion-3.0_beta]$
```

如图所示，与原项目相比，有修改的地方只有新建的 example\_for\_beginner 文件，用 git status 查看状态时会显示为 untracked file。

##### 6. 将新修改文件添加至暂存区

`git add example_for_beginner`

`git status` （添加成功后状态显示如下）

```
[liuqy@admin419 relion-3.0_beta]$ git add example_for_beginner
[liuqy@admin419 relion-3.0_beta]$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   example_for_beginner
[liuqy@admin419 relion-3.0_beta]$
```

##### 7. 提交更新

`git commit -m "你可以根据此次修改来命名"`

例：

```
$ git commit -m "Story 182: Fix benchmarks for speed"
[master 463dc4f] Story 182: Fix benchmarks for speed
2 files changed, 2 insertions(+)
create mode 100644 README
```

如图所示，提交更新后会显示如下信息，当前你是在哪个分支（master）提交的，在本次提交中有多少文件被修改，有多少行添加和删除过等。

#### 8. 推送至远程仓库

`git push -u origin master`

至此，本次修改便成功上传到了远程仓库。

补充：

（1）当 `git add` 文件后发现文件错误，可以使用 `git rm` 来删除提交至暂存区的文件；如：

`git rm -rf example_for_beginner`

`git status` (移除后查看状态显示如下，因为没有其他修改，项目和初始下载时一致)

```
[liuqy@admin419 relion-3.0_beta]$ git rm -rf example_for_beginner
rm 'example_for_beginner'
[liuqy@admin419 relion-3.0_beta]$ git status
# On branch master
nothing to commit, working directory clean
[liuqy@admin419 relion-3.0_beta]$ ls
AUTHORS  betaGuide.pdf  cmake  CMakeLists.txt  data  LICENSE  README.md  scripts  src  tests
[liuqy@admin419 relion-3.0_beta]$
```

#### （2）忽略文件

`cat .gitignore`

```
[liuqy@admin419 relion-3.0_beta]$ cat .gitignore
build
external
#Eclipse files
.cproject
.project
Debug
[liuqy@admin419 relion-3.0_beta]$
```

凡是在.gitignore 文件中的文件或目录都不会被追踪，例如图中，build/目录下所有文件的修改都不会被追踪。

### (3) 查看提交历史

git log

```
[liuqy@admin419 relion-3.0_beta]$ git log
commit 2cb94cd5dc53c2168fab292960c09f79738123fa
Author: liuqy <liuqy@sugon.com>
Date:   Fri Apr 26 11:26:56 2019 +0800

    fix no matching function for windowFourierTransform2 and std::tuple

commit e3e2c1d8ed7c286a3e8df82cb083332174e787b7
Author: liuqy <liuqy@sugon.com>
Date:   Wed Mar 27 16:48:07 2019 +0800

    fix ld: HIPFFT

commit dfc4a1008db927e232234d333d888244cbaa6b81
Author: liuqy <liuqy@sugon.com>
Date:   Wed Mar 27 15:35:19 2019 +0800
```

### (4) 更新本地仓库，使之与远程仓库保持一致

git pull

## 二. 最简操作流程

### 1. 克隆项目

- (1) git clone [https://github.com/hpc-app/relion-3.0\\_beta](https://github.com/hpc-app/relion-3.0_beta) (克隆项目)
- (2) git config --global user.name "liuqy" (设置用户信息)  
git config --global user.email [liuqy@sugon.com](mailto:liuqy@sugon.com)
- (3) git status (查看文件状态)
- (4) git add file\_name (追踪文件, 提交至暂存区)
- (5) git commit -m "方便标记本次修改的备注" (提交更新)
- (6) git push -u origin master (推送至远程仓库)

注: 拉取指定分支及所有子仓库

git clone -b "branch\_name" --recursive

[ssh://git@10.0.50.206:10022/419/relion-3.0\\_beta.git](ssh://git@10.0.50.206:10022/419/relion-3.0_beta.git)

## 2. 自己创建项目

- (1) git init (例如在 relion-3.0\_beta 目录下)
- (2) git remote add origin [ssh://git@10.0.50.206:10022/419/relion-3.0\\_beta.git](ssh://git@10.0.50.206:10022/419/relion-3.0_beta.git)  
(创建远程仓库)
- (3) git add . (追踪所有文件)
- (4) git commit -m "Initial commit" (提交更新)
- (5) git push -u origin master (推送至远程仓库)

注: git pull (更新本地仓库, 使之与远程仓库保持一致)

## 3. 下载和切换至特定的历史版本

- (1) git clone [https://github.com/hpc-app/relion-3.0\\_beta.git](https://github.com/hpc-app/relion-3.0_beta.git)
- (2) cd relion-3.0\_beta/
- (3) git log
- (4) git checkout b5db330999cf7463db81c65fe15f9ea23548768e