

Руководство пользователя

 computing.kiae.ru/user-support/user-manual.html

Оглавление

Политика безопасности ресурсов MBK

Пользуясь нашими вычислительными ресурсами вы автоматически соглашаетесь с нашей [политикой безопасности](#), поэтому вам нужно внимательно прочитать этот документ и следовать его предписаниям.

Как сообщить о проблеме

В случае любых проблем, относящихся к вашей работе на данном кластере, пожалуйста, пишите письмо с

- подробным описанием проблемы,
- вашим именем,
- названием машины, которое вы использовали для входа на кластер (например, ui2.computing.kiae.ru),

по адресу «help@computing.kiae.ru».

Доступ к ресурсам

В настоящее время зарегистрированным пользователям доступны следующие кластеры:

- [HPC2](#), доступ с машины ui2.computing.kiae.ru.
- HPC4, доступ с машины ui4.computing.kiae.ru.

Заход на указанные выше головные машины кластеров осуществляется с использованием протокола SSH. Для соединения можно использовать SSH-клиентов, поддерживающих протокол SSH версии 2. Подойдут, например,

- [Putty](#) (Windows),
- [SSH Secure Shell Client](#) (Windows, также есть версия для UN*X, но OpenSSH лучше),
- [OpenSSH](#) (UN*X, входит в комплект практически всех современных UN*X-подобных операционных систем).

Для копирования файлов можно использовать следующие утилиты:

Создание SSH-ключей

Для аутентификации на нашем кластере в настоящий момент используются SSH-ключи. Если вы не знаете, что это такое, то сейчас вам это будет, по мере возможности, объяснено.

Если коротко, то SSH-ключи — это пара файлов, один из которых называется закрытым ключом, обычно защищен паролем и не показывается никому, во избежание разных неприятностей. Второй файл называется открытым ключом и может быть показан кому угодно; более того, его нужно поместить на сервер, чтобы он начал вас пускать. Из открытого ключа, в-принципе, можно восстановить закрытый, но из-за сложности разложения чисел на простые множители на современных машинах это займет достаточно большое количество времени.

Так вот, вся штука заключается в том, что имея только открытый ключ сервер может проверить, есть ли у вас соответствующий ему закрытый ключ. А поскольку предполагается, что закрытый ключ есть только у вас (именно поэтому его так важно никому не показывать), то сервер сможет утверждать, что если проверка прошла успешно, то к нему стучитесь именно вы.

Теперь один из важных практических вопросов: как создать эти самые SSH-ключи (на сервер их положат наши администраторы, поэтому здесь вы можете быть спокойны).

Создание SSH-ключей на Unix-системах

Предполагается, что у вас установлен пакет OpenSSH, а стало быть, есть утилита `ssh-keygen`. Запускаем:

```
$ ssh-keygen -t rsa -b 2048
Generating public/private rsa key pair.
Enter file in which to save the key (/home/vasya/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your public key has been saved in /home/vasya/.ssh/id_rsa.pub.
The key fingerprint is:
22:f7:7b:96:c5:4b:b2:0d:2d:9f:5f:67:57:48:46:37 vasya@some_host_in_the_net
```

Соответственно, если вы хотите сохранить ваш ключ не в `/home/vasya/.ssh/id_rsa`, а куда-то еще, вы можете ввести полный путь в ответ на приглашение «Enter file in which to save the key». Но если вы пока не очень в курсе, как использовать различные SSH-ключи для различных машин, то лучше оставить все как есть.

После отработки `ssh-keygen` закрытый ключ будет находиться в файле `/home/vasya/.ssh/id_rsa`, а открытый — `/home/vasya/.ssh/id_rsa.pub`. Именно последний файл вам и нужно отослать нам. Пожалуйста, не перепутайте ;))

Если вы используете нестандартное имя файла с закрытым ключом, то вам либо придется указывать его расположение каждый раз с помощью ключа «-i», либо добавить в файл `~/.ssh/config` следующие строки:

```
Host ui2.computing.kiae.ru
    IdentityFile ~/.ssh/id_rsa-hpc2
```

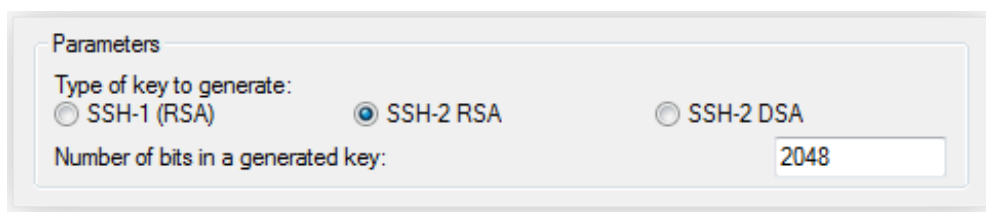
Если вы используете один и тот же открытый ключ для нескольких наших кластеров, то можно использовать настройки, в которых имя узла будет содержать символ «*», который соответствует любому количеству любых символов:

```
Host ui*.computing.kiae.ru
    IdentityFile ~/.ssh/id_rsa-kiae
```

Создание SSH-ключей на Windows-системах

Putty

Для создания SSH-ключа вам нужно скачать программу PuTTYgen с [сайта Putty](#). Запускаем программу и выбираем тип ключа «SSH-2 RSA», а его размер — 2048 бит:



Parameters

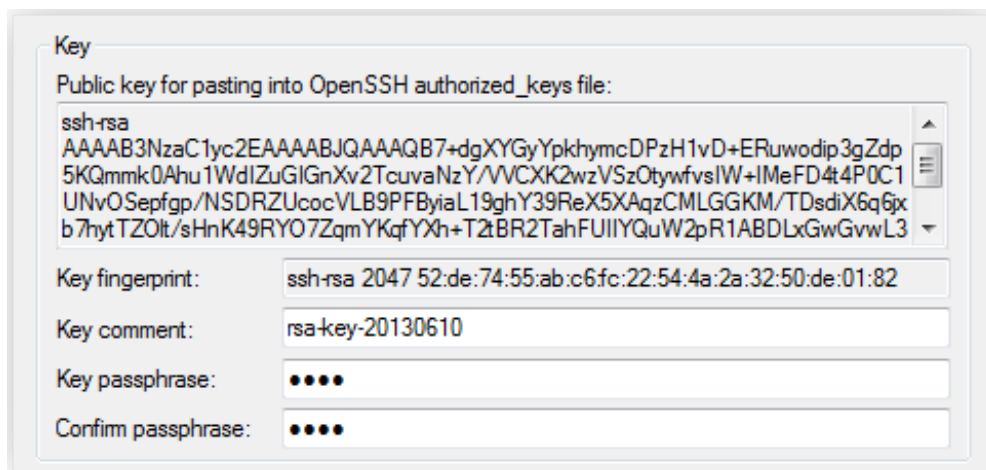
Type of key to generate:

☐ SSH-1 (RSA) ☒ SSH-2 RSA ☐ SSH-2 DSA

Number of bits in a generated key:

Нажимаем на кнопку «Generate».

Далее PuTTYgen создает ключ и предлагает ввести для него пароль:



Key

Public key for pasting into OpenSSH authorized_keys file:

```
ssh-rsa
AAAAB3NzaC1yc2EAAAABJQAAAQBB7+dgXYGyYpkhymcDPzH1vD+ERuwodip3gZdp
5KQmmk0Ahu1WdIZuGIGnXv2TcuvaNzY/VVCXK2wzVSzOtywfvslW+IMeFD4t4P0C1
UNvOSepfgp/NSDRZUcocVLB9PFByiaL19ghY39ReX5XAqzCMLGGKM/TDsdix6q6jx
b7hytTZ0lt/sHnK49RYO7ZqmYKqfYXh+T2BR2TahFUIIYQuW2pR1ABDLxGwGvwL3
```

Key fingerprint:

Key comment:

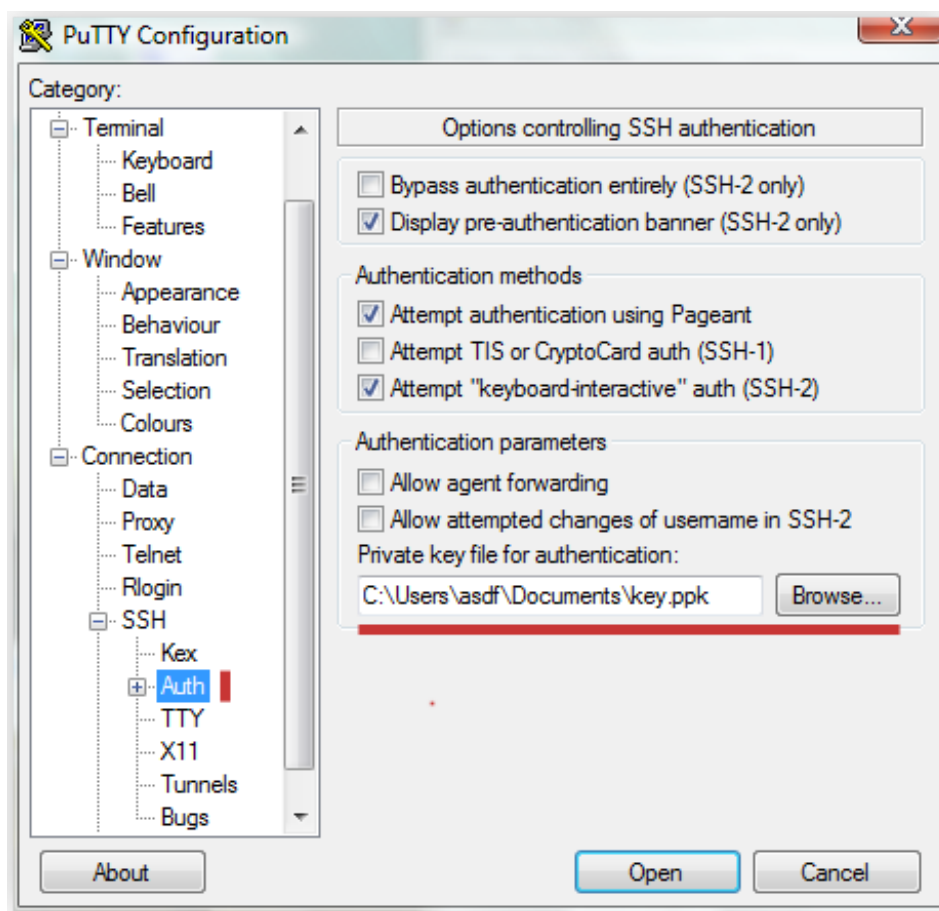
Key passphrase:

Confirm passphrase:

Пароль нужно выбирать такой, чтобы враг его не раскрыл, а вы — не забыли. Также можно ввести комментарий (любой).

Сохраняем private key (закрытый ключ) в файл mykey.ppk, а public key (открытый ключ) — в файл id_rsa.pub.

id_rsa.pub нужно послать нам, а на закрытый ключ нужно натравить Putty, чтоб он знал, откуда его брать. Картинка, я надеюсь, все объясняет:

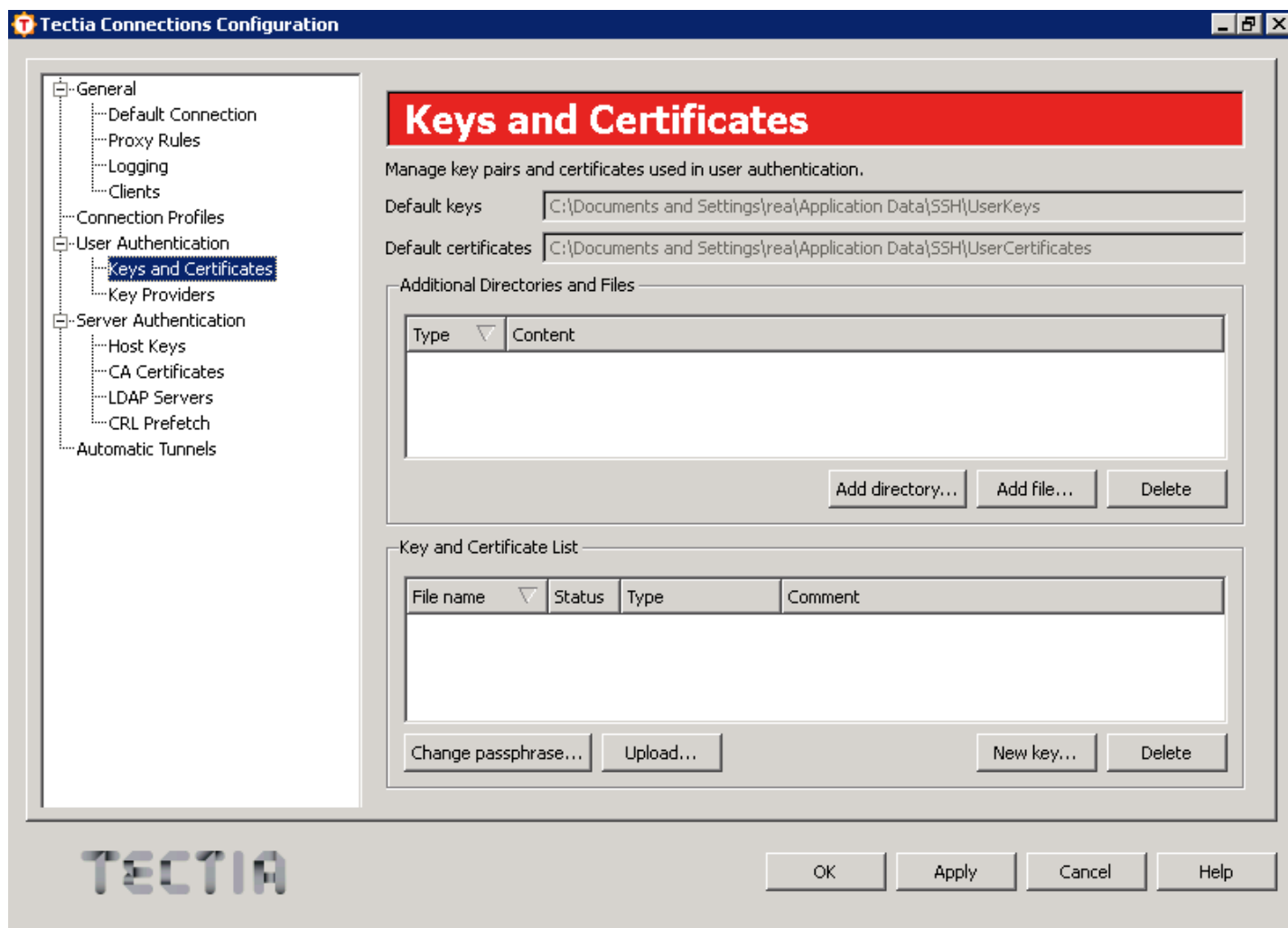


That's all, folks!

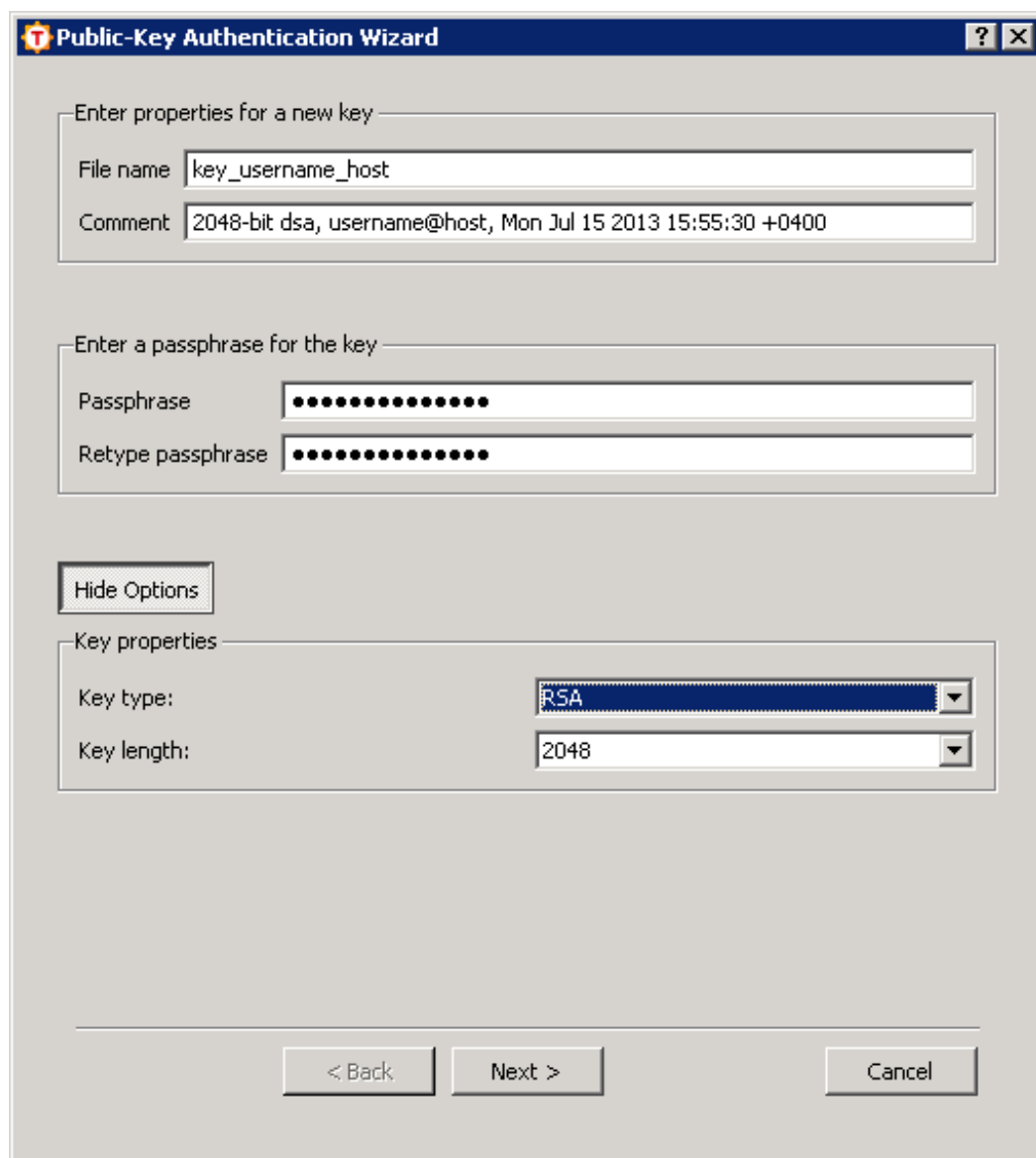
Tectia SSH Client (ssh.com)

Tectia SSH Client — это преемник Windows SSH Client от ssh.com. По нашему мнению, Putty лучше, но некоторые любят этот клиент, поэтому мы вас научим создавать ключи для SSH и в нем.

В меню «Edit» выбираем пункт «Tectia Connections...», в узле дерева слева «User Authentication» выбираем «Keys and Certificates» и получаем примерно следующее окно:



Нажимаем на кнопку «New key...» и получаем следующее окно, в котором необходимо ввести параметры создаваемого ключа:



The image shows a 'Public-Key Authentication Wizard' window. It has a title bar with a red 'T' icon, the text 'Public-Key Authentication Wizard', and standard window controls. The window is divided into three main sections. The first section, titled 'Enter properties for a new key', contains two text boxes: 'File name' with the value 'key_username_host' and 'Comment' with the value '2048-bit dsa, username@host, Mon Jul 15 2013 15:55:30 +0400'. The second section, titled 'Enter a passphrase for the key', contains two text boxes for 'Passphrase' and 'Retype passphrase', both filled with dots. The third section, titled 'Key properties', is preceded by a 'Hide Options' button. It contains two dropdown menus: 'Key type' set to 'RSA' and 'Key length' set to '2048'. At the bottom of the window are three buttons: '< Back', 'Next >', and 'Cancel'.

Public-Key Authentication Wizard

Enter properties for a new key

File name: key_username_host

Comment: 2048-bit dsa, username@host, Mon Jul 15 2013 15:55:30 +0400

Enter a passphrase for the key

Passphrase:

Retype passphrase:

Hide Options

Key properties

Key type: RSA

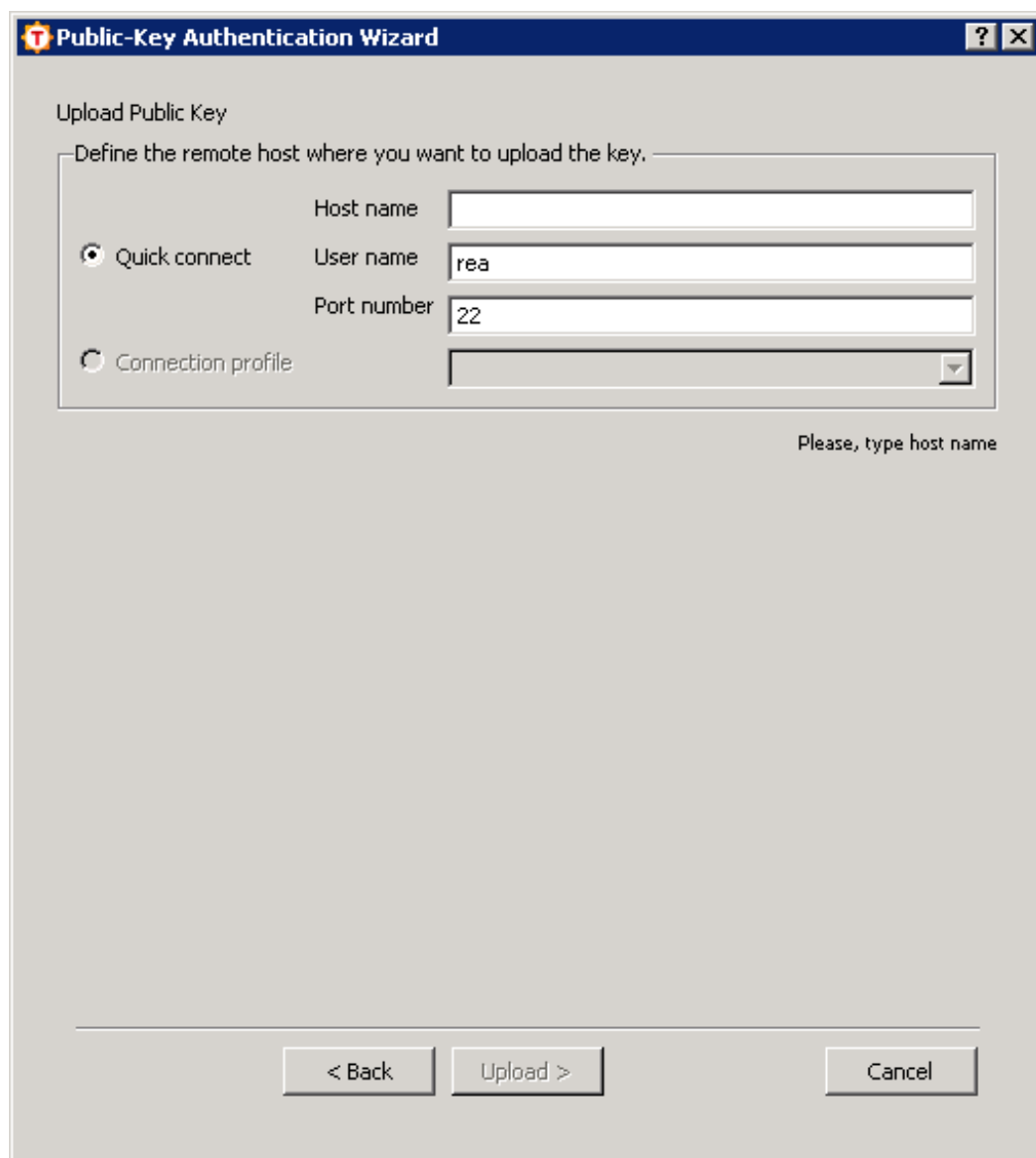
Key length: 2048

< Back Next > Cancel

Ваши действия будут следующими:

- «File name», естественно, отвечает за имя файла с открытым и закрытым ключами, которые будут сгенерированы. Имя нужно ввести.
- В поле «Comment» можно ввести комментарий, например, для чего предназначен этот ключ.
- В поля «Passphrase» и «Retype passphrase» необходимо ввести пароль для закрытого ключа.
- Далее необходимо нажать кнопку «Advanced Options», выбрать тип ключа «RSA», а количество бит — не менее 2048.

Нажав кнопку «Next» вы инициируете создание ключевой пары, что может занять некоторое время. Далее вы увидите следующее окно, предлагающее нам загрузить ключ на сервер:



The image shows a Windows-style dialog box titled "Public-Key Authentication Wizard". It has a blue title bar with a red 'T' icon, a help button (?), and a close button (X). The main area is light gray. At the top, it says "Upload Public Key". Below that, a text label reads "Define the remote host where you want to upload the key." followed by a horizontal line. There are two radio buttons: "Quick connect" (selected) and "Connection profile". To the right of "Quick connect" are three text input fields: "Host name" (empty), "User name" (containing "rea"), and "Port number" (containing "22"). To the right of "Connection profile" is a dropdown menu (empty). At the bottom right of the main area, the text "Please, type host name" is displayed. At the bottom of the dialog, there are three buttons: "< Back", "Upload >", and "Cancel".

Public-Key Authentication Wizard

Upload Public Key

Define the remote host where you want to upload the key.

☒ Quick connect

Host name

User name: rea

Port number: 22

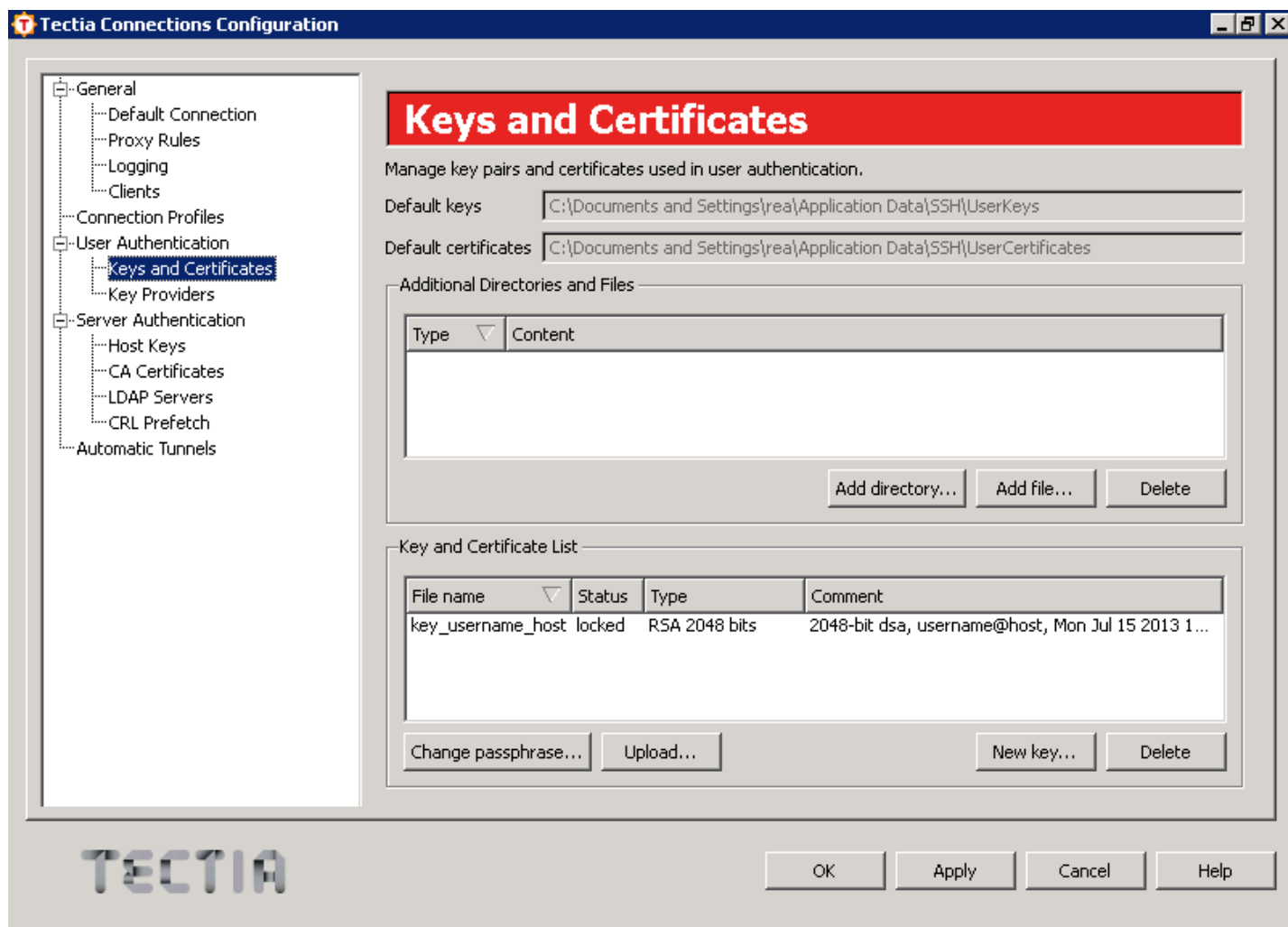
☐ Connection profile

Please, type host name

< Back Upload > Cancel

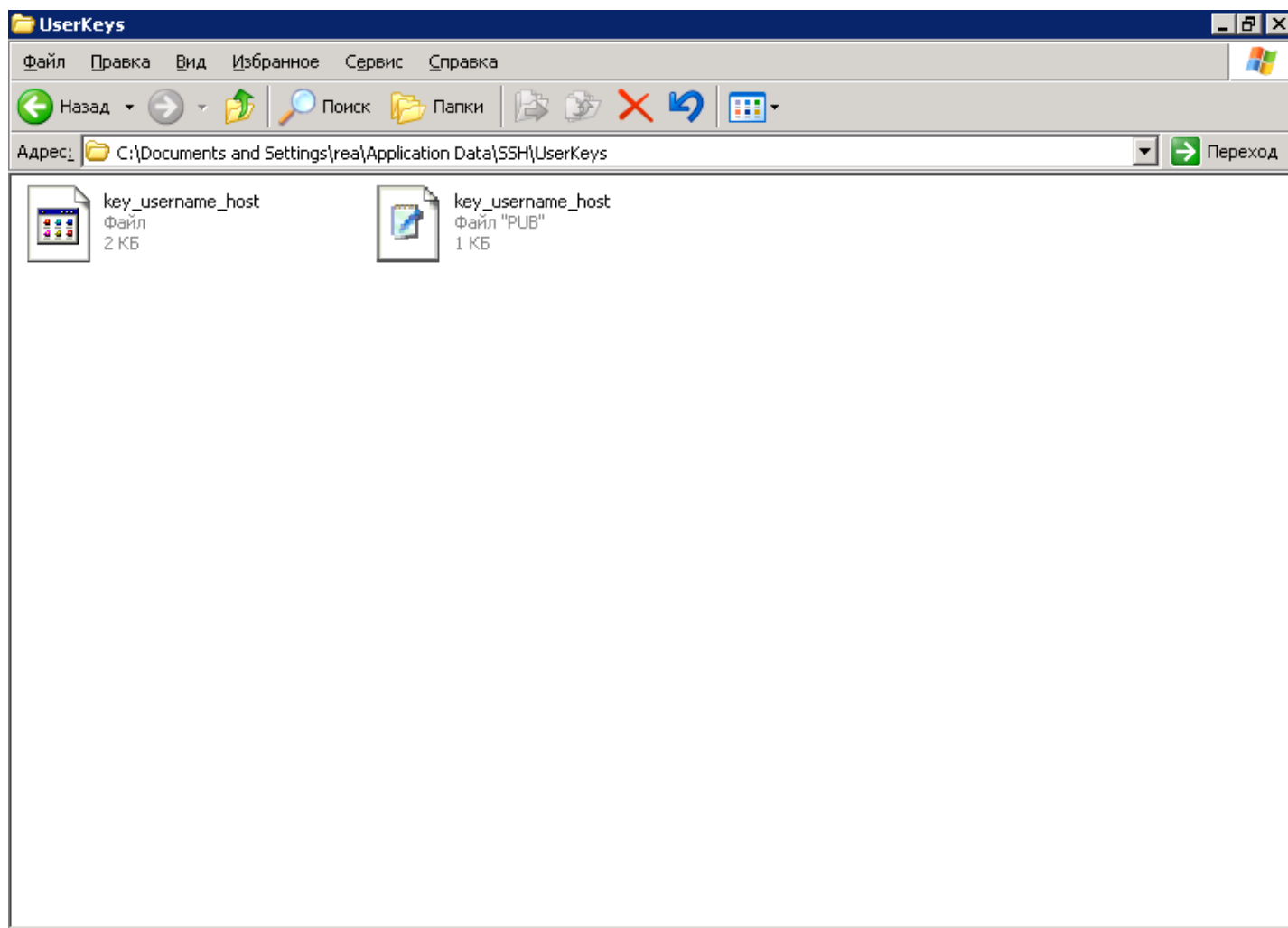
Нажимаем «Cancel», ничего никуда загружать не нужно.

После этого мы видим первоначальное окно,



но уже с одним (или, если у вас до этого были созданы публичные ключи, с еще одним) ключом. Поле «Default keys» сверху окна показывает, где сохраняются ваши открытый и закрытый ключ.

Файл с закрытым ключом не имеет расширения, файл с открытым ключом имеет расширение «.pub»:

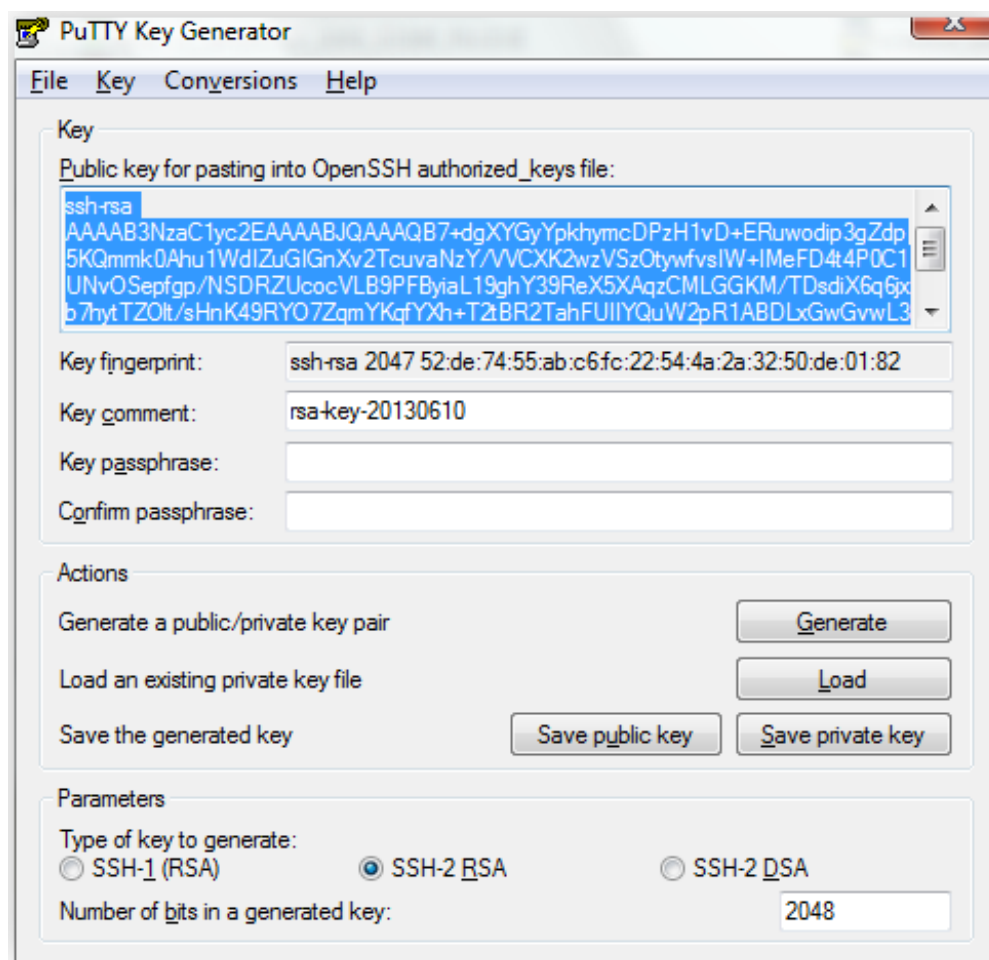


That's all, folks!

Экспорт ключей из Putty/Tectia в формат OpenSSH

Если вам нужно экспортировать закрытый ключ с Windows-системы в Unix и вы использовали Putty или Tectia SSH Client, то сейчас мы объясним, как это можно сделать. Для преобразования необходим Putty key generator, в одном из [предыдущих разделов](#) объясняется, где его взять.

Шаг номер 1: вы загружаете существующий закрытый ключ в Putty key generator (меню «File», пункт «Load private key»). Получается примерно такое окно:



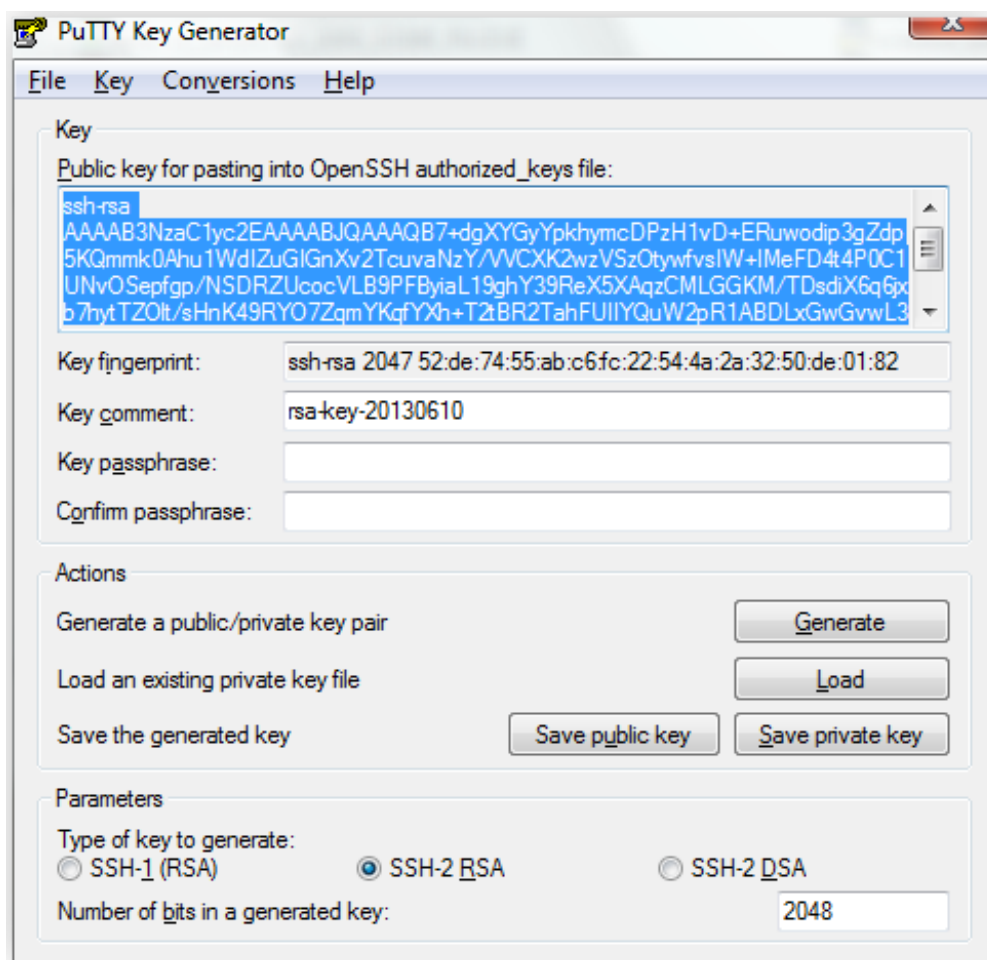
Шаг номер 2: находим в меню «Conversions» пункт «Export OpenSSH key», вводим имя файла и выбираем каталог, в который будет сохранен закрытый ключ. Пароль сохраненного закрытого ключа будет таким же, как и пароль к исходному ключу в формате Putty.

Шаг номер 3: переносим закрытый ключ на Unix-машину, лучше — копированием файлов с использованием какого-то offline-носителя, а не через сеть или электронную почту.

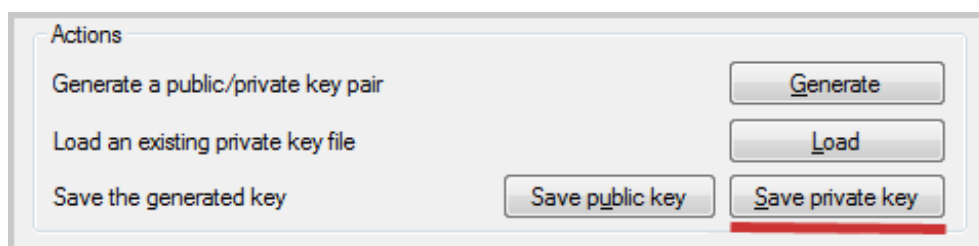
Экспорт ключей из OpenSSH в формат Putty/Tectia

Если вам нужно экспортировать закрытый ключ с Unix-системы использующей OpenSSH в Windows, где вы используете Putty или Tectia SSH Client, то этот раздел — для вас.

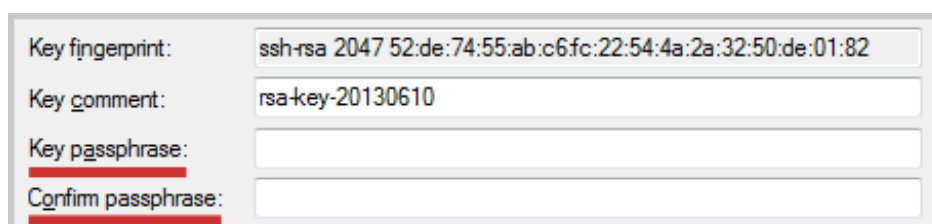
Шаг номер 1: находим в меню PuTTY Key Generator «Conversions» пункт «Import key», выбираем файл с ключом в формате OpenSSH, вводим в открывшемся окне пароль для этого закрытого ключа и получаем примерно такое окно:



Шаг номер 2: нажимаем в этом окне кнопку «Save private key»,



выбираем имя файла и каталог, куда будет сохранен файл закрытого ключа. Пароль для этого файла будет совпадать с паролем исходного ключа. Если вы хотите его изменить, то перед сохранением ключа в формат Putty необходимо ввести новый пароль в поля «Key passphrase» и «Confirm passphrase»:



Типичные проблемы, возникающие при работе с SSH-ключами

При переносе SSH-ключей между различными операционными системами иногда возникает путаница.

Ниже будет показано, как отличить файлы с ключами различных форматов.

Вот как должен выглядеть закрытый ключ для Unix-систем:

```
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4, ENCRYPTED
DEK-Info: DES-EDE3-CBC,F45627B370E24DB1

XD4u/JN7snja3Wl1Dj7eoIGKiAx8IOALvLbmUTik4liCMxk3OPW1eOTQxAV7CJom
PMfH5kOMABer45D4yiQzGYGc/mxStfNjtm8V7j6N4acb6E4BKRUGmv7ZkLrw9csr
PpZzSQ5wTYiXRzJE55r5Q+CWd2V5qVJypAW2Q8pUMGF6A+DSxno1LSc8TwQ43o8G
FTGw8bCEgnfhXmGGLm6lirbXOQIJRCqbszB1PGZ4uNNaLakbs7O85dqjKFsTMXLv
XCloDn+8wFr8eg7mZAKajs0jz9l0PBDAYxivzrw97GRmemHynqsDSaE4F6pxaed9
lhxLwtK0VZH1ofEZUdT64lHJUX4oow9t2r7ajrItlnGBo7redY4yTaLKArxSOdVG
AKOO0PyAv4J2Mm0F8bNwYecUskpu+efr4n2343jJKmXONc+KxfZjdDBFsVFkmpv3
p4AyryKisiT/BbgAdLYXomVKeMSBQe6eDUSQtIabirOBsk63aiJAY5j5yzqzudJ8
IzmCuNB7p4SkmqiPoqW7vPR5G2PlK4G790vszcYr221w5QE3keVB/iRszyJrQH18
k9LzX3oejumGknLubihJqelWBDP54tG1L88YB9uPQ4jSUFov8+M68ebBONVh1eB9
uUe0C3FwxUhm53i7v8DnDFki8onnk6mUxa8pgIJJBUMdDnie0CGh1QAT6NiLE2tT
mnBIG7pHxU3HbnUdA+yDbvvfkoP6Lc5XWCQsn8SNjDv2gPkNrYRbxaRjgOykTiQ
ZHEEU+JYwzyOtw6tbu0OUfch4BqKhVc0YwZrDDKHOXAo0PgGcuxoE8EAKZptV3lz
XyFZjtfw5KvulRmwjuMydu0Alg5qo8cpyCSFCyxRQlWjyiPwOI3w7ixLpZFpQGMQ
bmVrgLKB/XdgPnmXj7K/6KD2YU2FxzjCFdbGdDUY6E/cBAHD/7sHjGV6CXJ72Zko
oGWhLgkRk/Dy9doysm6DwCiLS7K/cddUkZcFKvxzBdmOaTt+jlB2tXKDvRAJIwrm
GiEx2LlKjbbgoTrV+rjuFFgVhsHualxP52NsvujQZVpeFtomZ/amk3ceOMTFTkab
QJcb/zOwjG+PrtiQ6BR/Te0kl44S2L3AR5AOCVD13klNEOZ1yHyCtti04xM7JavP
Jy+RSumIt7hSD9A0e4nHHXEhPZnGgG8ekVrR6FEQ+0FbvYLpv05Ir+igQSMftZwA
YdBjA16KeJL4jKAOWzVe5tdA0BQcJvPjzPK97N6HkCrbcmSy7mQAsXCZ7BcInwwa
UD8zOeH8Ii8atzY+YM+bQRoRfQLkzpJ3pVe68ZwqKFH1+YQNlh0sCJ3slihLKKVR
mUXutpa2c385Yb9djLifKaSDPYG3rutmdx7HY1JzYvvkIau+ixiO1H6dETI8tLZC
4l1FTSisGt4LZyH3WgPpQzhiWs0KX9yIQ/0lqRhgEL8zqqDm3jo/jIQdFMvfHB1I
YSKjoySUN3GXk8MfBsxxgbJRNwfcduB5qcsAxYkNVJgcZHSceOM5NoatlHVlbyV
Ymu8j7BqNZ5a1W/YYadEV2prdQeAUOTX8yGVq14MU/5X6uTZyCj9fQ==
-----END RSA PRIVATE KEY-----
```

Детали могут отличаться, но первая и последняя строчки почти всегда имеют вид

```
-----BEGIN RSA PRIVATE KEY-----
-----END RSA PRIVATE KEY-----
```

либо

```
-----BEGIN DSA PRIVATE KEY-----
-----END DSA PRIVATE KEY-----
```

Если вы пытаетесь использовать для Putty ключ в формате OpenSSH, то вам необходимо его [сконвертировать](#). Без преобразования формата файла Putty не будет правильно распознавать ваш закрытый ключ.

Вот как выглядит закрытый ключ для Putty:

```
PuTTY-User-Key-File-2: ssh-rsa
Encryption: aes256-cbc
Comment: rsa-key-20130610
Public-Lines: 6
AAAAB3NzaC1yc2EAAAABJQAAQAB7+dgXYGyYpkhymcDPzH1vD+ERuwodip3gZdp5
```

```
KQmmk0Ahu1WdIZuGIGnXv2TcuvaNzY/VVCXK2wzVSzOtywfvswIW+IMeFD4t4P0C1
UNvOSepfgp/NSDRZUcocVLB9PFBYiaL19ghY39ReX5XAqzCMLGGKM/TDsdiX6q6j
xb7hytTZolt/sHnK49RYO7ZqmYKqfYXh+T2tBR2TahFUIIYQuW2pR1ABDLxGwGvw
L3T21aIT5Lt3//6ioCT0jlIkBuVii1hlm2quiYqogDs1U1ihOGL0wKNvhpds4Hfd
wyRCjU1WJkZGGrLkdtvlg4AQhM7J8FVrYhTNZe90GJZXjnJP
Private-Lines: 14
Xp4pdSAaE8Ftj9Va68OgwyTtO1V75m7cLmSHJSxzVaQCpw8zzn2gPeFn1/7fuusN
Q1cZ8uDdpzI5E2+iEtM5hSn4SoOfH5eyhSa+FYObxj2IN+y1wS4vIbPmj7pY4Mom
BGvxPdGZzKSITblxiE6YSFCNq4CmTIjtg/OXU5/slw7oifdEPcvPDudaGtztmv4a
zSMPO9tvgrk1C5hzY5FfDoKpZHK94mgYjypHt7NEAbEls7GQCn7qTsng2Au2xeu/
15ZDmb52zu/iLYb2JALnz+h5qyE2XFHsVjwyY+rzxstEWgkgCZdCrbOexHrgIlyc
sKr7H3c3pJ7ZsVsXFvowLVRcFhq/8Sa7rTr7ONkaUBPuiF00fdp2UiC3bI5U9F1Q
MRhXKEfogEdJDZdWvGibHyX2on8PkJzDZCuQaG/K6pm92VHhv8AYMs4ADfrUVg6k
Y7A5q+sDtXM1QAiL/Qct1Hr8uAp8sLfKYsnBSc1teTHdIjugEjfOn8zMR96vgFEi
bXWcA8o4WROMJ42f9Dy3hTsg7kf4ZcGmY8ua3deYABu3KuAttUNTshAF5qfPGjEk
gR53PoPgLzA1IeYIgX9w9INHqgfolSgWuuTb5i1UdXKnk7SwXxmy11o9F3DSRGU9
p17bKLQGjrDcaLyxavUyZoHb1zwiDoGBJ4pefC1e1LKJ6OUiWXCWcUsTieuj9q+L
eEr4RWrr59A/11jGvbrTi2Qjxp71aqHmQjsiXonT/bl6Xydb3zS8e4AmJJifxu179
EgfXBPePYarjX6PPivt6uURnRDyTVU6jXJl4ddIudtpUpY14M4BekA4MZ9m2PMsx
S5syWlWFvfPIoHMLpdDWV/D2kimvamcEbbCxK0uI9VlVHNmTotI49OYt0XaHrZGS
Private-MAC: 464d517a6c7b61184a4f7f1ec2ad4f33c1b15294
```

Характерными особенностями данного формата является первая строка вида

```
PuTTY-User-Key-File-2: ssh-rsa
```

вслед за которой довольно часто следуют строки вида

```
Encryption: aes256-cbc
Comment: rsa-key-20130610
```

Если вы пытаетесь использовать на Unix-системах ключ в формате Putty, то вам необходимо его [сконвертировать](#). Без преобразования формата файла OpenSSH не будет правильно распознавать ваш закрытый ключ.

Ваши пароли и SSH-ключи

Ваши пароли и закрытые части SSH-ключей не нужно показывать или передавать никому. Вообще никому, даже если вас об этом просит служба поддержки или администратор, с которым вы переписываетесь или разговариваете по телефону. Любой, кто спрашивает вас о вашем пароле — это человек, который пытается методами социальной инженерии вытащить из вас секретные сведения и пробраться на наш кластер. **Исключений — нет, пожалуйста, запомните это.**

Также не нужно давать ваш пароль и/или SSH-ключ вашим коллегам, близким, собакам, кошкам и прочей живности. Если кто-то хочет получить доступ на наш кластер, то он должен заполнить [заявку](#) и мы ее обязательно рассмотрим.

Если вы все же решите передать кому-то ваш пароль или SSH-ключ и это обнаружится, то доступ для вас на наши ресурсы будет закрыт до момента выяснения всех обстоятельств, а, возможно, даже и навсегда.

Установленное ПО

По поводу того, как настраивается окружение для разработки, написано в разделе [настройка рабочего окружения](#).

Компиляторы

- Intel C/C++ Compiler 14.0.2: компилятор запускается командой `icc`. Для компиляции MPI-приложений нужно использовать команду `mpicc`.
- Intel Fortran Compiler 14.0.2: компилятор запускается командой `ifort`. Для компиляции MPI-приложений нужно использовать команды `mpif77` и `mpif90`.

Библиотеки MPI

- Platform MPI 9: может быть использован как с программами, скомпилированными с помощью Intel C/C++ и Intel Fortran, так и с программами, скомпилированными с помощью GNU C/C++.

Доступные файловые системы

На наших суперкомпьютерных ресурсах в данный момент используется два различных типа файловых систем, NFS и Lustre.

Домашняя файловая система

Для поддержки домашних каталогов пользователей (тех каталогов, в которые вы попадаете при заходе на login-узлы) у нас на данный момент используется NFS. Эти каталоги доступны только с login-узлов, вычислительное поле их не видит. Все login-узлы всех кластеров видят одно общее файловое пространство на NFS. Домашние каталоги на NFS не подвергаются периодической очистке, но на данный момент у нас нет механизма создания резервных копий данных, поэтому копии дорогих вам данных, исходных кодов и сценариев запуска рекомендуется периодически копировать на свои рабочие машины и хранилища.

У нас есть пользовательские каталоги, доступные по имени `/home/users/$USERNAME`, и групповые каталоги, доступные как `/home/groups/gABCD` для чтения и записи всем членам указанной группы. На NFS у нас установлены групповые квоты для полного пространства, которое занимают как индивидуальные пользовательские каталоги, так и групповой каталог. Квоты ставятся в соответствии с вашими заявками при регистрации группы, они могут быть расширены (разумным образом), если это необходимо. По вопросам расширения нужно писать в нашу службу поддержки help@computing.kiae.ru.

Технически NFS располагается на нескольких независимых серверах и домашний каталог каждого из пользователей находится на конкретном сервере. Это означает, что при проблемах с одним NFS-сервером пострадают все пользователи с домашними каталогами, располагающимися на нём, но другие пользователи будут иметь возможность продолжать работу.

Параллельная файловая система

У каждого из суперкомпьютеров есть локальная параллельная файловая система, основанная на Lustre. Она доступна как с login-узла этого суперкомпьютера, так и с узлов его вычислительного поля и должна использоваться для хранения временных данных расчётов и программного обеспечения. Различные кластеры не видят параллельных файловых систем друг друга.

Организация каталогов на Lustre сходна с домашней файловой системой:

- есть каталоги для каждого пользователя, они доступны по имени `/s/l2/users/$USERNAME`,
- существуют групповые каталоги, доступные как `/s/l2/groups/gABCD`

Эти каталоги предназначены для хранения временных данных в процессе вычислений и переноса входных/выходных данных с/на домашнюю файловую систему. (В не столь отдалённом будущем) они будут периодически очищаться от старых файлов.

Для хранения более постоянных объектов (например программного обеспечения, каких-то вспомогательных данных, которые часто используются для многих вычислительных задач, и т.п.) у нас предусмотрены каталоги с немного другими именами:

- каталоги для каждого пользователя называются /s/ls2/u-sw/\$USERNAME,
- каталоги для группы называются /s/ls2/g-sw/gABCD.

Для того, чтобы не особенно ограничивать пользователей в объёмах промежуточных файлов (задачи разные бывают), на Lustre квот нет. Поэтому, пожалуйста, будьте разумны в плане занимаемого там места и стирайте, либо переносите на NFS всё, что вам не очень потребуется.

Копирование данных между файловыми системами

Предполагается, что вы будете копировать ваши данные между NFS и Lustre вручную (командами `cp`, `rsync` и другими).

В-принципе, вы можете просто работать на самой Lustre, но поскольку она используется для хранения данных, к которым обращается значительная часть вычислительного поля, и сама Lustre проектировалась именно как параллельная файловая система, вы можете испытывать проблемы со скоростью работы с файлами: у вас долго будет работать команда `ls`, файлы в редакторе также будут долго открываться и сохраняться. Поэтому вам, скорее всего, будет сильно удобнее использовать Lustre и домашнюю файловую систему по их назначению.

Настройка программного окружения

В связи с тем, что на кластере могут использоваться различные комбинации установленного программного обеспечения, у нас используется несложная система настройки программного окружения для каждого пользователя. Называется она `modules`.

Например, чтобы воспользоваться компиляторами от Intel, выполните команду:

```
module load intel-compilers
```

а чтобы скомпилировать параллельную задачу с использованием библиотеки MPI, выполните:

```
module load mpi
```

Всё это можно сделать одной командой:

```
module load intel-compilers mpi
```

Следующая команда позволяет узнать какие пакеты имеются в системе

```
module avail
```

а команда

```
module list
```

выдает список уже используемых в данном сеансе модулей.

Более подробную информацию о системе «`modules`» можно узнать, запустив команду `man module`.

Для настройки пользовательского окружения на головных машинах вполне достаточно поместить в файл `~/.bash_profile` следующие строки:

```
module load mpi intel-compilers
```


Эта строка говорит о том, что пользователь будет использовать библиотеки MPI (mpi) и компиляторы Intel C/C++ и Intel Fortran (intel-compilers). Естественно, не обязательно использовать именно эти конкретные инструменты, указанная строка является лишь примером.

В сценариях запуска для задач нужно указывать те же строки, что и в ~/.bash_profile. В разделе про запуск задач будут приведены соответствующие примеры.

Запуск задач

Запуск задач осуществляется командой sbatch. Совсем подробную документацию можно посмотреть командой man sbatch, здесь будет рассмотрена только базовая функциональность.

Запуск простого сценария

Важно: batch-система умеет запускать только сценарии, поэтому не пытайтесь вместо сценариев запустить исполняемый файл: чаще всего из этого ничего не выйдет.

Самый простой пример: пусть мы хотим запустить на выполнение сценарий test.sh, находящийся в текущей директории и содержащий следующие строки:

```
#!/bin/sh
#SBATCH -D /s/ls2/users/eygene
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -t 01:00:00
#SBATCH -p hpc2-16g-3d
hostname
df
date
sleep 10
date
```

Этот сценарий должен быть сделан исполняемым:

```
chmod +x test.sh
```

Запускаем сценарий на выполнение командой sbatch test.sh;

```
$ sbatch test.sh
sbatch: Submitted batch job 19
```

В ответ нам выдали идентификатор нашей задачи в очереди — 19.

Пользуясь этим идентификатором мы можем, например, узнать текущий статус нашей задачи:

```
$ squeue -j 19
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST (REASON)
19	hpc2-16g-3d	test.sh	eygene	PD	0:00	2	(JobHeld)

В данном случае задача пока не была запущена. Вот как выглядит состояние запущенной задачи:

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST (REASON)
19	hpc2-16g-3d	test.sh	eygene	R	0:01	2	n[29-30]

Значение различных полей таблицы должны быть достаточно понятны.

Командой scancel можно удалить задачу из очереди: scancel 19.

После успешного завершения нашей задачи в каталоге `/s/ls2/users/eygene`, из которого задача запускалась образуются файлы `19.out` и `19.err`. В них было записано содержимое потоков стандартного вывода и стандартной ошибки вашей программы. Файлы будут находиться в каталоге `/s/ls2/users/eygene`, поскольку в сценарии мы написали

```
#SBATCH -D /s/ls2/users/eygene
```

что заставляет Slurm при запуске задания переходить в указанный каталог. Этот каталог должен существовать и находиться на параллельной файловой системе, поскольку задание выполняется на рабочих узлах, которые из файловых систем видят только параллельную (про это подробнее написано [здесь](#)).

Файлы будут называться `19.out` и `19.err` поскольку мы сказали

```
#SBATCH -o %j.out
```

```
#SBATCH -e %j.err
```

В этих директивах «%j» заменяется на идентификатор задачи, все остальное — оставляется как есть.

Директива

```
#SBATCH -t 01:00:00
```

указывает на максимальное время выполнения данной задачи (в данном случае — один час). Если вы можете более-менее точно оценить верхнюю границу этого времени, пожалуйста, указывайте ее: это позволит планировщику более разумно планировать задачи. Однако вы должны понимать, что если ваша задача будет «убита», если время ее выполнения превысит указанное. Поэтому максимальное время выполнения лучше указывать с небольшим запасом.

Директива

```
#SBATCH -p hpc2-16g-3d
```

указывает, что мы хотим запустить задачу в очередь «hpc2-16g-3d». Чтобы понять, какую очередь нужно использовать для ваших нужд, пожалуйста, обратитесь к разделу «[Список очередей и политика их использования](#)»

Кстати говоря, вышеприведенные строчки вполне эквивалентны передаче утилите `sbatch` параметров командной строки «`-o . -e .`». И это работает для любых параметров `sbatch`, не только для указанных.

Запуск одиночной задачи

Запускать сценарии — это очень здорово, но чаще приходится запускать уже откомпилированные программы, подавать им на вход какие-то файлы и забирать результаты, которые сохраняются в другие файлы.

Но всё это несложно: передать программе любые аргументы можно из сценария, поэтому остаётся только вопрос копирования файлов на машину (с машины), где задача будет выполняться. Но это не проблема: на нашем кластере используется разделяемая файловая система, поэтому все узлы кластера видят одно файловое пространство. Поэтому вы спокойно можете запускать вашу задачу из любого каталога на головном узле — нижеприведенный сценарий запуска позаботится о том, чтобы задача «увидела» все те же файлы, которые есть в этом каталоге.

Вот какой файл сценария нужно использовать:

```
#!/bin/sh
```

```
#SBATCH -D /s/ls2/users/eygene/my-precious-task
```

```
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -t 01:00:00
#SBATCH -p hpc2-16g-3d
```

```
module load intel-compilers
```

```
`pwd`/program mytask.in | tee mytask.log."$SLURM_JOBID"
```

Здесь предполагается, что запускаемая программа называется “program” и она принимает на вход один аргумент — имя входного файла; в нашем случае “mytask.in”.

Конструкция “| tee mytask.log.”\$SLURM_JOBID”” применяется для того, чтобы весь вывод программы перенаправлялся в файл с именем “mytask.log.идентификатор_задачи”. Для программ, которые выдают результаты своей работы на экран это удобно, поскольку можно контролировать работу программы просматривая содержимое указанного файла. Если ваша программа ничего не выводит на экран, то эту конструкцию можно опустить, оставив только “`pwd`/program mytask.in”.

Запуск параллельной задачи

Предполагается, что вы ознакомились с разделом [запуск одиночной задачи](#). Материал этого раздела лишь дополняет рассказанное там.

Как рассказывалось в разделе [настройка программного окружения](#), в сценарии запуска задачи окружение должно настраиваться точно так же, как и как в файле ~/.bash_profile. Будем предполагать, что вы используете следующие директивы команды module load: mpi и intel-compilers. Поэтому первым отличием сценария запуска MPI-задачи является наличие строк

```
module load mpi intel-compilers
```

Строго говоря, по-сравнению с сценарием для одиночной задачи тут появляется только загрузка модуля «mpi».

Следующим отличием является добавление еще одной строчки вида «#SBATCH ...»:

```
#SBATCH -n 100
```

Эта директива говорит о том, что вам нужно 100 ядер.

Последней модификацией в сценарии является добавление слова “mpisub” в начало команды запуска задачи.

Собирая всё вместе мы получаем следующий сценарий запуска задачи:

```
#!/bin/sh
#SBATCH -D /s/ls2/users/eygene/my-precious-task
#SBATCH -n 100
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -t 01:00:00
#SBATCH -p hpc2-16g-3d
```

```
module load mpi intel-compilers
```

```
$MPIRUN `pwd`/program mytask.in | tee mytask.log."$SLURM_JOBID"
```

Список очередей и политика их использования

Все очереди у нас разбиты на классы: каждый класс содержит в себе несколько очередей, которые различаются по максимальному времени выполнения задачи и общему приоритету задач, но живут на одних и тех же физических ресурсах.

Общая стратегия такая: чем больше максимальное время выполнения задачи в очереди, тем меньше приоритет задач в этой очереди, при прочих равных. Это сделано для того, чтобы вы, уважаемые пользователи, помещали короткие задачи в подходящие очереди, планировщик знал, когда эти задачи точно закончатся и лучше планировал ресурсы.

Кластер HPC2

В данный момент на кластере организовано два класса очередей для локальных пользователей:

- «debug»:
 - класс представлен одной отладочной очередью для параллельных и последовательных задач, «hpc2-debug-10m»;
 - очередь состоит из одного узла с двумя четырехядерными процессорами Intel Xeon E5450 с тактовой частотой 3.00 ГГц и 16 Гбайт оперативной памяти;
 - все пользователи нашего кластера могут запускать задачи в эту очередь;
 - любой пользователь может поставить в эту очередь не более одной задачи;
- «hpc2-16g»:
 - класс очередей для параллельных и однопроцессорных задач;
 - очереди состоят из одинаковых счётных узлов с двумя четырехядерными процессорами Intel Xeon E5450 с тактовой частотой 3.00 ГГц и 16 Гбайт оперативной памяти;
 - задачи разных пользователей получают различный приоритет, который влияет на время ожидания задач в очереди;

Для класса debug есть только одна очередь, debug-10m, с пределом времени счета в 10 минут;

Для класса hpc2-16g есть следующие очереди:

- hpc2-16g-3d: очередь для задач, длящихся не более трех дней;
- hpc2-16g-1w: очередь для задач, длящихся не более недели;
- hpc2-16g-1m: очередь для задач, длящихся не более месяца (31 день); доступ в эту очередь открывается только по договоренности.

Если вам необходимо использовать очереди с ограниченным доступом, обратитесь в нашу службу поддержки пользователей help@computing.kiae.ru, обосновав необходимость.

Кластер HPC4

В настоящий момент на кластере есть единая очередь для вычислений с использованием графических сопроцессоров NVidia K80, она называется hpc4-gpu-3d, максимальный срок жизни задачи в ней — 3 дня.

Общие ограничения ресурсов в очередях

Во всех очередях действуют ограничения на размер виртуальной памяти для одного процесса. Эти пределы работают так, что если вы пытаетесь распределить памяти в сумме больше установленного

предела, системные функции распределения памяти вернут нулевой указатель. Что с этим делать решает сама задача, никто ее принудительно не убивает: задача

- либо завершается сама, если она не может пережить отсутствия нужного количества памяти
- либо продолжает считать, если она без этой памяти сможет обойтись
- либо выпадает в core, если автор программы не проверяет, выделили ли память или нет, а просто использует возвращенный указатель

Текущие ограничения на виртуальную память: 1,945,600 Кбайт виртуальной памяти на процесс.

Образцы цитирования MBK для публикаций

По условию использования ресурсов MBK руководитель группы и пользователи должны в публикациях результатов, полученных с использованием ресурсов MBK, ссылаться на MBK. Ниже приводятся примеры цитирования, действующие с начала 2016 года:

Данная работа была выполнена с использованием высокопроизводительных вычислительных ресурсов федерального центра коллективного пользования в НИЦ «Курчатовский институт»,
<http://computing.kiae.ru/>.

This work was carried out using high-performance computing resources of federal center for collective usage at NRC “Kurchatov Institute”, <http://computing.kiae.ru/>.

Иногда задаваемые вопросы

Как узнать, на каких узлах выполнялась программа

Список узлов содержится в переменной окружения SLURM_JOB_NODELIST. Значением этой переменной может быть, например «n[14,26-29]». Это означает, что задача выполнялась на узлах n14, n26, n27, n28 и n29.

Поэтому простейшим из способов сохранить эти данные в выводе вашей задачи будет добавление следующей строки в сценарий запуска задачи:

```
echo "CPU list: $SLURM_JOB_NODELIST"
```

Строчку можно добавить, например, после директив «module load».

Каким образом распределяются физические машины для каждой задачи

Чтобы различные пользовательские задачи не могли «отъесть» память друг у друга, каждой задаче выделяются свои физические машины. Скажем, запросив 20 ядер, вы получите 3 физических узла, на которых вам будет выделено 8+8+4 ядер (если каждый узел оснащен восемью процессорными ядрами). Никого больше на эти узлы, пока вы на них считаете, не пустят. Из этого следуют две вещи:

1. Количество процессов лучше выбирать кратным количеству ядер на узлах.
2. Иногда может быть так, что количество занятых всеми задачами ядер не равно полному количеству ядер в кластере, но тем не менее, все запускаемые задачи встают в очередь: для них просто может не быть свободных физических машин.

Как получить образ памяти (core dump) для программ, скомпилированных с помощью Intel Fortran.

Образ памяти иногда необходим для анализа ошибок в программе. По-умолчанию, для всех ошибок «severe» в Intel Fortran образ памяти на диск не сбрасывается. Если вам нужно получить этот образ, то

нужно установить переменную окружения `decfort_dump_flag` в значение «y»:

```
export decfort_dump_flag=y
```

Скорее всего вы также захотите получать некоторую информацию об именах функций, номерах строк в исходных файлах и тому подобном. Для этого нужно скомпилировать отладочный образ вашей программы. Это можно сделать, указав при компиляции и компоновке флаг '-g', оповещающий инструментарий о том, что в исполняемый файл требуется включить отладочную информацию.

Кстати говоря, для Intel Fortran отладочный образ имеет еще одно свойство: при возникновении системных ошибок на экран выдается информация о последовательности вызова процедур, в которой будут присутствовать имена переменных, процедур и исходных файлов. Без использования отладочного образа, на месте перечисленных имен будет стоять грустное слово «Unknown».

У меня не работает самая простая MPI-программа на Fortran. Спасите!

Есть очень простая программа, которая выглядит так:

```
program mpi_probe
  include 'mpif.h'
  integer :: irank, isize, ierr
  call MPI_INIT(ierr)
  call MPI_COMM_RANK(MPI_COMM_WORLD, irank, ierr)
  call MPI_COMM_SIZE(MPI_COMM_WORLD, isize, ierr)
  print *, 'irank isize', irank, isize
  call MPI_FINALIZE(ierr)
end
```

Программа запускается, но сразу валится в segmentation fault со следующей диагностикой:

```
forrtl: severe (174): SIGSEGV, segmentation fault occurred
```

Image	PC	Routine	Line	Source
libc.so.6	000000358F72E2B0	Unknown	Unknown	Unknown
libmpi.so.1	0000002A95755D7B	Unknown	Unknown	Unknown
libmpi.so.1	0000002A9575EE08	Unknown	Unknown	Unknown
a.out	0000000000402CFB	Unknown	Unknown	Unknown
a.out	0000000000402CA2	Unknown	Unknown	Unknown
libc.so.6	000000358F71C3FB	Unknown	Unknown	Unknown
a.out	0000000000402BEA	Unknown	Unknown	Unknown

Причин этому, конечно, может быть целая куча. Но одна из самых часто встречающихся ошибок следующая: у пользователя в той же директории, из которой компилируется программа, есть файл `mpif.h`. Причем файл этот, скорее всего, не от HP-MPI или от HP-MPI, но другой версии. Поэтому не совпадают прототипы функций, типы переменных и вообще, все плохо.

Мораль: таскать за собой какой-то `mpif.h` не нужно, даже если вы знаете, что он от нашей версии HP-MPI (чего вы знать не можете, поскольку версия иногда меняется). В системе уже есть `mpif.h` и он идет в комплекте с библиотекой HP-MPI. Более того, он автоматически берется из правильного места, если компилятор видит инструкцию «`include 'mpif.h'`». Поэтому об `mpif.h` заботиться не нужно. Более того, таскать этот файл за своей программой совершенно бессмысленно, а даже и вредно: система найдет этот заголовочный файл сама, поскольку данный файл является системным, принадлежит конкретной версии конкретной библиотеки MPI и ни в коем случае не является частью вашей программы.

Статические массивы размером более 2GB в Intel Fortran.

Если при компоновке вашей программы возникают сообщения следующего сорта

```
/tmp/fort0ivMR0.o(.text+0xda): In function `output_t_':
: relocation truncated to fit: R_X86_64_PC32 eos_par_mp_dens_
/tmp/fort0ivMR0.o(.text+0x18a): In function `output_t_':
: relocation truncated to fit: R_X86_64_PC32 eos_par_mp_dens_
/tmp/fort0ivMR0.o(.text+0x18f): In function `output_t_':
: relocation truncated to fit: R_X86_64_32 eos_par_mp_dens_
/tmp/fort0ivMR0.o(.text+0x386): In function `output_t_':
: relocation truncated to fit: R_X86_64_PC32 eos_par_mp_dens_
/tmp/fort0ivMR0.o(.text+0x46d): In function `output_t_':
: relocation truncated to fit: R_X86_64_PC32 eos_par_mp_dens_
/tmp/fort0ivMR0.o(.text+0x472): In function `output_t_':
: relocation truncated to fit: R_X86_64_32 eos_par_mp_dens_
/tmp/fort0ivMR0.o(.text+0x59a): In function `output_t_':
: relocation truncated to fit: R_X86_64_32 eos_par_mp_dens_
/tmp/fort0ivMR0.o(.text+0x5cd): In function `output_t_':
: relocation truncated to fit: R_X86_64_PC32 eos_par_mp_dens_
/tmp/fortwpWEfe.o(.text+0xc): In function `pm_':
: relocation truncated to fit: R_X86_64_PC32 tabhad_
/tmp/fortwpWEfe.o(.text+0x5e): In function `pm_':
: relocation truncated to fit: R_X86_64_32S tabhad_
/tmp/fortwpWEfe.o(.text+0x6a): In function `pm_':
: additional relocation overflows omitted from the output
```

то, скорее всего, у вас используются статические массивы размером более 2GB. Чтобы такие программы правильно собирались, к ключам компилятора на стадии сборки самой программы нужно добавить параметры «-mmodel medium -shared-intel».

Еще одним вариантом является использование ключа «-mmodel large», но этот ключ для большинства программ не нужен и приводит только к увеличению их размера. Подробности можно узнать на [техническом форуме Intel](#).

Как заставить работать FTP в Midnight Commander.

По-умолчанию, в Midnight Commander FTP работает в [активном режиме](#). Настройки нашего кластера не позволяют работать в данном режиме, поэтому нужно попросить Midnight Commander использовать пассивный режим. Это очень просто: открываем файл «.mc/ini», ищем там раздел «[Midnight-Commander]» и добавляем туда строку

```
ftpfs_use_passive_connections=1
```

Если переменная «ftpfs_use_passive_connections» уже была определена, то заменяем её значение на единицу, как показано выше.

В результате, файл «.mc/ini» будет выглядеть так (многоточиями отмечены пропущенные строки, не имеющие отношения к нашей проблеме):

```
...
[Midnight-Commander]
...
ftpfs_use_passive_connections=1
...
```

Sbatch выдает сообщение «No partition specified or system default partition»

Такое сообщение возникает, когда вы не указали очередь, в которую хотите запустить свою задачу. Это нужно делать ключом командной строки «-р имя_очереди» или директивой SBATCH в файле сценария запуска:

```
#SBATCH -p имя_очереди
```

При компиляции программ на языке Fortran компилятором Intel выдается предупреждение о функции feupdateenv

При компиляции программ на Fortran компилятор фирмы Intel может выдавать следующее предупреждение:

```
/opt/intel/ifc/9/lib/libimf.so: warning: warning: feupdateenv is not implemented and will always fail
```

Это всего лишь предупреждение и оно не влияет на нормальную работу программ. Вот объяснение от компании Intel, находящееся в заметках к компилятору Intel C/C++ 9.x:

```
In some earlier versions of Intel C++ Compiler, applications built for Intel EM64T linked by default to the dynamic (shared object) version of libimf, even though other libraries were linked statically. In the current version, libimf is linked statically unless -i-dynamic is used. This matches the behavior on IA-32 systems. You should use -i-dynamic to specify the dynamic Intel libraries if you are linking against shared objects built by Intel compilers.
```

A side effect of this change is that users may see the following message from the linker:

```
warning: feupdateenv is not implemented and will always fail
```

This warning is due to a mismatch of library types and can be ignored. The warning will not appear if -i-dynamic is used.

Я поместил в .bash_profile «module load ...», но при заходе на машину выдается сообщение об ошибке

Файл ~/.bash_profile в редакторе vi выглядит следующим образом:

```
module load openmpi intel-compilers
~
~
~
".bash_profile" [dos] 2L, 54C
```

При заходе на кластер выдается сообщение об ошибках:

```
: No such file or directory
-bash: module: command not found
```

Причина проста: файл .bash_profile был создан в DOS/Windows и затем был перенесен в Unix. Это видно по наличию символов «[dos]» в строке статуса редактора vi. Нужно отметить, что реальное содержимое файла ~/.bash_profile особенного значения не имеет — важно то, что сам файл был создан в DOS/Windows, поскольку именно это приводит к ошибкам.

Решение проблемы простое: нужно перекодировать файл из DOS/Windows в Unix-кодировку. Для этого

достаточно набрать команду «fromdos имя_файла», которая перекодирует указанный файл из кодировки DOS (CP866) в кодировку Unix (KOI8-R). В нашем случае нужно дать команду «fromdos ~/.bash_profile».

Программа на языке Fortran завершается, говоря «MPI_ERR_TYPE: invalid datatype»

Скорее всего, в вашей программе не объявлены параметры MPI_REAL, MPI_INTEGER и другие, отвечающие за тип передаваемых данных при приеме/пересылке. Побороть это возможно включением инструкции "include 'mpif.h'" в вашу программу. Естественно, причина может быть еще более простой: вы указали неправильный (или незарегистрированный) тип данных MPI.

Обратите, пожалуйста, внимание, что в программе следующего вида

```
program main
include 'mpif.h'

print *, "m = ", mpi_real
call test
end

subroutine test
print *, "m-test = ", mpi_real
end
```

значение переменной «mpi_real» внутри подпроцедуры «test» не будет совпадать с (правильным) значением «mpi_real» внутри «main». Причина проста: область видимости «правильной» mpi_real ограничена программой «main». Если хочется использовать «mpi_real» внутри подпроцедуры «test», то нужно написать так:

```
program main
include 'mpif.h'

print *, "m = ", mpi_real
call test
end

subroutine test
include 'mpif.h'
print *, "m-test = ", mpi_real
end
```

Работа с кириллическим текстом

На головных узлах кластера по-умолчанию настроена кодировка KOI8-R. Поэтому, если ваша клиентская программа понимает кириллические кодировки, достаточно указать ей данную кодировку и вы получите возможность работать с родным языком ;))

Другой возможной проблемой является то, что файлы, которые вы копируете на головные узлы кластера, могут быть в кириллических кодировках, отличных от KOI8-R. Но это поправимо: на головных узлах установлены программы «fromwin», «towin», «fromdos» и «todos». Как видно из их имен, эти программы умеют преобразовывать файлы из/в кодировки Windows (CP1251) и DOS (CP866). Любой из этих программ достаточно указать в качестве аргумента имя файла и этот файл будет преобразован в соответствующую кодировку.

Странные падения программ на Fortran

Иногда бывает так, что программы, полностью или частично написанные на Fortran, валятся в segmentation fault без видимой причины. Если посмотреть на проблему в отладчике, то будет видно очень странное: точное место, где все заканчивается — это машинная инструкция «call».

Очень часто эта проблема возникает потому, что у программы заканчивается стек: некоторые авторы программ на Fortran любят делать большие локальные массивы в подпроцедурах и не использовать для этого динамическую память. Однако, разработчики Intel Fortran уже об этом подумали и сделали ключ компилятора «-heap-arrays». Он перемещает все локальные массивы в динамическую память, значительно облегчая нагрузку на стек.

Попробуйте, может быть, этот ключ спасет вас от падений программы.

Хочется использовать больше памяти, чем есть на ядро

Бывают параллельные задачи, которым не хватает памяти, скажем, из расчета 1 GB/ядро и вычислительные процессы задачи организованы так, что каждый из них потребляет примерно одинаковое количество памяти. Понятно, что если таких процессов на рабочем узле живет столько, сколько есть ядер, то дело плохо.

Один из вариантов решения проблемы — сказать Slurm, что мы хотим выделять не одно ядро под процесс, а несколько. Для этого есть директива

```
#SBATCH --cpus-per-task N
```

где N — это как раз число ядер, выделяемых для каждого процесса.

Эффективно это приведет к тому, что каждому из процессов будет доступно не менее чем $N \cdot k$ GB оперативной памяти на машинах с k GB/ядро.