Linux in Statistics, AIMS 2018-2019 Exercises

Susan Hutchinson
Department of Statistics
University of Oxford

January 2019

What is covered

After reading this document and working through the examples you should be able to:

1. Log on to a remote system
2. Make simple use of the Linux command line including
    2.1. Navigation, absolute and relative paths
    2.2. File and directory manipulation
    2.3. View the contents of files
    2.4. Help commands
3. More advanced use of the command line
    3.1. View parts of a file
    3.2. Wildcards in filename matching
    3.3. Pipes and redirection
4. Download and copy data between systems
5. Run R jobs from a file
6. Manage jobs on a remote system
7. Checkpoint a job
8. Further Linux command line use
    8.1. File names
    8.2. Search and sort file contents
    8.3. Merge file contents from different sources
9. Connect from other locations (Eduroam, home network)

Preparation

Before you start

i. Make sure you have added the terminal window to the Dock.
ii. Open a terminal window

Speed things up

Don't forget that when using the command line in a terminal window, there are ways to make life easier for you:

Filename and command completion

- <tab> key completes commands and filenames

Arrow keys allow us to:

- recall previous commands
- change previous commands

Open a copy of this document

Browse to `http://www.stats.ox.ac.uk/pub/susan/cdt/AIMS/AIMS_Intro.pdf` and open a copy of this document. In some places it will be easier to copy and paste commands rather than type them in.

1 Log on to a remote system

1.1 Commands that will be used

Command	Purpose
ssh *user*@*hostname*.stats.ox.ac.uk	Log on to a different system from outside the department, named *hostname*.
ssh *hostname*	Log on to a different system, named *hostname* from the same Department.
cd	Change to your home directory.
curl *URL of file*	Get data using a URL. Output can be redirected to a file using >*filename*.
tar -xvfz *filename*	Extract the files from a compressed tar archive *filename*.

Table 1: Logging on to a different system and transferring data

1.2 A first connection

This section explains how to connect to the AIMS/CDT servers, located on the Statistics network.

From an Engineering desktop, you first need to log onto our gate server with your Statistics username. If you don't have this please let me know.

ssh *user*@gate.stats.ox.ac.uk

Replace ***user*** with your Statistics username as shown on your user account details form. Your username should be the your initial followed by up to seven characters from your last name.

1.3 The AIMS/CDT servers

Once you have logged onto the Statistics gate server, you can then log onto one of the CDT/AIMS servers. These are

- **greyheron**
- **greypartridge**
- **greyplover**
- **greywagtail**

I'll refer to these as the **grey*** servers from now on.

Another server, **greyostrich**, has four Nvidia Tesla K80 GPU cards and should only be used for GPU work.

1.4 Connect to an AIMS/CDT server

Now that you have logged on to **gate.stats.ox.ac.uk** you can connect to one of the **grey*** servers. In these exercises please use **greyplover**, but the instructions should apply equally to all servers.

To logon to **greyplover** use

ssh greyplover

Note that this time, only the short form of the hostname is needed, but that you do need to enter your password again.

Figure 1 may help explain what is happening.

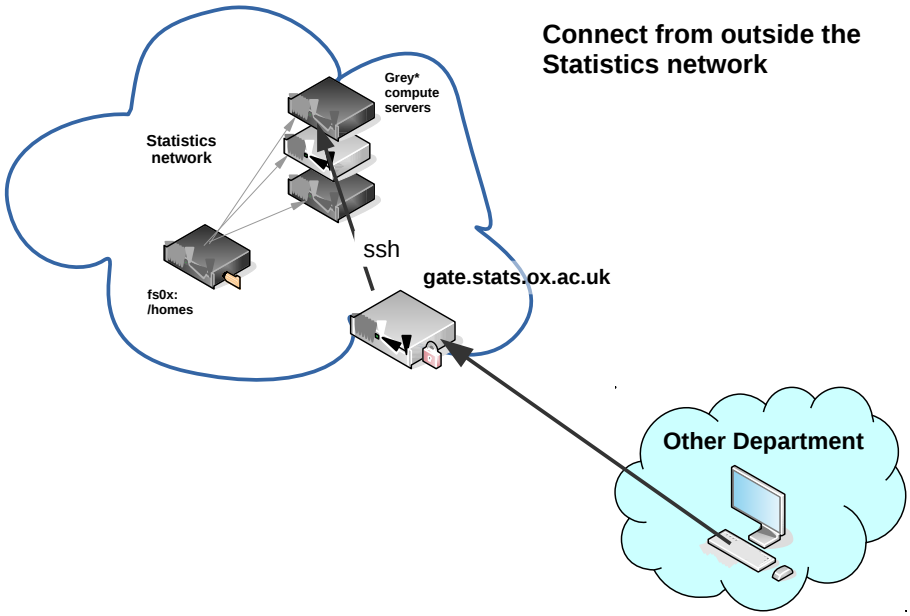


Figure 1: An example ssh command

1.5 Find your data

Once on a remote system you will have access to your files in the **/homes** directory. Data stored here is available on *all* Statistics computers.

On each **grey*** server you will also find two directories where you can store data:

/data/*hostname*/aims/aims18/*user*
/data/*hostname*/not-backed-up/aims/aims18/*user*

For example, if your username is smith, and you are logged onto **greyplover** then the directories are:

/data/greyplover/aims/aims18/smith
/data/greyplover/not-backed-up/aims/aims18/smith

Data in the first directory is a backed up daily, data in the second, *never*. There is a system-wide limit of 300GB changed data per day for backups so please, if you are moving a lot of data around, check with other members of the group to make sure they are not doing the same thing.

Later we will see how to transfer data between systems.

1.6 Download some files

Once you are logged onto **greyplover** enter these commands exactly as they appear. The commands will all be explained in more detail later.

These commands will move you to your **/homes** directory, download a single file containing files and directories needed for these exercises, then unpack the single file into multiple files and directories. Although it is possible to use a browser to download this file you can also do this from the command line as follows.

```
cd
curl http://www.stats.ox.ac.uk/pub/susan/cdt/CDT.tgz >CDT.tgz
tar -xvzf CDT.tgz
```

to unpack them.

1.7 Use graphical applications

Note that if you need to use any GUI (graphical user interface) tools, then use

ssh -X *user*@gate.stats.ox.ac.uk

A simple test is to run the command

xclock &

which should open a window displaying an analogue clock. Note the use of & which allows you to keep control of the terminal window after the GUI application has opened.

2 Introduction to the Linux command line

From now on, please use **greyplover** for all your commands unless otherwise instructed.

In each section, Linux commands are introduced and explained, then examples and exercises are provided to illustrate some commonly used features.

2.1 Where am I?

This exercise explores Linux file system navigation.

Note that all commands are typed in lower case. There are very few Linux commands which have any uppercase (CAPITAL) letters.

You will be using these commands:

Command	Purpose
pwd	Print working directory. In other words, "where am I?"
ls [options] ***directory***	List files. If used on its own, it lists everything in the current working directory (where you are currently located).
file ***filename***	Tells you what sort of file the file named *filename* (for example) is.
cd	Change directory. In other words, change my current location.
man ***command***	Command manual pages.

Table 2: Navigation and file query commands

Right away we will see how quiet Linux commands are by default. Try typing in

cd

at the prompt and you will get no output at all. This does not mean that anything has gone wrong. For many commands, no output means successful completion.

A digression on prompts. You can customise your prompt to look however you like. We won't do that now, but you may notice that it changes as you move around the file system.

Not all commands are silent. Try

pwd

You should get a response like: `/homes/user`. In all examples please replace `user` with your username. Now try

ls

You should now see a listing of all the files in the directory `/homes/user`. Let's find out about some of the contents of `/homes/user` using the **file** command.

file TestDir

The shell tells you that this isn't a regular file, it's a directory. In other words it's a special file which acts as a holder for yet more files (like a folder in Windows).

```
file longfile.txt
longfile.txt:  ISO-8859 text
```

so this file is a simple text file.

2.1.1 Absolute and relative paths

We will make use of the following character shortcuts.

Symbol(s)	Description	Purpose
`.`	One dot	The current directory.
`..`	Two dots	The directory above the current directory.
`/`	A slash or forward slash	The root or top directory.
`~`	The tilde character	The home directory.

Table 3: Symbols used by **`cd`**

Let's explore the idea of relative and absolute path names using the cd command. Change into the directory **`TestDir`**.

```
cd ~/TestDir
pwd
cd ..
pwd
cd ..
pwd
```

and so on until you can't go any further (you won't see an error, you just stop going anywhere). `..` is a special location which means up one level. All directories contain a `..` so you can go up a level. The exception is called `/` or sometimes "the root" or just "slash". You can't go any higher than / so `..` doesn't take you anywhere. Note that there is another special directory called `.` (a single dot) which means "current location".

During the above task you went up the directories one level at a time. Now let's reverse the process and go back to the Desktop directory one level at a time. You should be in "/". Note that you don't have to do the pwds but it may help you visualize what is going on. You can also use **`ls`** to have a look around each level if you have time.

```
cd homes
pwd
cd user
pwd
cd TestDir
pwd
```

Remember always to replace **`user`** by your own username.

We're now going to make use of two things, the **ls** command and the knowledge that the file called `/usr/share/icons` is a directory, to further illustrate the concepts of absolute and relative pathnames.

```
cd
cd ../../usr/share
ls icons
```

and you will get a listing of the contents of the directory.

```
ls /usr/share/icons
```

and you should get the same list of files.

The absolute (i.e. complete) location of the icons directory is **/usr/share/icons**. We have just asked to see what is kept inside it in two different ways. The first is a relative pathname while the second is the full or absolute path. Imagine the **icons** directory is a particular house, say 42, High Street, Abingdon and I ask you to deliver a letter there. I could tell you to deliver the letter to "42 High Street, Abingdon": the full/absolute address. No matter where you are in the UK, that's enough information. However, if you were already in Abingdon I could tell you to deliver the letter to the relative address of " 42, High Street" or even better, if you were standing on the high street just "number 42" would be enough. The **ls icons** command worked because you were already in the **/usr/share/** directory. It wouldn't work from another location. The command **ls /usr/share/icons** command will work from anywhere (although it's more long winded). Let's prove it by changing our current location using the cd command.

```
cd
cd TestDir
pwd
```

you should get **/homes/user/TestDir** i.e. you have moved into the TestDir directory.

```
ls /homes/user/TestDir
```

should give you a list of files in that directory. You could use **ls** on its own without the name of the directory because you have already moved there with **cd**. Let's see what happens when we make a mistake:

```
ls TestDir
```

should give you an error saying there is `No such file or directory` which is correct. The command fails because **TestDir** on its own is a relative path and you've started from the wrong place.

Try to answer/do the following:

1. Were you just using absolute or relative paths in the last example?
2. Now try to get back to the root (or `/`) directory with one command only using an absolute path.
3. Now get back to the `/homes/user/TestDir` directory using one command only.
4. What are the contents of the `/` directory? From your home directory use only one command to find out.
5. Make sure you can see your data directories **/data/greyplover/aims/aims18/user** and **/data/greyplover/not-backed-up/aims/aims18/user**.

Finally, a short cut! In order to change to a directory in your home directory, use the ~ character which represents your home directory to move to **`TestDir`** from any location.

```
cd ~/TestDir
```

2.2 File and directory manipulation

Now we're going to create a directory and put some files there.

Command	Purpose
`cd`	Change directory. In other words, change my current location.
`mkdir` ***`directoryname`***	Create a directory called ***`directoryname`***.
`touch` ***`file1 file2`***	Create one or more empty file(s) called ***`file1`***, ***`file2`*** ...
`cp` ***`file1 file2`***	Copy ***`file1`*** to ***`file2`***. Also used to copy whole directories.
`ls`	List files.
`rm` ***`file1`***	Remove (or delete) a file called ***`file1`***. Also remove whole directories.

Table 4: File and directory manipulation commands

Now execute these commands:

```
cd
mkdir directory1
cd directory1
touch file1 file2 file3 file4
```

Remember that words in italic should be replaced by names that you have chosen. Experiment to see what happens if you are not in your home directory. What happens if you try to create a directory in **`/usr/bin`**? Is there anywhere outside your home directory where you **are** allowed to create directories? [Hint: look at the top level directory – you should be able to create a files and directories in one of those. The name of the directory might also be a clue.]

Use the **`cp`** command to copy one file to another and then use **`ls`** to check that you have done what you want. Then delete a file using

```
rm file1
```

Now we are going to copy one directory to another. The commands you need are

```
cd
cp -r directory1 directory2
```

Use ls to make sure you have done what you want. The new directory should contain exactly the same files as the old one. Note use of the -r option. This makes cp copy the contents of a directory — this is known as a recursive copy. Finally remove the new directory with

```
rm -rf directory2
```

Note that this is a dangerous command and should be used with care!

Use **ls** to check that this has worked.

You should now be familiar with these simple file manipulation commands. Remember that in Linux the **rm** command really does delete files. There is no Recycle Bin to retrieve files that were deleted by mistake.

2.3 Viewing files

In these exercises we explore commands to view the contents of files.

Command	Purpose
cat *file*	Show the while contents of a file called ***file***.
more *file*	Display the contents of ***file*** a screenful at a time.
less *file*	Display the contents of ***file*** a screenful at a time, but with more options. For example, after starting **less** enter **G** to go straight to the end of a file and then move backwards.

Table 5: Viewing the contents of files

Use the following commands to look at the contents of the file `google.txt`.

```
cd
cd Files
cat google.txt
```

This is not very useful if the file is more than a screenful.

```
more google.txt
```

Note that <space> takes you to the next page and **q** will quit before the end of the file. Now try

```
less google.txt
```

See if you can get to the end of the file with a single command. Use google to help if you like. Then use **q** to exit.

2.4 Help commands

These exercises explore various ways to get help with and about commands. If you know what command you need, you can use the **man** command to find out the details of that command. Try it with a few of the commands you have used already. Not all commands have as many options as **ls**!

```
man ls
```

to find out details of the **ls** command.

i. What option is used to display modification time?
ii. What option is used to display the size of a file?
iii. How can you reverse the order of the sort so that the largest/most recently changed file is at the bottom of the list?
iv. Check that they do what you expect.

Command	Purpose
man *command*	Read the manual page for a command. So **man ls** would give details of the **ls** command and **man more** of the **more** command.
apropos *word*	Search the manual pages for names and descriptions which contain ***word***. So **apropos copy** would list all the commands that have the word **copy** in the description.
which *command*	Display the location of the command being used.
whatis *command*	Gives a brief description of a command.

Table 6: Wild cards and globbing

Sometimes you might not be sure exactly what the command is. In that case you can use the **apropos** command which finds all command descriptions which match a given word. So to find out what commands there are to manipulate files are available use

apropos file

Note that the output from this command is very long! In a future session I can explain how make this more useful.

Sometimes you need to know where Linux stores command. Use which to display the location of the file. Try it with **less**, **more**, **cp**, **apropos**:

which less
which cp
which R
which pdflatex

Did you notice that **R** and **pdflatex** are stored in different places? The **/usr/local** directory is used to share frequently used application so that we can provide a more up-to-date version than that which comes with a standard installation.

Finally you may have seen a command and want to know briefly what it does. Use the **whatis** command to find out. Try this on some commands you have already used.

This completes the section on simple command line use.

3 More advanced command line

3.1 Look at parts of a file

We have already used **cat**, **more** and **less** to look at the contents of files. But what if we just want to look at parts of a file, or find out some characteristics of a file such as the number of lines or words?

Command	Description
head *file*	Display the first 10 lines of a file.
tail *file*	Display the last 10 lines of a file.
wc *file*	Counts the number of characters, words and lines in a file.

Table 7: Displaying parts of a file

Using

```
cd
cd Advanced
cat longfile.txt
```

you see all of **longfile.txt**; using

```
less longfile.txt
```

gives you a screenful at a time. Use the spacebar to move on a screenful, G to go to the end of the file and q to exit. Now use

```
head longfile.txt
tail longfile.txt
```

to look at the first 10 lines with **head** and last 10 lines with **tail**. Use the **man** command find out how to look at the first 3 lines and last 20 lines of the file.

```
man head
man tail
```

The answers are at the end of the exercises. Now use the **wc** command to find out the number of lines, words and characters there are in **longfile.txt**.

3.2 Use wildcards to match file names

File globbing or wildcard expansion allows you to use special characters to match more than one file or directory name.

Change to the directory called **WildCards**.

```
cd ~/WildCards
```

Now experiment with wild card characters. What do the following match?

```
ls foo? ls foo2* ls foo[1-2]
```

What command would you need to match only the files **foo20** and **foo2bar**? [Hint: you might need to use more than one wild card character.] The answers are at the end. Now use

Character	What it matches
`*`	The * (asterisk or star) matches any number of characters or none.
`?`	The ? (question mark) matches exactly one occurrence of any character.
`[ ]`	Matches any characters in a given range.

Table 8: Wildcard characters

```
wc -l *
```

to find the length of each file. Note that the -l is a hyphen (`-`) followed by the lowercase letter `l`. You should see output like this:

```
  1 foo
  2 foo1
  2 foo10
  1 foo2
  1 foo20
  3 foo2bar
 10 total
```

Now see if you can create a match so that `wc -l` just shows the files with a 1 (the number one) in their name. Again a possible answer is at the end.

3.3 Pipes and redirection

In this exercise we are going to explore two very powerful command-line features which increase the flexibility and range enormously.

Command	Description
`du -sk`	Display the sizes in Kilobytes of all the files in a directory.
`grep`	Search for the characters with a given pattern in file.
`sort`	Sorts the contents of one or more files.
`tail`	View the last few lines of a file.
`wc`	Counts the number of characters, words and lines in a file.

Table 9: Commands used in our pipe

tand we will be using the following characters.

Almost all Unix/Linux commands use standard input for receiving instructions and standard output for displaying the results. Commands make use of special data streams to move input and output to and from the command. STDIN can be though of as a gateway into the code, STDOUT

Command	Purpose
>	Sends the output from a command to the named file. If the file already exists the previous contents will be lost. If the file doesn't exist it will be created.
»	Appends the output from a command to the named file. If the file doesn't exist it will be created.
<	Reads input from the named file. NB This option is rarely used.
|	Uses the output from one command as the input to the next.

Table 10: Characters used for pipes and redirection

is a gateway for output. Note that there is a third standard data stream called STDERR (standard error) which commands often use to print error messages and warnings. We won't mention STDERR again today. By default STDOUT gets routed to the screen display. It is also easy to connect a commands STDIN to the keyboard device for example. The STDIN allows a program to ask you questions and you can type responses.

Redirection is a way of "grabbing" STDOUT or STDIN and forcing it to go somewhere other than the default. The most common instance is to redirect STDOUT into a new file. This is extremely useful because it means you can run a command and save the results automatically. To redirect STDOUT use the $>$ symbol followed by the target so command > file.txt redirects the output from command into a new file called file.txt. To redirect STDIN into a command use $<$ like this: command $<$ file.txt.

Connecting commands together with pipes is one of the most powerful features of Linux. Linux does not have a command to count the number of files in a directory but it does have one command to list the files (ls) and a second command (wc) to count the number of lines in a list.

You could therefore use a pipe (|) to glue ls and wc together:

```
ls | wc -l
```

Note that we are using the -l (that is a hyphen followed by the lower case letter l) argument or option for wc. To get information about any command use

man command eg man wc

3.3.1 Examples using pipes and redirection

In the directory **Searching** there are three small text files. Check their contents using **cat**. If we wanted to search for all occurrences of "green" in the files **fruit** and **veg** we would use

```
grep green fruit veg
```

The output is displayed on the screen. To store the output from the command in a file use

```
grep green fruit veg > out.txt
```

which would store the output of the **grep** command in a file called **out.txt**. Check this with

```
cat out.txt
```

If you run the command again, this time using

```
grep green fruit veg » out.txt
```

you should see that there are now two copies of the output in the file. It is also possible to use **<** redirect the input from the keyboard to a file. For example

```
cat <out.txt
```

also works.

In this short example we can count how many times green appears, but when there are many matches it would be useful to use **wc** to find out. We can redirect the output from the command into a file and then check the length. Note that in all cases the **-l** option to **wc** is a minus sign followed by the lower case letter **l**.

```
grep green fruit veg >out.txt
```

and then run **wc** on the file

```
wc -l out.txt
```

but it would be much more efficient to join the two commands together with a pipe. Use

```
grep green fruit veg | wc -l
```

Here the two commands **grep green fruit veg** and **wc -l** are joined together by a special symbol called a pipe.

Now we're going build a longer command which will find the 5 largest files in a directory. When building pipes of commands it often helps if you make sure each link in the pipe works before adding the next. First display the size of all the files in the **/usr/bin** directory.

```
du -sk /usr/bin/*
```

Now sort this output by size, so that the largest are first.

```
du -sk /usr/bin/* | sort -nr
```

Now display the final 5 lines which will be the 5 largest files.

```
du -sk /usr/bin/* | sort -nr | head -5
```

Now how you would you find the 5 largest files in the **/usr/bin** directory beginning with the letter 's'?

4 Downloading and copying files to remote systems

As well as logging on to remote systems, it is often necessary to copy files and directories from one system to another. These commands will be used.

Command	**Purpose**
ssh-keygen	Generate an ssh public/private key pair to enable moving between computers without entering a password.
curl ***URL***	Transfer files and directories from the web address (URL) provided.
scp *file host*:*location*	Copy a single file, *file*, to a remote system, *host*.
scp *file user*@*host*.**stats.ox.ac.uk**:*location*/*file* .	From outside the department, copy a single file *file*, from location *location*, to the the current directory on your local computer, using the same file name.
scp -r *directory user*@*host*.**stats.ox.ac.uk**:*location*	From outside the department, copy a directory, *directory*, to location, *location*, on a remote system, *host*.

Table 11: The **scp** command

4.1 Remove the password prompt between Statistics systems

Set up ssh keys so that you are not prompted for a password each time you move between Statistics systems. On **gate** do the following:

i. Enter the command

ssh-keygen -t rsa

Generate a public key/private key pair. The following output should appear

```
Generating public/private rsa key pair.
Enter file in which to save the key (/homes/user/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /homes/user/.ssh/id_rsa.
Your public key has been saved in /homes/user/.ssh/id_rsa.pub.
The key fingerprint is:
20:30:e6:b0:5f:a0:aa:4f:d0:09:c7:b0:f4:25:45:4e
                                user@gate.stats.ox.ac.uk
The key's randomart image is:
+--[ RSA 2048]----+
|o.=.oE           |
```

```
|.X.+=            |
|= +.o..          |
|.= o . .         |
|o +     S        |
|..               |
|. .              |
| o               |
|  .              |
+-----------------+
```

ii. **cd**

iii. **cd .ssh**

Note the dot before the directory name. Now check whether the file **authorized_keys** exists. If it does use this command:

cat id_rsa.pub »authorized_keys

if the file doesn't exist use

cat id_rsa.pub >authorized_keys

Now check that this works:

ssh greyplover

The first time you connect you will see

```
The authenticity of host 'greyplover (163.1.210.96)' can't be
established.
ECDSA key fingerprint is
3a:b1:d2:0d:a3:09:cf:46:e9:43:04:87:ac:f3:8e:10.
Are you sure you want to continue connecting (yes/no)?
```

Enter **yes**.

Now log onto **greyplover** via **gate.stats.ox.ac.uk**. Check with **ssh** to log onto **greypartridge**.

4.2 Download data from the web to a grey* server

This is the simplest method for downloading data sets that are available on the web.

Use the **curl** command with the URL (web address). For example, to download the dataset 'Professional socialisation...' from the UK Data Service, to your home directory, use

```
cd
curl http://ws.ukdataservice.ac.uk/REST/Download/Download/\
DSO/1479tab_f3b6bad2bdb23b5924e346085ab27f69.zip > data.zip
```

[The **curl** command needs to be entered all on one line.]

4.3 Copy a single file from one grey* server to another

To copy **mandel.R** in the directory **R** to the a directory, **/data/greypartridge/aims/aims18/user** on **greypartridge** do the following:

- Make sure you are in the directory where the file you want to copy is stored.
- Make sure you know the location on the **greypartridge** server where you will copy to file to.

```
cd ~/R
scp mandel.R greypartridge:/data/greypartridge/aims/aims18/user/.
```

Note the final dot "." in the location. This means that the file will have the same name as the version that is being copied. If you want to give a different name use

```
scp mandel.R greypartridge:/data/greypartridge/aims/aims18/user/my_mandel.R
```

for example.

4.4 Copy a directory from one grey* server to another

Copy the directory **R** to a directory of the same name on **greyplover**.

- Make sure you are in the directory above the directory you want to copy is stored.
- Make sure you know the location on the **greypartridge** where you will copy to directory to.

```
scp -r ~/R greypartridge:/data/greypartridge/aims/aims18/user/.
```

4.5 Copy a single file from an Engineering Lab system to a grey* server

- Make sure you are in the directory where the file you want to copy is stored.
- Make sure you know the location on the **grey*** server where you will copy to file to.

scp *filename user*@*hostname*.**stats.ox.ac.uk**:*location*

for example to copy the file **mandel.R**, keeping the same name, from your home directory to the directory **/data/greyplover/aims/aims18/user/** on **greyplover** as the user **user** use

```
scp mandel.R user@greyplover.stats.ox.ac.uk:/data/greyplover/aims/aims18/user/.
```

4.6 Copy a file or directory from other systems to a grey* server

The simplest approach is to first connect the VPN. See Section 9 on page 26 for more information about the Statistics VPN. If this is possible then once connected, the instructions in the previous sections will work.

If this is not possible then there are various options.

If the file or directory is not large, then it is simpler to copy to your home directory on **gate.stats.ox.ac.uk** and then to a data directory on a **grey*** server as described above.

However, if there is a lot of data or the file is very large use the following. In this example **longfile.txt** will be copied from a system with a Linux/Unix command line to the local data area on greypartridge.

```
cat longfile.txt | ssh user@gate.stats.ox.ac.uk 'ssh greyplover\
"cat > /data/greyplover/aims/aims18/user/ENG_longfile.txt"'
```

[Note that the command needs to be typed all on one line.]

In the next example the contents of the directory **TestDir** will be transferred to the directory **ENGTestDir** which must be created in **/data/greyplover/user/aims/aims18/**.

```
cd ~/TestDir

tar -zcBpf - .  | ssh user@gate.stats.ox.ac.uk 'ssh greyplover\
"cd /data/greyplover/aims/aims18/user/ENGTestDir && tar -zxBpf - ."'
```

[Again, the command needs to be typed all on one line.]

5 Run R jobs from a file

Command	Purpose
`R CMD BATCH` *file*	Run an R job from a file, *file*
`tail -f` *file*	Read a file, *file*, from the end, watching as data is appended.

Table 12: The R BATCH command

If an R job lasts more than a few minutes or will be running on a remote system, use the **`R Batch`** command.

Here is some code – with thanks to Tom Jin – to compute points from the Mandelbrot set. You will find in the **`R`** directory, in a file **`mandel.R`**.

```
#! /usr/bin/env Rscript

source("mandelbrot.R")

gridSize <- 4000

# Compute fractal
y <- iterate.until.escape(immin = -1.5, immax = 1.5, remin = -2,
             remax = 1, delta = 3/gridSize,
             trans = function(z,c)z^2+c,
             cond = function(z)abs(z) <= 2, max = 127)

# Output the fractal.
if(interactive()) {
  image(y, col = topo.colors(128), useRaster = TRUE)
} else {
  library(png)
  writePNG(y, target = "mandelbrot.png")
}
```

Note the test for an interactive session **`+if(interactive()) {`**. If R is running from the terminal, the plot is displayed on the screen, if run from a script, then the plot is saved to a file, `mandelbrot.png`.

An additional file is needed (**`source("mandelbrot.R")`**. This should also be in the **`R`** directory. It should be stored in the same directory as your script. Assuming the following

- the R commands are saved in a file, **`mandel.R`**,
- it is stored in the current directory,
- the data file **`mandelbrot.R`** is in the same directory

then the command

`R CMD BATCH mandel.R &`

would run the command. Remember to run the job "in the background" by appending an **`&`** to the command so that you keep control of the command line.

The output that would usually appear on the screen will be sent to a file, **`mandel.Rout`** by default. The contents of the **`mandel.Rout`** will look something like this:

```
R version 3.1.2 (2014-10-31) -- "Pumpkin Helmet"
Copyright (C) 2014 The R Foundation for Statistical Computing
Platform: x86_64-unknown-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> #! /usr/bin/env Rscript
> source("mandelbrot.R")
> gridSize <- 4000
> # Compute fractal
> y <- iterate.until.escape(immin = -1.5, immax = 1.5, remin = -2,
+ remax = 1, delta = 3/gridSize,
+ trans = function(z,c)z^2+c,
+ cond = function(z)abs(z) <= 2, max = 127)
> # Output the fractal.
> if(interactive()) {
+ image(y, col = topo.colors(128), useRaster = TRUE)
+ } else {
+ library(png)
+ writePNG(y, target = "mandelbrot.png")
+ }

>
> proc.time()
   user  system elapsed
 65.772   5.495  71.942
```

5.1 Monitoring jobs

It is possible to watch the output from the job as it is written to the output file. For this job use

`tail -f mandel.Rout`

This is a particularly useful command. For more information about **`tail`** use **`man tail`**.

6 Managing jobs on remote systems

Command	Purpose
screen	Connect and disconnect from a session from multiple locations and allow long-running processes to persist without an active shell session.

Table 13: The **screen** command

Once you have the **R** script and any associated files on the server you are ready to submit the job.

On the remote system you should use the **screen** command. This allows you to submit **R** (and other) jobs, then disconnect from your session. Your desktop computer can then be switched off or rebooted, without interrupting or stopping the **R** job on the remote system. To check the process of the your job you simply **ssh** again to the same server, and start the **screen** command again.

An example session would look like this.

```
ssh greyplover
screen
R CMD BATCH mandel.R &
```

Don't forget run the job in the background. If you want to check that the job is running use

```
tail -f mandel.Rout
```

Once you are happy the job is running use the sequence

CTRL-a d

to detach from the screen process. You should see a message like:

```
screen
[detached from 6422.pts-0.greyplover]
```

You can then logout. To reattach the screen session log back into the server and use

```
screen -r
```

If you have multiple **screen** sessions on a server, then the command

```
screen -list
```

will display all your screen sessions. For example:

```
 screen -list
There are screens on:
        7375.pts-0.greyplover        (Detached)
        6422.pts-0.greyplover        (Detached)
2 Sockets in /var/run/screen/S-jones.
```

To attach a particular session use

```
screen -r 7375.pts-0.greyplover
```

Once you have finished with a screen session reattach the session and type in

```
exit
```

You can use **screen -list** to check that it has closed. As ever, use **man screen** for full details.

There is a longer **screen** tutorial here: http://www.rackaid.com/blog/linux-screen-tutorial-and-how-to/.

There is an alternative to the screen command, **tmux** which is installed on all **grey*** servers.

6.1 Running commands in the background

When you start a command from a terminal window that takes a long time, or opens a new window it is good practice to add an & (ampersand) character after the command to keep access to the command line.

7 Using checkpointing to keep jobs running

Command	Purpose
`dmtcp_launch`	Checkpoint your script

Table 14: The `dmtcp` command

To further protect your jobs against both unexpected events such as power failures, or scheduled reboots use

```
dmtcp_launch R CMD BATCH mandel.R &
```

This means that in the event of a reboot, your job will start at the point at which is was interrupted.

8 Further Linux command line

The following are some more advanced Linux command line features for you to try for yourself.

8.1 File and directory names

We're going to look at some of the problems you can encounter with files and directories. You should have a directory called **Files**. Now do this:

```
cd
cd TestDir
```

First look in the **Cases directory**.

```
cd Cases
```

Make sure that you can read all three files there. Now look in the **OpenThis** directory.

```
cd ..
cd OpenThis
```

i. Now see if you can read the files **star**, **astar** and ***star**. What happens when you try to read the file called ***star**? The * character has caused confusion because it is a wildcard - it matches any character. We will be looking at wildcards and pattern matching very soon. See if you can read this file without reading the other ones. Using google is not cheating!
ii. Now change into the directory **Open This**. Using double quotes or backslashes may help.
iii. Now try to read the file called **-ReadMe**. Unfortunately using quotations round the filename or the escape character (\) won't work this time. In each case the command (one of **cat**, **more** or **less**) is interpreting the leading **-** (hyphen) as an option. Again, don't be afraid to google to see if you can find the answer to this. Try deleting the file called **-DeleteMe**

8.2 Searching and sorting

In this exercise we will use commands to search for patterns within files and sort the contents of files.

Command	Description
grep *pattern file*	Search for the characters in pattern within file.
sort *file*	Sorts the contents of one or more files.

Table 15: Some searching and sorting commands

Now change into the Searching directory.

```
cd
cd Searching
```

I suggest you use the **cat** command to check the contents of the two files, **fruit** and **veg**, so that you can see what they contain. Now use

```
grep melon fruit
```

Can you see what has happened? Only the matching lines in the file are shown. Now try

```
grep green fruit veg
```

What do you think has happened here? Did you notice that the file names were included in the output. This is because more than one file (both **fruit** and **veg**) has been searched and so the output tells us where the matches were found. Using

```
man grep
```

see if you can find the option that makes **grep** ignore case so that both Melon and melon would be found.

Finally use the **sort** command to order the files. Note that you can sort more than one file which merges the output of all the files.

```
sort fruit veg
```

What is the option that reverses the order - so that the fruit and vegetables are sorted in reverse alphabetical order?

8.3 Merging information from different files

Command	Description
awk	A pattern scanning and text matching program.
paste	Merge lines of a file.

Table 16: Commands used in our pipe

This is a rather contrived example but demonstrates some very useful Unix/Linux commands. Make sure you are still in the **Searching** directory.

```
cd
cd Searching
```

Now we are going to use the **awk** command to print out only the first column of the file called creatures. Have a look at the file before you run this command.

```
awk '{print $1}' creatures
```

What do you think would appear if you replaced $1 by $2? Make sure you copy the command exactly. In particular you need the single quote character and curly brackets. Now change this to

```
awk '{print "A", $1, "eats"}' creatures
```

As well as a list of animals you should also see that the words "A" and "eats" appears either side of each animal. Now we are going to use the paste command to include the output from the fruit file.

```
awk '{print "A", $1, "eats"}' creatures | paste -d" " - fruit
```

Again, make sure that you have entered the commands exactly as they appear here. To explain what the **paste** command is doing: the -d" " option makes a space rather than a tab appear before the name of each fruit, the - before fruit means that the output from **awk...** is included.

Now see if you can make the fish eat the vegetables! See the Answers section below for an answer.

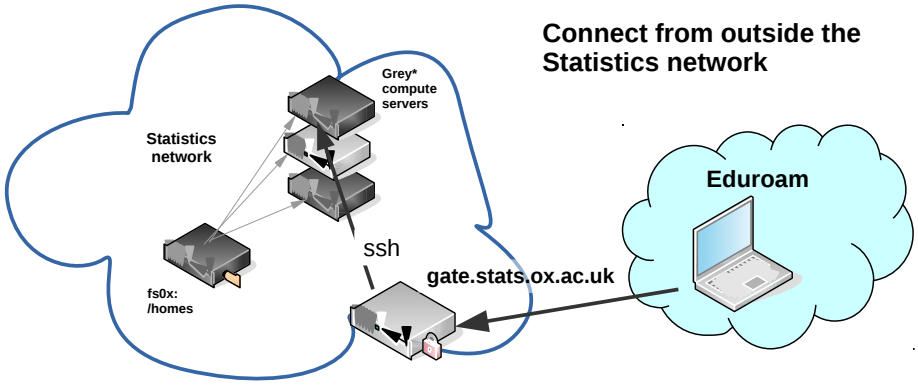
9 Connecting from other locations

These sections explain how to connect from other locations.

9.1 Connecting from Eduroam

If you are using an Eduroam connection, you will either need to connect to the **grey*** servers via ***user*@gate.stats.ox.ac.uk**, or connect to our VPN and then connect directly to one of the grey* servers.

Figure 2 shows how to connect to a **grey*** server when on a device connected to Eduroam. For



Example command from Eduroam to gate: **ssh *user*@gate.stats.ox.ac.uk**
Example command from gate to a grey* server: **ssh greyplover**

Figure 2: An example ssh command from an Eduroam-connected device

example, to connect to **greyplover** use

***user*@gate.stats.ox.ac.uk**

then from **gate**

ssh greyplover

9.2 Connecting from a home computer

Similarly, when connecting from a home computer you will need to use

***user*@gate.stats.ox.ac.uk**

then from **gate**

ssh greyplover

as shown in Figure 3.

9.3 Connections from the Statistics VPN

Statistics provide a VPN service. How to connect is described here:`http://www.stats.ox.ac.uk/about_us/it_information/generalaccess/using_the_vpn_service`.

Once you are connected you need to use

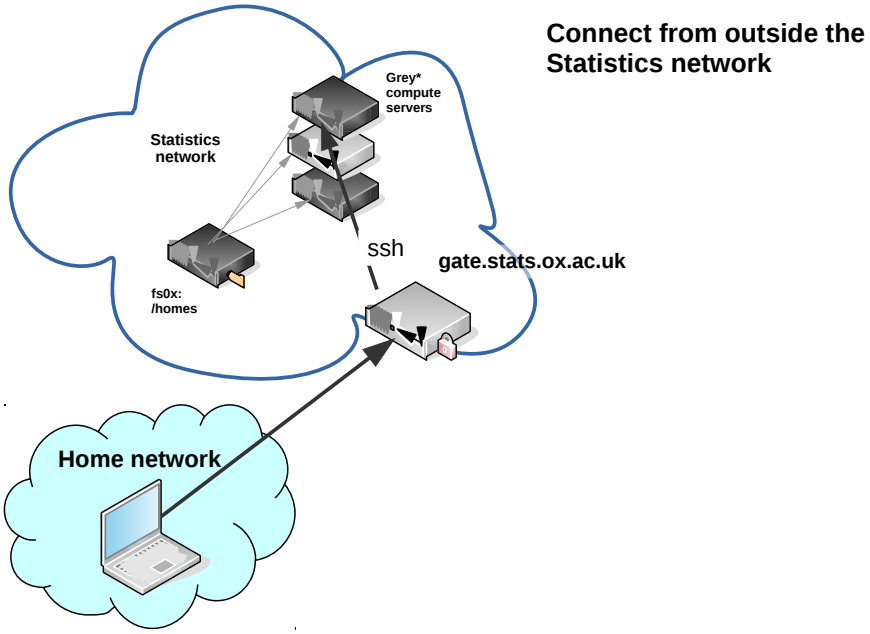
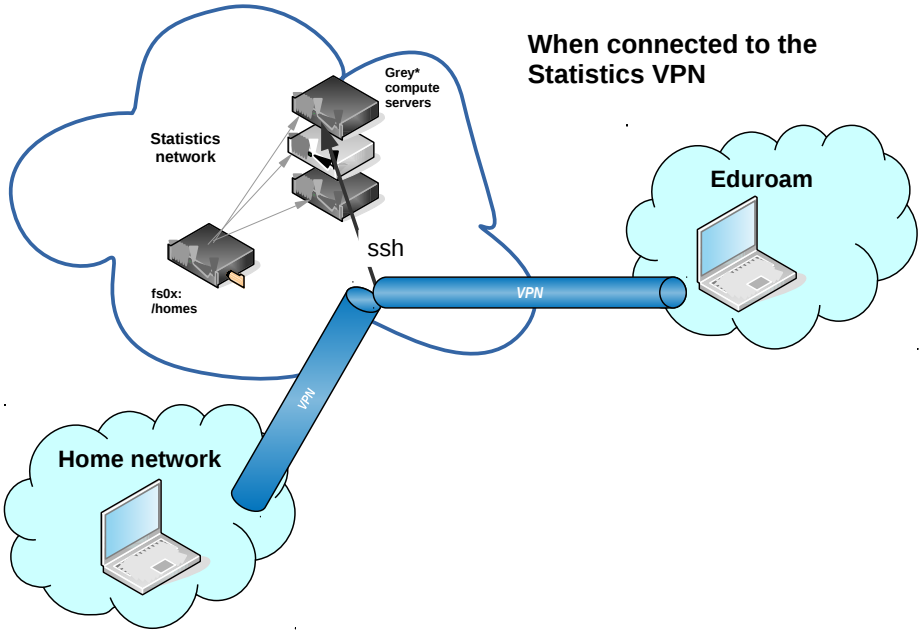


Figure 3: An example ssh command from home computer

***user*@greyplover.stats.ox.ac.uk**

to connect to a server. It is no longer necessary to connect to **gate.stats.ox.ac.uk**.

Figure 4 gives an example of connecting to a **grey*** server when using the VPN.



Example command to connect from Eduroam/home computer to a grey* server:
ssh *user*@greyplover.stats.ox.ac.uk

Figure 4: Example ssh command from a home computer or Eduroam when using the VPN

10 Conclusion

That concludes the Introduction to Linux exercises. The Introduction to Linux on the Academic Research Computing site here: `http://www.arc.ox.ac.uk/content/introduction-linux` covers much of what we have done today, but also contains some useful links. Note that it will refer to systems that you cannot use, but the commands described should work.

11 Answers

2.1.1

1. A relative path was used.
2. `cd /`
3. `cd ~/TestDir`
4. `ls /`

2.3 In `less`, use `G` to go straight to the end of a file.

2.5 `wc longfile.txt`

2.6 `ls foo2?*` and `wc -l *1*`

2.8 `du -sk /usr/bin/s* | sort -nr | head -5`

7.3 `awk '{print "A", $2, "eats"}' creatures | paste -d" " - veg`