

Chapter 1: Unix skills for sequence analysis

Yanbin Yin and Mitrick A. Johns

1.1 History and comparison with Windows

Operating systems (OS) are the software system that manages the computer hardware. If the human body is the computer then the brain is the OS that sends orders to different organs. You can imagine that if the brain did not function, you could not move a single finger even a little bit. The best known computer OSs include Unix, Mac OS, and Windows. **Unix** was developed in 1969 by Ken Thompson and Dennis Ritchie at Bell Labs, as a way to implement multi-user and multi-tasking computer systems. Unix proved to be very useful as well as easy to port to many different kinds of computers. Because Unix was not patented, many variants have appeared over the years. **Linux**, a free, open-source version of Unix created by Linus Torvalds, is probably the most widely used form of Unix in academia. The GNU Foundation has developed many useful system tools; because of this, Linux is often referred to a “GNU-Linux”. **Mac OS** is the third well-known OS, essentially also rooted in Unix. While Windows and Mac OS dominate the personal computer market, Unix is the primary choice for servers and research-oriented computing.

Nowadays many Unix distributions support a graphical user interface. However, as biologists who want to learn bioinformatics, we need to use Unix’s powerful command line utility (or simply called **Shell**).

Think about using Windows:

1. Can you open a FASTA format DNA sequence file (or simply any text file) as large as 3 Giga bases, say the entire human genome? The answer is very likely no, if using Windows wordpad or notepad. Even if you can, scrolling up and down the window would be extremely slow; or
2. Supposing the DNA sequences have many letter Ns, can you change all Ns in the DNA sequences to Xs? Very likely no; or
3. Can you simply count how many Ns in the file? No way!

However, using the Unix Shell, you can do all of these easily using just one or two commands connected using Unix’s pipe (you will know what it is after you finish this chapter). Even using the same computer, having a Unix OS will enable you to do a lot of things that you cannot do with the Windows OS.

1.2 Installing Linux

Suppose you are a biologist without any training in using command line interface, the simplest

thing you can do is to find a desktop/laptop that has Linux OS already installed. If such a machine is not readily available, you can easily install **Ubuntu** (the most popular and free Linux distribution) on your laptop/desktop with a Windows OS (you CAN have two OSs installed). Here are the basic steps to install Ubuntu:

1. Go to <http://www.ubuntu.com/download/desktop> to download Ubuntu desktop version (the file will be like *ubuntu-12.04.1-desktop-i386.iso* or *ubuntu-12.04.1-desktop-amd64.iso*, depending on if your processor is 64-bit or 32-bit: on a Windows machine, mouse on “my computer” in the desktop and right click to find out).
2. Burn a CD using the downloaded ISO file.
3. Use the CD to install Ubuntu on your desktops/laptops (the same machine you burned the CD or a different one) following the step-by-step instruction on the screen. It is very similar to installing Windows and you will be asked to supply a user name and password.

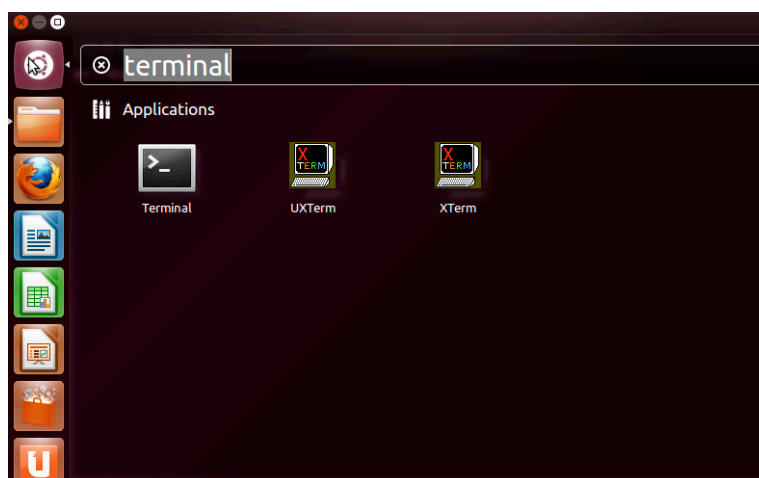
Alternatively, many academic institutions have Unix servers available for use by community members. More and more institutions have high performance computing clusters that have hundreds or even thousands of nodes, with each node containing many CPUs. These machines are designed to provide high throughput scientific computing service to researchers on campus. Most high performance servers use Unix, and as a faculty or student you can usually request an account for free access to their computing facility.

1.3 Unix command line terminal

In the graphical interface of Windows, Linux and MAC, you can click your mouse to make something happen. However, under **command line terminal** (or console) interface, you have to type in a command using the keyboard and hit *Enter* to let something happen.

Suppose you are sitting in front of a desktop with Ubuntu installed, to get to the command line interface from the Ubuntu graphical interface, search for the terminal application and open it (**Figure 1.1**):

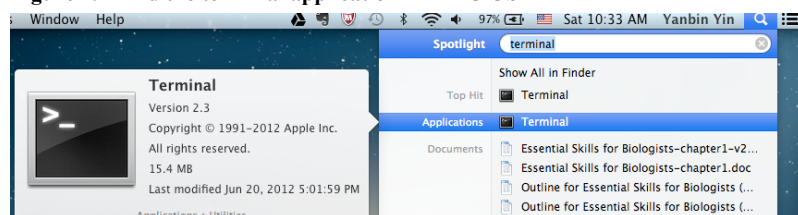
Figure 1.1 Find the terminal application in Ubuntu



Then you can work on the terminal directly on the machine with the Ubuntu Linux OS installed. If you want to use this Linux machine as a server so that you can access it from other computers, you will need to install an **SSH** server client on this Linux machine first (beyond the scope of this book chapter).

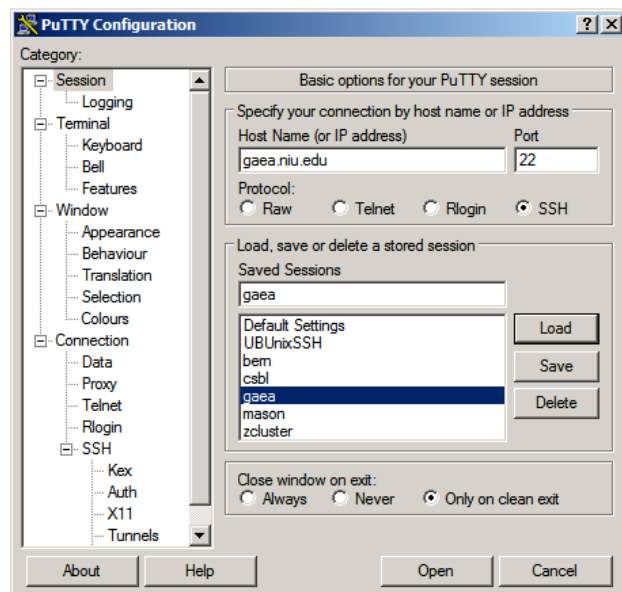
Alternatively if you are using a MAC desktop/laptop, you can bring up the MAC terminal easily by searching for “terminal” in the spotlight search box (upper right corner of your desktop, **Figure 1.2**).

Figure 1.2 Find the terminal application in MAC OS



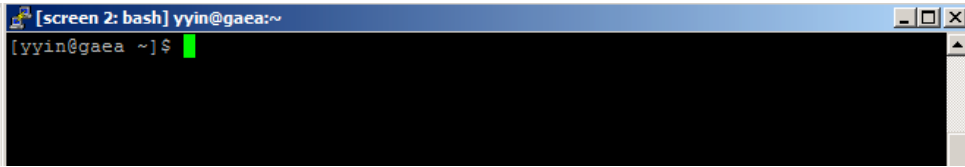
Lastly but most commonly you have a Windows desktop/laptop; if you want to access a **remote** Unix server from your Windows machine, you need to install an SSH terminal client, e.g. **PuTTY** (<http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>). Once you have the SSH client installed in your Windows machine and running, enter the IP address or host name of the remote server to log in. For example, we log into the *gaea.niu.edu* server using PuTTY in **Figure 1.3**:

Figure 1.3 Access remote Linux server using PuTTY in Windows



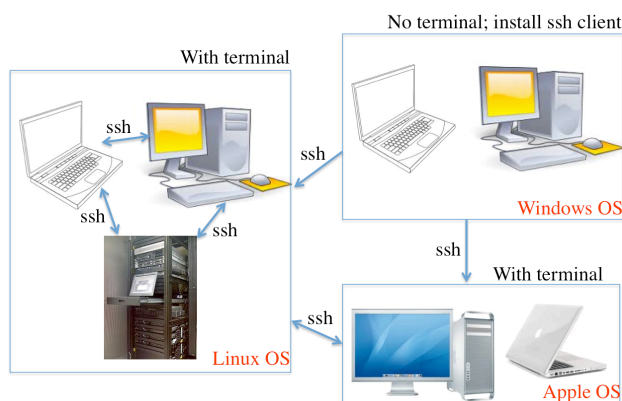
You will be asked for a user name and then a password. These should have been given to you after your system administrator opened the account for you. If this is your first session with the server, you will be asked if you want to accept the encryption keys. These keys are necessary for secure communications, so type yes and proceed. If you logged in successfully, you will be in a terminal session like **Figure 1.4**:

Figure 1.4 The prompt after you log in a remote Linux server using PuTTY



To summarize, there are multiple ways to access Unix terminal environment (**Figure 1.5**). You can either install Linux OS in your machines (desktop/laptop), or you install an SSH client on a Windows machine to access other Linux machines (desktop/laptop/cluster). Since Mac OS is also based on Unix, you can access the Mac terminal using the SSH client too. If you are already on a Linux machine, you can open a terminal and use the “*ssh*” command to easily connect to other Linux machines, e.g. from your laptop Linux to remote Linux cluster.

Figure 1.5 Different ways to connect to Unix terminal environment



All the following sections will be introduced supposing you are using PuTTY connecting to a remote Linux server from a Windows machine, which we believe is the most common scenario.

1.4 How the command line interface works

Figure 1.4 shows the command prompt: `yyin@gaea ~]$`, with the cursor sitting immediately after the prompt. Command prompt symbols are different on different machines, but they often end with a “`>`” or a “`$`”. To make something happen in a command line interface, you type your command at the prompt, and then press the Enter key. The computer then executes the command. During execution, output text is printed on the screen. When execution is completed, the computer produces a new command prompt for you. If the cursor is not visible, the computer is busy executing your command and you need to wait for it to finish before issuing a new command. If an error has occurred, the computer will tell you about it. Error messages are often cryptic at first; however, online searches coupled with experience will usually make their meaning clear.

A critical contrast between Unix and other systems: there is no UNDO command in Unix! If you execute a command, there is no taking it back. It pays to look at what you have typed on the command line before you hit *Enter*. And, it pays to keep backup files and modify only copies of your data, not the data themselves.

Most Unix commands are short set of letters, such as “*ls*” or “*cp*”. These are often followed by

option flags, which start with - or --, such as *-l* or *--archive*. Some commands are then followed by one or more parameters, which DO NOT start with -. Some commands have many options and parameters: you can learn about the syntax of a command by entering “*man*” (manual) followed by the command. Thus, “*man cp*” brings up the manual pages for the *cp* command.

A small trick: in case you forget to spell a command, at any time when you type the first few letters of a command, hit the *tab* key to automatically complete the command; or hit the *tab* key twice to get a list of commands. You can also hit the *tab* key to complete a file name or folder name if they exist. If pressing *tab* key does not give you anything, it means the command or folder/file name does not exist.

A trick to move the cursor in the command line if using PuTTY: if you misspelled a command or a file name, press *Alt+BkSp* to delete a whole word before the cursor, *Ctrl+/_* to redo. Press *Alt+b* to move cursor back a word, *Alt+f* to move forward, *Ctrl+a* to go to the beginning and *Ctrl+e* to go to the end. These allow you quickly move cursor in the command line if you have already typed in a long line of words (very common for experienced users). If you use MAC or other SSH client, these tricks may not work.

The following sections intend to walk you through the most useful Unix/Linux commands and command combinations (using *pipe*, which will be explained later) that you will need to learn for doing biological sequence analysis. We assume that you have already had the basic biological knowledge to understand what DNA and protein sequences are. Many of the examples use commands without explaining every detail of the command syntax. When you need to understand the details of a command, you can always run “*man command*” to see the manual page for that command or do a google search.

1.5 Let's get some real sequence data

We now use the *wget* command to download the yeast protein database from NCBI (National Center for Biotechnology Information: www.ncbi.nlm.nih.gov/):

pwd: find out where you are in the Linux system; normally after you logged in a terminal, you are at your home directory, denoted as ~

```
[yyin@gaea ~]$ pwd
/home/yyin
```

mkdir: make a directory called book under /home/yyin

```
[yyin@gaea ~]$ mkdir book
```

cd: change directory to book; note in the command prompt bracket, the ~ has become book, showing the working directory is changed

```
[yyin@gaea ~]$ cd book
[yyin@gaea book]
```

wget: get the yeast.aa.gz file from the web

```
[yyin@gaea book]$ wget -q ftp.ncbi.nih.gov/blast/db/FASTA/yeast.aa.gz
```

ls: list the files

```
[yyin@gaea book]$ ls
yeast.aa.gz
```

gzip: uncompress the data

```
[yyin@gaea book]$ gzip -d yeast.aa.gz
[yyin@gaea book]$ ls
yeast.aa
```

Here we used six different commands: *pwd*, *ls*, *mkdir*, *cd*, *wget* and *gzip*. Among them,

- *pwd* does not need followed by a file or directory name (called command argument in Computer Science)
- *ls* and *cd* do not either but a lot of times you want to have a file or directory name (*cd* cannot be followed by a file name)
- *mkdir* must be followed by a directory name
- *wget* and *gzip* are also followed by a command option or flag (*-q* and *-d* in the examples)

We will explain in more details later.

1.6 The *man* command and Linux file system

Most Unix commands have more than one option. These options help enrich the functionality of the commands they are following and sometimes are very powerful. However, options are difficult to memorize. But at any time you can use **man** followed by a command name to get the manual of the command, which list you all the options. For example:

man: show manual of the *ls* command

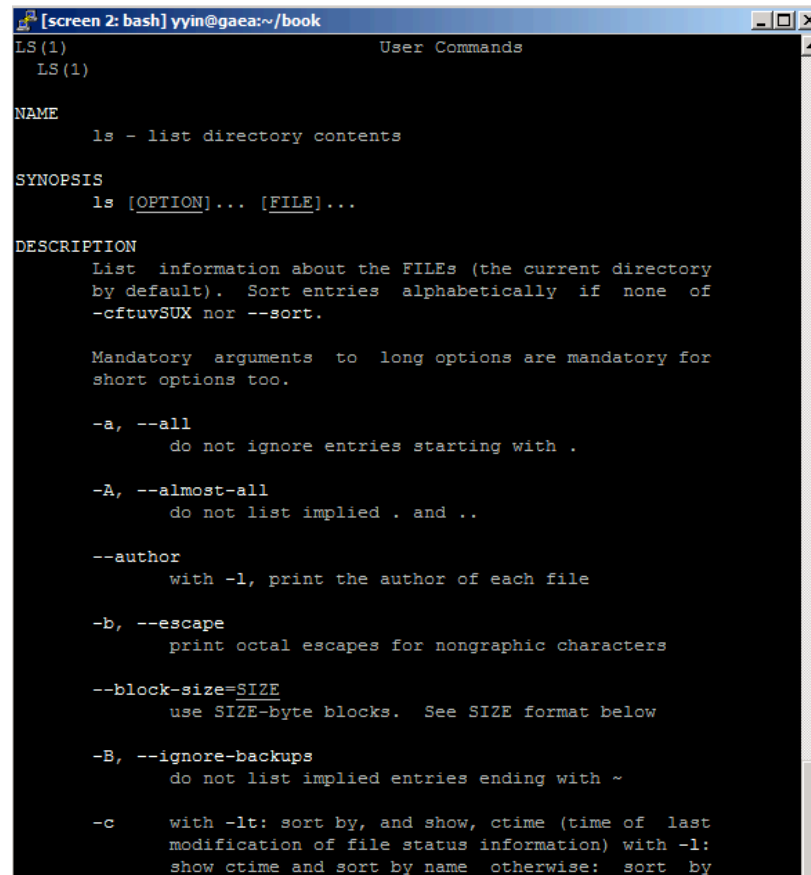
```
[yyin@gaea book]$ man ls
```

You will get a lot of information (Figure 1.6), often more than one page; you can use *space* to page down or *u* to page up (or simply use the *PgUP* and *PgDn* keys on the keyboard). Type *q* to exit from the man page.

Manual pages (or “*man* pages”) are written in a formal style that can be intimidating. The syntax of the command, what needs to be on the command line to run it successfully, is given in the *SYNOPSIS*. The command itself is followed by a set of options or flags. The individual options are discussed in the *DESCRIPTION* section. Following the options are arguments, things like the name of the file or directory that the command is aimed at. Options and arguments that are not necessary are surrounded in square brackets like *[]*. When an option or argument can be used more than one time, it will be followed by “...”. If you have trouble understanding how to run the command, you will often find more information about it using online search engines. And, don’t

be afraid to try things out! Nothing you can do will wreck the computer.

Figure 1.6 Manual page of command *ls*



```
LS(1)                                User Commands
LS(1)

NAME
    ls - list directory contents

SYNOPSIS
    ls [OPTION]... [FILE]...

DESCRIPTION
    List information about the FILEs (the current directory
    by default). Sort entries alphabetically if none of
    -cftuvSUX nor --sort.

    Mandatory arguments to long options are mandatory for
    short options too.

    -a, --all
        do not ignore entries starting with .

    -A, --almost-all
        do not list implied . and ..

    --author
        with -l, print the author of each file

    -b, --escape
        print octal escapes for nongraphic characters

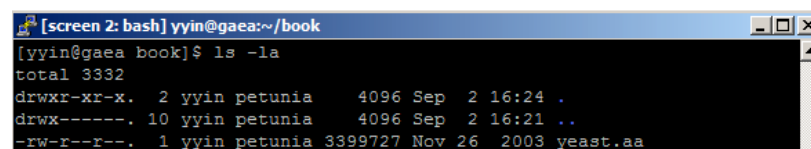
    --block-size=SIZE
        use SIZE-byte blocks. See SIZE format below

    -B, --ignore-backups
        do not list implied entries ending with ~

    -c
        with -lt: sort by, and show, ctime (time of last
        modification of file status information) with -l:
        show ctime and sort by name otherwise: sort by
```

Let's try *ls -la* to list all the files and directories including hidden ones (Figure 1.7):

Figure 1.7 The outcome of executing *ls -la*

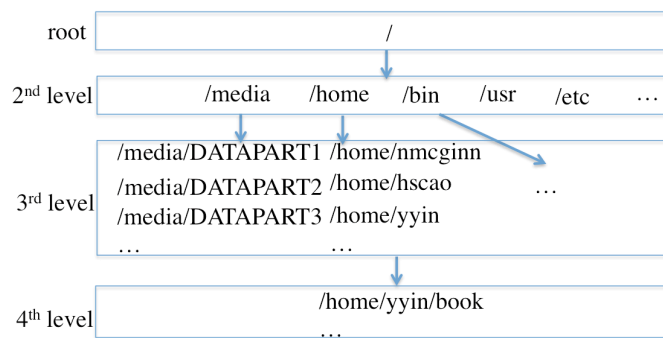


```
[yyin@gaea book]$ ls -la
total 3332
drwxr-xr-x. 2 yyin petunia  4096 Sep  2 16:24 .
drwx----- 10 yyin petunia  4096 Sep  2 16:21 ..
-rw-r--r--. 1 yyin petunia 3399727 Nov 26  2003 yeast.aa
```

Here we saw three items, among which only *yeast.aa* is a real file. “.” means the current folder (also called the **working directory**), while “..” means the parent/upper directory. These are called relative paths (relative to your current folder). To find the absolute path, use the *pwd* command; in this example, “.” is */home/yyin/book*, and “..” is */home/yyin*. We will explain with further details.

Note that the Linux file systems have a tree structure (Figure 1.8). “/” is the **root** directory and all absolute paths of all folders and files start from “/”. There are often many folders under “/” and for safety reasons most of them are system folders that only system administrator (or root user) can access, particularly for the right to write and execute (see section 1.9). General users normally only have the full set of rights under their home directory. For example, my home shown above is in */home/yyin*. I can create, remove, move and copy any folders and files within my home. When you log in, you start out in your home directory. The programs and files necessary to run the computer are found in directories under */bin* or */usr* or some place similar. Most of your work will occur within your home directory, although you will be using tools stored elsewhere.

Figure 1.8 Linux tree structure of file system



For navigating the directory tree, two commands are essential: *pwd* and *cd*, which we introduced a little bit above. Here are more descriptions: *pwd* (print working directory) tells you where you are. To issue this command, type “*pwd*” on the command line, then press the *Enter* key. The computer will return “/home/yyin”, which tells you that you are in the yyin directory which is underneath the home directory, which is underneath the root (/) directory.

The second useful command is *cd* (change directory). The command “*cd*” by itself will bring you back to your home directory from wherever you are. Most of the time, *cd* is followed by the path to the directory you wish to move to. For instance, “*cd /home/majohns*” will move you from wherever you are to the /home/majohns directory.

An important concept in Unix file systems is the “**path**”, which is the list of sub-directories leading to where you are. The path can be **absolute** or **relative**. An **absolute path** ALWAYS starts with /, which means that it is a path from the root directory to where you are. Thus, /home/yyin is the path leading from the root (/) to its sub-directory “home”, and then to the sub-directory of home called yyin. The *pwd* command gives you the absolute path to the directory you are in.

Relative paths NEVER start with /. The **relative path** leads from where you are to where you want to go, moving through the directory tree. A critical trick here: “..” refers to the directory one level above where you are in the tree. For instance, if you are in /home/yyin, the command “*cd ..*” will bring you to the /home directory, and the command “*cd ../../*” will bring you all the way to the root directory. If you wanted to move from /home/yyin to /home/majohns, you would type in the command “*cd ../majohns*”. This command would first bring you up one level to /home, then down one level to /home/majohns.

The directory you are in is sometimes referred to as “.” (dot). On many Unix machines, scripts in your home directory can only be executed if you explicitly tell the computer that it is located there, by adding “./” to the front of the command name. For instance, if you have a script you want to execute called “*run_blast*” located in your home directory, you will probably have to type “*./run_blast*” to execute it.

1.7 More commands for file and directory manipulation

We already knew that if you want to create a new directory, use *mkdir*. If you want to remove an empty directory, use *rmdir*. However, if we want to delete the non-empty book directory, we have to exit the book directory first (*cd ..*) and then do “*rm -rf book*”. Here we used the *rm* command. Actually, “*rm*” is often used to remove files. However, be careful to *rm* files or directories, as removed items normally cannot be recovered. This is especially true when you use the *-rf* options: they remove all files from the directory you specify and all sub-directories under it. Use *-i* option (interactive) when removing things if you want to be careful.

If you want to move files or directories from one folder to the other, use *mv*. For example:

mv: move the file to the folder one level above your working directory

```
[yyin@gaea book]$ mv yeast.aa ..
```

Now move it back, recalling that “.” refers to your current directory:

```
[yyin@gaea book]$ mv ../yeast.aa .
```

mv: can also change file names (case sensitive)

```
[yyin@gaea book]$ mv yeast.aa yeast.AA
```

Now change it back:

```
[yyin@gaea book]$ mv yeast.AA yeast.aa
```

cp: make a copy of a file

```
[yyin@gaea book]$ cp yeast.aa yeast.aa-2
```

rm: remove file

```
[yyin@gaea book]$ rm yeast.aa-2
```

cp: make a copy of an non-empty directory

```
[yyin@gaea book]$ cp -r ../book ../book2
```

rm: delete an non-empty directory

```
[yyin@gaea book]$ rm -rf ../book2
```

mv: change the folder name (if book2 does not exist any more)

```
[yyin@gaea book]$ mv ../book ../book2
```

1.8 Viewing files

Viewing a file is easy with the commands *more*, *less*, *cat*, *head* and *tail*. Note these commands cannot edit files, and are only used to view file content. Here are the main differences between these commands:

- *more*: displays the file content page by page, use *space* to page down and *Enter* to move down

one line at a time. When you page down, you cannot go back to the previous page.

- **less**: similar to more, but you can go back to previous pages use *PgUp* or *u*. Moreover you can type “/” followed by any word to search; type *q* to exit. For us, *less* is the most useful file viewing command.
- **cat**: display all the file content at once so you will only see the last page; but *cat* is mainly used to concatenate multiple files into one file.
- **head**: show the first few lines of a file. The default is to show the first 10 lines, but *head -20* shows the first 20 lines, etc.
- **tail**: same as head but show the last few lines (default is the last 10 lines, with *tail -50* showing the last 50 lines).

Unlike Windows notepad and wordpad, these commands do not need to load all the file content into the computer memory, so you can view **very large files** (except for *cat*).

1.9 Change file permissions

Normally on a Linux cluster or workstations, you are not the only user: other people could also use SSH clients to log into the same server from their desktops/laptops. Sometimes you want to share files or directories with your classmates or colleagues who are also using the server; you might need to change the permissions to the files to allow other view, write or execute your files (e.g. if the file is a perl script). Recall we can use *ls -la* to list the detailed information about files in a folder.

```
[yyin@gaea book]$ ls -la
total 3336
drwxr-xr-x. 2 yyin petunia 4096 Sep 3 15:49 .
drwx----- 12 yyin petunia 4096 Sep 3 16:11 ..
-rw-r--r--. 1 yyin petunia 3399727 Nov 26 2003 yeast.aa
```

For example, here *yeast.aa* was created in 2003, Nov 26 with size 3399727 bytes. “*-rw-r--r--*” has ten columns: the first one “-” means *yeast.aa* is a file; it will be “*d*” if it is a folder (directory). The following nine columns indicate the permission of read (*r*), write (*w*) and execute (*x*) granted for the user (first three columns), the group (middle three) and others (last three). “-” means no permission. So for *yeast.aa*, all can read, but only *yyin* can write the file.

But these can be changed by using the **chmod** command.

If I do not want any other people to read the file, I can remove read (*r*) permission for group (*g*) and others (*o*).

```
[yyin@gaea book]$ chmod go-r yeast.aa
[yyin@gaea book]$ ll
total 3328
-rw-r--r--. 1 yyin petunia 20 Sep 3 14:25 newfile
-rw----- 1 yyin petunia 3399727 Nov 26 2003 yeast.aa
```

If I want to grant all people in the `petunia` group (which my account `yyin` belongs to) the write permission, I can do `chmod g+w yeast.aa`.

If I wanted to make a file executable as a program by everyone, I could give them permission with `chmod a+x filename`.

1.10 Creating and editing files

In your bioinformatics work you will create and manipulate many files. In Linux, file names (and also directory names and commands) are case sensitive, unlike in Windows. This means that the files `data.txt` and `Data.txt` are different on Unix machines, but they would be the same file if you transferred them to a Windows machine. It is best to only use letters, numbers, hyphens, underscores, and periods in file names. You should not use *space* and some of the special characters in a file name, e.g. `/`, `\`, `*`, `?`, etc. Spaces and special characters tend to generate weird error messages when used on the command line, as Linux interprets these characters as something other than part of a file name.

The methods for viewing files are simple, but creating or editing a file is different. There are different programs too that allow you to create a new file or edit existing files, such as *vi*, *pico*, *nano* and *emacs*. Use *man* followed these commands to learn more. You don't need to learn all of them, but being familiar with at least one of them is necessary for bioinformatics data analysis. *pico* and *nano* are similar and very easy to use, especially for those who are used to GUI-based editors like Microsoft Word. When you open a file using these two commands, you will see command references at the bottom of the window, which suggest you what to do, e.g. `Ctrl+x` to exit.

However, here we are going to introduce the more powerful editor *vi*, although it is not very easy to learn. *emas* is more powerful but harder to learn. With *vi* you can conveniently create, edit and search a file's content. For example,

```
[yyin@gaea book]$ vi newfile.txt
```

After you are in the *vi* editor, type *i* and then you can start to insert words (**Figure 1.9**).

Figure 1.9 Edit mode of *vi*



After you are done, press *Esc*, followed by *:x* and then you save and exit.

Check the new file:

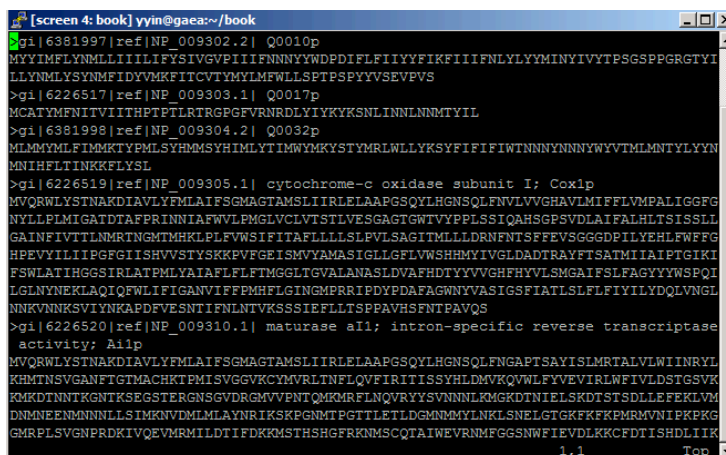
```
[yyin@gaea book]$ ls -l
total 3328
-rw-r--r--. 1 yyin petunia      20 Sep  3 14:25 newfile
-rw-r--r--. 1 yyin petunia 3399727 Nov 26  2003 yeast.aa
```

Now we have two files under the book folder.

Let's now use *vi* to open the existing yeast.aa file (Figure 1.10).

```
[yyin@gaea book]$ vi yeast.aa
```

Figure 1.10 Using *vi* to open a file



Note that you are now in the *vi* editor, which has two modes: **command mode** and **edit mode**. The default mode after you get into the *vi* editor is command mode; hit *i* on the keyboard to get you into the edit mode, in which we showed how to create newfile.txt, while hit *Esc* key to get you back to the command mode.

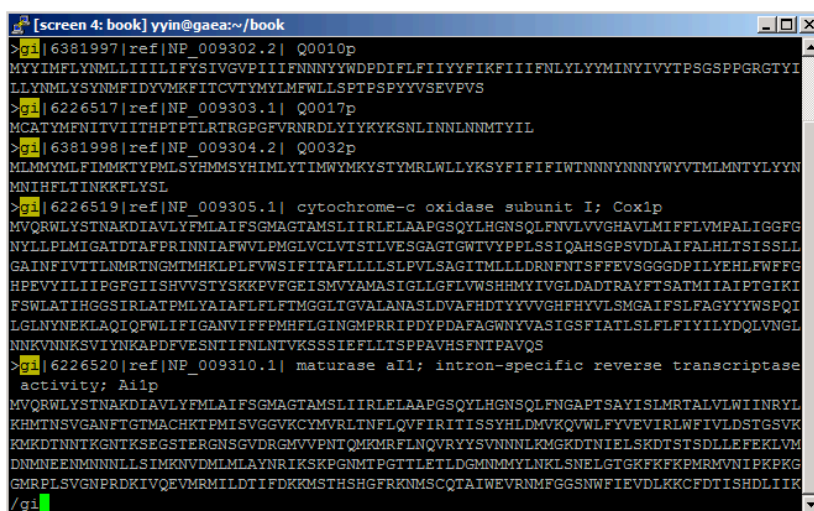
Exiting *vi*: *Esc* to the command mode and type *:x* or *:wq* to save any changes you made; type *:q!* to exit without saving changes; type *w* to save without exit.

Move the cursor: in both modes, using the *arrow* keys, *home*, *end*, *PgUp* and *PgDn* keys allows you to move the cursor around in the text screen. If you want to go to certain row, **in the command mode**, type the row number and *G*, the cursor will jump to the right row immediately. For example, *1* and *G* will get you back to the beginning of the file; just press *G* to get to the end of the file.

In command mode, you can do a number of different things. The most useful are:

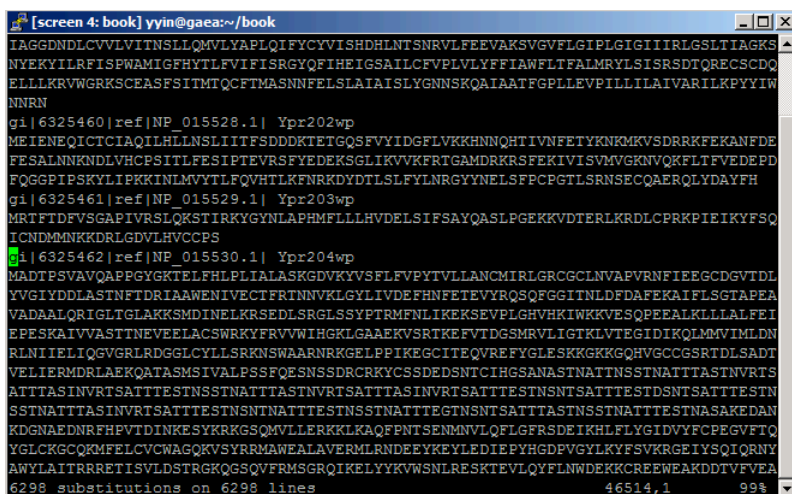
- Search (**Figure 1.11**): hit slash (“/”) to get the cursor to the left-bottom corner; you can type any word or letter to search it; type *n* to go to the next instance.

Figure 1.11 Search function in *vi* command mode



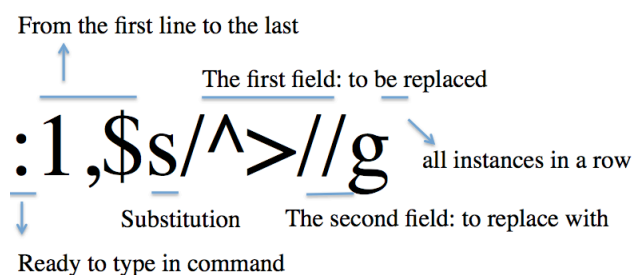
- Replace (**Figure 1.12**): hit *Esc* (at any time, hitting *Esc* to get back to the default status is the safest thing to do) and type “:I,\$s/^>/g” and then enter will replace all “>” to nothing (i.e. delete all “>”).

Figure 1.12 Substitute function in command mode



Here are some detailed explanation (**Figure 1.13**), “.” means you are now ready to type in a command; “1,\$” means from the first line to the last line in the file; “s” means substitution; “/^>//” means the first field “^>” will be replaced by the second field (nothing); “^” means the start of each line, so “^>” means all “>” at the start of a line (“>” in other places of a line will not be replaced); “g” means global. After this command is done, the cursor and screen will stop at the last replacement and the bottom line tells you how many substitutions were performed (in this case 6298). As you see all “>” are gone.

Figure 1.13 Detailed explanation of substitution syntax



The above are examples of some of the most often used *vi* utilities. There are many other *vi* tricks that you can find by a simple google search, e.g. <http://www.ccsf.edu/Pub/Fac/vi.html>.

Often you will want to create a FASTA format file by getting a sequence from NCBI, e.g. as a query file for BLAST search. You can do this using *vi* easily: 1) you type *vi query.fa* and 2) after you are in *vi*, type *i* to get into edit mode and copy a FASTA sequence, say from NCBI and right click to paste the file in *vi*, and then 3) hit *Esc* to exit edit mode and then *:x* to save the file and exit *vi*. Or you can write a Perl or Shell script file using *vi*.

1.11 Transferring files from remote locations

Doing bioinformatics means that you often could have access to multiple Unix workstations or servers. Often times you have the need to remotely access other machines or copy files and directories from one computer to another. You will need to move data into your Linux machine, downloading from GenBank or some other sources. If you are working on a remote server, you will need to download your processed results in some form to your personal computer or to a print server. Most communication is done through *ssh* (connecting from one machine to another, see **Figure 1.5**) and *sftp* (transferring data between two machines) commands. These methods are security-enhanced updates of the telnet and *ftp* programs. You may never use *telnet*, but *ftp* is still widely used for downloading large data sets. Both communication methods have both a command line version and a GUI (graphic) version; however we are going to talk about the command line versions here. *ssh* sends a stream of information between computers: you can send commands and see the results through *ssh*. *sftp* is used to transfer files of any size. *sftp* and *ftp* use almost identical commands.

To talk with another machine through SSH, you need to have access to an account on both machines. Invoke the program by typing e.g. "*ssh majohns@gaea.niu.edu*" on the command line. When a connection is made, you will be prompted for your password on the remote machine.

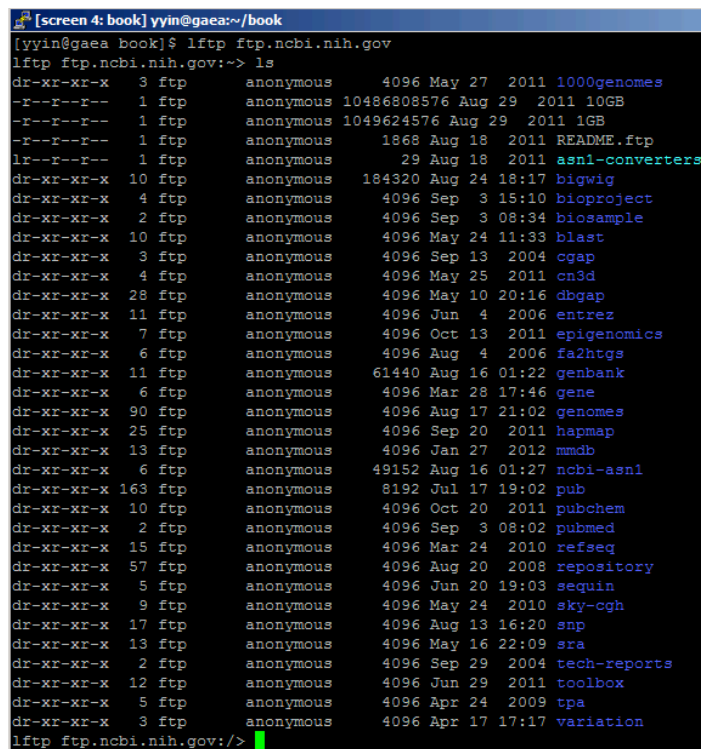
Once you are connected, you are effectively present in the remote account, and you can do whatever you need to: move between directories, look at files, run programs, download information, etc., using all the regular Unix commands.

sftp for file transfer is invoked the same way: "*sftp majohns@gaea.niu.edu*", and again you will be asked for a password. It starts an interactive session, using something like "*sftp>*" as a prompt. You can move between directories in the remote machine with *cd* commands and list directory contents with *ls*. More importantly, you can download any file with *get*: "*get some_data.txt*". And, you can upload a file with *put*: "*put someother_data.txt*". It is possible to upload or download multiple files simultaneously, with the *mput* and *mget* commands. These same commands can also be used with the regular (non-secure) *ftp* program.

Linux has *ftp* and *sftp* preinstalled with the OS. There are actually more convenient third-party ftp tools such as *ncftp* and *lftp*, which you have to install by yourself. For example, you can get *lftp* from <http://lftp.yar.ru/get.html> and install it in your home directory (we will show how to do this later). Suppose you have *lftp* installed, let's *lftp* to NCBI ftp site (**Figure 1.14**):

```
[yyin@gaea book]$ lftp ftp.ncbi.nih.gov
```

Figure 1.14 Execute *ls* after log in a remote ftp site



```
[screen 4: book] yyin@gaea:~/book
[yyin@gaea book]$ lftp ftp.ncbi.nih.gov
lftp ftp.ncbi.nih.gov:> ls
dr-xr-xr-x  3 ftp      anonymous    4096 May 27  2011 1000genomes
-r--r--r--  1 ftp      anonymous 10486808576 Aug 29  2011 10GB
-r--r--r--  1 ftp      anonymous 1049624576 Aug 29  2011 1GB
-r--r--r--  1 ftp      anonymous  1868 Aug 18  2011 README.ftp
lr--r--r--  1 ftp      anonymous   29 Aug 18  2011 asn1-converters
dr-xr-xr-x 10 ftp      anonymous 184320 Aug 24 18:17 bigwig
dr-xr-xr-x  4 ftp      anonymous  4096 Sep  3 15:10 bioproject
dr-xr-xr-x  2 ftp      anonymous  4096 Sep  3 08:34 biosample
dr-xr-xr-x 10 ftp      anonymous  4096 May 24 11:33 blast
dr-xr-xr-x  3 ftp      anonymous  4096 Sep 13  2004 cgap
dr-xr-xr-x  4 ftp      anonymous  4096 May 25  2011 cn3d
dr-xr-xr-x 28 ftp      anonymous  4096 May 10 20:16 dbgap
dr-xr-xr-x 11 ftp      anonymous  4096 Jun  4  2006 entrez
dr-xr-xr-x  7 ftp      anonymous  4096 Oct 13  2011 epigenomics
dr-xr-xr-x  6 ftp      anonymous  4096 Aug  4  2006 fa2htgs
dr-xr-xr-x 11 ftp      anonymous 61440 Aug 16 01:22 genbank
dr-xr-xr-x  6 ftp      anonymous  4096 Mar 28 17:46 gene
dr-xr-xr-x 90 ftp      anonymous  4096 Aug 17 21:02 genomes
dr-xr-xr-x 25 ftp      anonymous  4096 Sep 20  2011 hapmap
dr-xr-xr-x 13 ftp      anonymous  4096 Jan 27  2012 mmdb
dr-xr-xr-x  6 ftp      anonymous 49152 Aug 16 01:27 ncbi-asn1
dr-xr-xr-x 163 ftp     anonymous  8192 Jul 17 19:02 pub
dr-xr-xr-x 10 ftp      anonymous  4096 Oct 20  2011 pubchem
dr-xr-xr-x  2 ftp      anonymous  4096 Sep  3 08:02 pubmed
dr-xr-xr-x 15 ftp      anonymous  4096 Mar 24  2010 refseq
dr-xr-xr-x 57 ftp      anonymous  4096 Aug 20  2008 repository
dr-xr-xr-x  5 ftp      anonymous  4096 Jun 20 19:03 sequin
dr-xr-xr-x  9 ftp      anonymous  4096 May 24  2010 sky-cgh
dr-xr-xr-x 17 ftp      anonymous  4096 Aug 13 16:20 snp
dr-xr-xr-x 13 ftp      anonymous  4096 May 16 22:09 sra
dr-xr-xr-x  2 ftp      anonymous  4096 Sep 29  2004 tech-reports
dr-xr-xr-x 12 ftp      anonymous  4096 Jun 29  2011 toolbox
dr-xr-xr-x  5 ftp      anonymous  4096 Apr 24  2009 tpa
dr-xr-xr-x  3 ftp      anonymous  4096 Apr 17 17:17 variation
lftp ftp.ncbi.nih.gov:>
```

Here we were not asked for a user name and password, because NCBI ftp site allows anonymous access. Otherwise you will have to use *-u* to supply your user name. Now we are in the NCBI's ftp site (**Figure 1.14**). You can use *ls* to list the data folders and files and *cd* to a particular folder to check what data files are there. After you locate a file you want, you can get that file using *get* (**Figure 1.15**). Again if you are getting a file not in your current folder, you need provide a full path to that file.

Figure 1.15 Use wildcard to list files

```
lftp ftp.ncbi.nih.gov: /> cd blast/db/FASTA/
lftp ftp.ncbi.nih.gov:/blast/db/FASTA> ls *.gz
-r--r--r-- 1 ftp anonymous 91553 Nov 26 2003 alu.a.gz
-r--r--r-- 1 ftp anonymous 24465 Nov 26 2003 alu.n.gz
-r--r--r-- 1 ftp anonymous 4283092 Nov 26 2003 drosoph.aa.gz
-r--r--r-- 1 ftp anonymous 36924008 Nov 26 2003 drosoph.nt.gz
-r--r--r-- 1 ftp anonymous 834124838 Sep 2 15:09 env_nr.gz
-r--r--r-- 1 ftp anonymous 2901146883 Sep 2 15:10 env_nt.gz
-r--r--r-- 1 ftp anonymous 1502833250 Sep 2 15:12 est_human.gz
-r--r--r-- 1 ftp anonymous 793883313 Sep 2 15:12 est_mouse.gz
-r--r--r-- 1 ftp anonymous 11335176523 Sep 2 15:17 est_others.gz
-r--r--r-- 1 ftp anonymous 7299848672 Sep 2 15:21 gss.gz
-r--r--r-- 1 ftp anonymous 7273942919 Sep 2 15:25 htgs.gz
-r--r--r-- 1 ftp anonymous 9422424979 Sep 2 15:29 human_genomic.gz
-r--r--r-- 1 ftp anonymous 33071994 Aug 12 02:15 igSeqNt.gz
-r--r--r-- 1 ftp anonymous 4548338 Aug 11 02:15 igSeqProt.gz
-r--r--r-- 1 ftp anonymous 5404187 Sep 2 15:31 mito.aa.gz
-r--r--r-- 1 ftp anonymous 27794334 Sep 2 15:31 mito.nt.gz
-r--r--r-- 1 ftp anonymous 182330058 Sep 2 15:31 month.aa.gz
-r--r--r-- 1 ftp anonymous 595618 Sep 2 15:31 month.est_human.gz
-r--r--r-- 1 ftp anonymous 20 Sep 2 15:31 month.est_mouse.gz
-r--r--r-- 1 ftp anonymous 29222181 Sep 2 15:31 month.est_others.gz
-r--r--r-- 1 ftp anonymous 13519366 Sep 2 15:31 month.gss.gz
-r--r--r-- 1 ftp anonymous 236172361 Sep 2 15:31 month.htgs.gz
-r--r--r-- 1 ftp anonymous 56653888 Sep 2 15:31 month.nt.gz
-r--r--r-- 1 ftp anonymous 4771047091 Sep 2 15:33 nr.gz
-r--r--r-- 1 ftp anonymous 11388236114 Sep 2 15:38 nt.gz
-r--r--r-- 1 ftp anonymous 17281866680 Sep 2 15:46 other_genomic.gz
-r--r--r-- 1 ftp anonymous 161177693 Sep 2 15:49 pataa.gz
-r--r--r-- 1 ftp anonymous 3343984021 Sep 2 15:51 patnt.gz
-r--r--r-- 1 ftp anonymous 13028788 Sep 2 15:51 pdbaa.gz
-r--r--r-- 1 ftp anonymous 459434 Sep 2 15:51 pdbnt.gz
-r--r--r-- 1 ftp anonymous 201259138 Sep 2 15:51 sts.gz
-r--r--r-- 1 ftp anonymous 125466460 Sep 2 15:51 swissprot.gz
-r--r--r-- 1 ftp anonymous 881144 Jan 13 2010 vector.gz
-r--r--r-- 1 ftp anonymous 79151750567 Sep 2 16:17 wgs.gz
-r--r--r-- 1 ftp anonymous 1951194 Nov 26 2003 yeast.aa.gz
-r--r--r-- 1 ftp anonymous 3732371 Nov 26 2003 yeast.nt.gz
lftp ftp.ncbi.nih.gov:/blast/db/FASTA> get yeast.aa.gz
```

Here in **Figure 1.15**, `ls *.gz` is to list all files ending with `gz`. This is because there are too many files in the folder and we are only interested in `gz` files. “*” is called wildcard, meaning any possible letters and characters (see **section 1.17**). You can also use it outside `lftp`.

Use `Ctrl+d` or `bye` to exit `lftp`.

`wget` is a program useful for downloading files from both FTP and HTTP sites. HTTP is used for World Wide Web applications: most web site URLs start with `http://`. `wget` is non-interactive: you simply enter the necessary options and arguments on the command line and the file is downloaded for you. For instance: the command “`wget ftp://ftp.ncbi.nlm.nih.gov/blast/db/16SMicrobial.tar.gz`” will download this blast database from NCBI into your working directory.

If you want to copy files between two Linux machines for both you have access to, there is a more convenient way than invoking any of the `ftp` family tools.

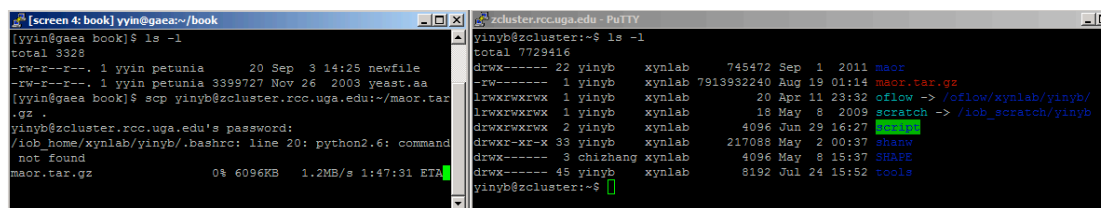
For example, I have access to another Linux server called `zcluster` and I want to copy one file from `zcluster` (in Georgia) to my local cluster (in Illinois) (**Figure 1.16**). I can do:

```
[yinyin@gaea book]$ scp yinyb@zcluster.rcc.uga.edu:~/maor.tar.gz .
```

Here `scp` is the command, `yinyb@zcluster.rcc.uga.edu:~/maor.tar.gz` is the remote file address: `yinyb` is my account on `zcluster`; `zcluster.rcc.uga.edu` is the remote server name (you can also use

IP address); “~/maor.tar.gz” is the path to the file on the remote server (yes, ~ is my home directory). After hitting *Enter*, I will be asked for the password to access the remote server using my account *yinyb*. Supply it and hit *Enter*, and the copying job will be started.

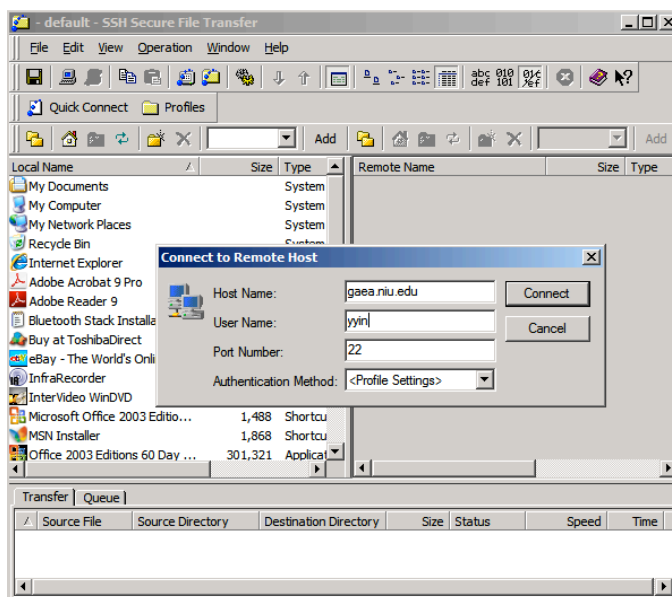
Figure 1.16 Use *scp* command to remotely copy files between two Linux machines



Copying a folder recursively will need an option *-r* after *scp*.

What if you want to transfer files/folders between your Windows laptop/desktop and the remote Linux server? You will need to install **SSH secure Shell** client (<http://ce.uml.edu/SSH3.2.9.exe>) on your Windows machine, which not only have a ssh client (similar to PuTTY) but also a file transfer client. After it is installed and opened, it appears as follows (**Figure 1.17**):

Figure 1.17 Use SSH Secure File Transfer client to transfer file between a Windows machine a Linux machine



1.12 Input and output redirection

Unix has a special way to direct input and output from commands or programs. By default, the input is from keyboard (called standard input, *stdin*): you type in a command and Shell takes the command and executes it. The standard output by default is to the terminal screen (*stdout*); if the command or program failed, you will also have standard errors dumped to the terminal screen (*stderr*).

However, if you do not want the output dumped to the screen, you can use “>” to redirect/write the output into a file. If the programs or commands failed to run (or did not fail but report some non-result message, e.g. when BLAST found some special characters other than standard letters in

DNA or protein sequences), you can use “2>” to dump the error message to a second file; or if you are not interested in the error messages, just dump it to nowhere (*/dev/null*).

For example, you want to save the first 20 lines of *yeast.aa*:

```
[yyin@gaea book]$ head -20 yeast.aa > yeast.aa.head
```

If somehow you misspelled *yeast.aa* and you don’t want the error message dumped to the screen:

```
[yyin@gaea book]$ head -20 yeast.ab 2> yeast.aa.2
```

```
[yyin@gaea book]$ cat yeast.aa.2
```

```
head: cannot open `yeast.ab' for reading: No such file or directory
```

When you use “>” to redirect output, the contents of the file you are redirecting to are overwritten and lost. If instead you want to append results to the end of an existing file, you can use “>>” to do that. For example:

```
[yyin@gaea book]$ tail -20 yeast.aa >> yeast.aa.head
```

Now *yeast.aa.head* has the first 20 lines and the last 20 lines of *yeast.aa*.

1.13 Pipes

Even more useful, Unix Shell also allows the output from one command to be passed to another command as input using **pipes** (“|”). This makes the processing of files using Shell very convenient and very powerful: you do not need to write intermediate files and load all data into the memory. For example, combining different Unix commands for text processing is like passing an item through a manufacturing pipeline when you only care about the final product:

```
[yyin@gaea book]$ cat yeast.aa | grep '^>' | wc
6298  31386 385460
```

Here are the three steps of processing: 1) “*cat yeast.aa*” is to dump the *yeast.aa* file; “|” is used to direct the output to the second step; 2) “*grep '^>'*” is used to filter the input (the output of last step) to only keep lines starting with “>”, i.e. the description line of the FASTA format sequences; 3) the output of the second step is then passed to the “*wc*” command, which is used to calculate the number of lines, the number of word and the number of bytes.

Therefore the above command line is used to calculate how many protein FASTA sequences are in the *yeast.aa* file. This is impossible to do if using Windows, while using Unix this is done in a blink.

1.14 More powerful commands to process text files

Here are some very useful Unix commands that can be combined to perform text processing: *grep*,

egrep, cut, sort, uniq, sed and *awk*. For example:

```
[yyin@gaea book]$ cat yeast.aa | grep '^>' | sed 's/>/' | awk '{print $1}' | head
-5
gi|6381997|ref|NP_009302.2|
gi|6226517|ref|NP_009303.1|
gi|6381998|ref|NP_009304.2|
gi|6226519|ref|NP_009305.1|
gi|6226520|ref|NP_009310.1|
```

This is used to extract the GenBank IDs: “*sed 's/>/'*” is used to trim the “>” (remember we can do the same thing in *vi*). *sed* is the command, meaning stream editor. With different options, it is used for multi-purposes of text processing, e.g. search and replace, search and delete, search and print etc. In this example, the option *'s/>/'* is explained as shown in **Figure 1.13**, i.e. used to find ‘>’ in the text and replace it with nothing (delete).

“*awk '{print \$1}'*” is used to extract the first field of each line (i.e. get rid of the description information following the IDs). Unlike other Linux commands, *awk* is more advanced and like a programming language. It has built-in functions for numeric and string operations, and you can even write if/else statement and loop controls in it. We will have more examples later in this chapter. Here we add another step to keep only the accession numbers:

```
[yyin@gaea book]$ cat yeast.aa | grep '^>' | sed 's/>/' | awk '{print $1}' | cut -f4
-d'|' | head -5
NP_009302.2
NP_009303.1
NP_009304.2
NP_009305.1
NP_009310.1
```

Here “*cut -f4 -d'|'*” is used to extract the fourth field separated by “|” (it’s just a field separator and not called pipe in the quotation marks). Try to *man* these commands to get to know their powerful options.

If you want to use **BLAST** to find similar proteins to your query (GenBank protein CAE52450.1), here is how you can run BLAST on a local Linux terminal (supposing BLAST is installed already):

The first compulsory step is to format the database:

```
[yyin@gaea book]$ makeblastdb -in yeast.aa -dbtype prot
```

Ant then run the BLAST:

```
[yyin@gaea book]$ blastp -query query.fa -db yeast.aa -out query.fa.out -outfmt 7
```

Figure 1.18 BLAST tabular format output

```

yyin@gaea:~/book
[yyin@gaea book]$ less query.fa.out | head
# BLASTP 2.2.26+
# Query: gi|45720158|emb|CAE52450.1| Prm9p [Saccharomyces cerevisiae]
# Database: yeast.aa
# Fields: query id, subject id, % identity, alignment length, mismatches, gap opens, q. start, q. end, s. start, s. end, value, bit score
# 43 hits found
gi|45720158|emb|CAE52450.1| gi|6319334|ref|NP_009418.1| 82.27 299 52 1 1 299 1 298 0.0 508
gi|45720158|emb|CAE52450.1| gi|6321385|ref|NP_011462.1| 89.87 237 24 0 63 299 1 237 2e-158 444
gi|45720158|emb|CAE52450.1| gi|6319336|ref|NP_009419.1| 56.30 238 95 3 63 299 1 230 1e-88 266
gi|45720158|emb|CAE52450.1| gi|6321387|ref|NP_011464.1| 56.30 238 95 3 63 299 1 230 1e-84 256
gi|45720158|emb|CAE52450.1| gi|6319331|ref|NP_009414.1| 47.86 234 121 1 63 296 1 233 7e-65 205

```

Here query.fa contains the FASTA format protein sequence of CAE52450.1. “-outfmt 7” option is to specify the output format to be tabular format: the default output format has much more information including the alignment, while here we do not need that. The following command line uses pipe and a combination of different Shell commands to parse the BLAST tabular format output (each field is explained in **Figure 1.18**) to extract hit IDs. In total 32 unique proteins were found.

```

[yyin@gaea book]$ cat query.fa.out | grep -v '^#' | cut -f2 | sort | uniq | wc -l
32

```

“**grep** -v ‘^#’” is used to get ride of the comment lines, starting with “#”. The “-v” option means invert match. “**cut -f2**” is to cut the second field: we did not specify the field separator using the “-d” option because the default separator is tabular space, which is what BLAST output uses. Since one query protein may hit a subject protein multiple times (BLAST reports gapped alignments), **sort** and **uniq** is used to remove redundant hit IDs.

If we apply an E-value cutoff to only show matches with E-value < 1e-80 (**Figure 1.19**), we can use **awk** to do the filtering:

Figure 1.19 Use **awk** to filter BLAST output

```

[screen 4: book] yyin@gaea:~/book
[yyin@gaea book]$ cat query.fa.out | awk '$11<1e-80'
gi|6319334|ref|NP_009418.1| 82.27 299 52 1 1 299 1 298 0.0 508
gi|45720158|emb|CAE52450.1| gi|6321385|ref|NP_011462.1| 89.87 237 24 0 63 299 1 237 2e-158 444
gi|6319336|ref|NP_009419.1| 56.30 238 95 3 63 299 1 230 1e-88 266
gi|45720158|emb|CAE52450.1| gi|6321387|ref|NP_011464.1| 56.30 238 95 3 63 299 1 230 1e-84 256
[yyin@gaea book]$

```

egrep is similar to **grep**, but can extract lines with multiple patterns (**grep** can only do one pattern unless applying some options). **sort** is used to sort the lines according to defined field: the default is on the first column in alphabetical order.

1.15 Compress and archive files

We have shown that many data files available at the NCBI ftp site are archived and compressed: the file names end in “.tar.gz”.

Disk space is a limited resource, and you have to monitor how much disk space you have used. To check the disk space usage, use the **du** (disk usage) command:

```

[yyin@gaea book]$ du -hs .
1.3G .

```

Again, the dot (.) means the working folder.

To check how much space you have already occupied in your entire home directory:

```
[yyin@gaea book]$ du -hs ~/
871G   /home/yyin/
```

To check how much space left on the entire storage file system, use the `df` command:

```
[yyin@gaea book]$ df -h
Filesystem                Size      Used Avail Use% Mounted on
/dev/mapper/vg_gaea-lv_root
                          50G   4.7G   43G   10% /
tmpfs                      12G    76K    12G    1% /dev/shm
/dev/sda1                  485M    40M   421M    9% /boot
172.20.1.253:/data1        40T   763G   40T    2% /data1
172.20.1.254:/data2       40T   792G   40T    2% /data2
/dev/mapper/vg_gaea-lv_home
                          384G   45G   320G   13% /ext
172.20.1.254:/home2       9.9T   3.1T   6.4T   33% /home
```

Here under `/home`, we still have 6.4 terabytes left.

If you want to compress your files, use `gzip` (similar commands: `compress`, `zip`, `gunzip` and `bzip2`).

```
[yyin@gaea book]$ gzip yeast.aa
```

You will get a file `yeast.aa.gz`. To uncompress the file:

```
[yyin@gaea book]$ gzip -d yeast.aa.gz
```

You may also want to compress all the files in a folder into a single file; you will need `tar` to create an archive of the folder and then use compression commands to compress it. Or you can do it in one command:

```
[yyin@gaea book]$ tar -zcf book.tar.gz .
```

To uncompress and release all files in the folder:

```
[yyin@gaea book]$ gzip -d book.tar.gz | tar xf -
```

1.16 Search files/folders in the file system

After a while you will have created many folders in your home and many files in each of your folders. Sometimes you will have a hard time finding a particular file. You can use `ls -lt` to list files in a folder in the order of modification time, or `ls -lt | less` to list them one page at a time. Or, you can use `find` to help you locate a file or files:

For example, there are 383 files under “`../fungal/dbcan-model/`” relative to the book folder (the absolute path would be `/home/yyin/fungal/dbcan-model/`):

```
[yyin@gaea book]$ ls ../fungal/dbcan-model/ # press tab key in the end
Display all 383 possibilities? (y or n)
```

List the files that do NOT end with *hmm* (there are 63 of them):

```
[yyin@gaea book]$ ls ../fungal/dbcan-model/ | grep -v hmm$ | head
CBM10.hmm.bgi-gut.out
CBM10.hmm.bgi-gut.out.c-evalue.ps
CBM10.hmm.bgi-gut.out.ps
CBM10.hmm.cow.out
CBM10.hmm.cow.out.c-evalue.ps
CBM10.hmm.cow.out.ps
CBM10.hmm.jgi-v3.5.pr.fa.out
CBM10.hmm.jgi-v3.5.pr.fa.out.c-evalue.ps
CBM10.hmm.jgi-v3.5.pr.fa.out.ps
CBM10.hmm.metahit.out
[yyin@gaea book]$ ls ../fungal/dbcan-model/ | grep -v hmm$ | wc
    63     63    1672
```

Here “**grep** -v *hmm*\$” is used to filter out file names ending with *hmm*, so \$ means the end of a word/string.

Similarly, we can also use the *find* command: find and list the files that end with *out* in the file names:

```
[yyin@gaea book]$ find ../fungal/dbcan-model/ -name "*out"
../fungal/dbcan-model/dockerin.hmm.nr.out
../fungal/dbcan-model/SLH.hmm.jgi-v3.5.pr.fa.out
../fungal/dbcan-model/CBM10.hmm.bgi-gut.out
../fungal/dbcan-model/cohesin.hmm.nr.out
../fungal/dbcan-model/CBM10.hmm.cow.out
../fungal/dbcan-model/dockerin.hmm.bgi-gut.out
../fungal/dbcan-model/SLH.hmm.cow.out
../fungal/dbcan-model/cohesin.hmm.cow.out
../fungal/dbcan-model/CBM10.hmm.metahit.out
../fungal/dbcan-model/SLH.hmm.bgi-gut.out
../fungal/dbcan-model/dockerin.hmm.metahit.out
../fungal/dbcan-model/CBM10.hmm.nr.out
../fungal/dbcan-model/SLH.hmm.metahit.out
../fungal/dbcan-model/SLH.hmm.nr.out
../fungal/dbcan-model/cohesin.hmm.bgi-gut.out
../fungal/dbcan-model/cohesin.hmm.jgi-v3.5.pr.fa.out
../fungal/dbcan-model/dockerin.hmm.cow.out
../fungal/dbcan-model/CBM10.hmm.jgi-v3.5.pr.fa.out
../fungal/dbcan-model/dockerin.hmm.jgi-v3.5.pr.fa.out
../fungal/dbcan-model/cohesin.hmm.metahit.out
```

Note that you can also do “`ls ../fungus/dbcan-model/*out`” to get the same result. “*” is **wildcard**, representing any letters, numbers and signs, so “*out” means any files ending with out. If you do not know which folder the files are in, find will recursively go through all folders under the specified path. The command: `find book -name *out` will start at the book folder and search every folder under it for files ending in out.

1.17 Regular expressions

Regular expression (**regex** or **regexp**) is a very powerful tool for text processing and widely used in text editors (e.g. vi) and programming languages (e.g. Shell commands: `sed`, `awk`, `grep` and `perl`, `python`, `PHP`) to automatically edit (match and replace strings) texts. Finding and replacing exact words or characters are simple, e.g. the `sed` example shown in section 1.13. However, if you want to match multiple words or characters, you will need wildcards or patterns.

Here is a list of commonly used **wildcards** and **patterns**:

```
* any numbers of letters, numbers and characters except for spaces and special
characters, e.g. () []+\/$@#%&;,?
. any single letter, number and character including special characters
\w any letter (a-z and A-Z)
\d any number (0-9)
+ previous items at least one times, e.g. \w+ matches words of any sizes
{n} previous items n times, e.g. \w{5} matches words with exactly five letters
\s space
\t tabular space
\n new line
^ start of a line
$ end of a line
^$ an empty line, i.e. without anything between ^ and $
[] create your own pattern, e.g. [ATGC] matches one of the four letters only, [ATGC]{2}
matches two such letters
```

We have shown some examples above using the wildcards in the Shell command line. Note that not all of them work in the command line, while they all work in different programming languages (see the perl programming chapters). Here are more examples used in the command line:

```
# remove lines without any letters or spaces (empty lines)
[yyin@gaea book]$ cat ../db/JGI/2199352009.faa | sed '/^$/d' | head

# grep lines starting with > followed by 1 or 2
[yyin@gaea book]$ cat ../db/JGI/2199352009.faa | grep '^>[1-2]' | head

# print the 4th column which matches “Human”; awk -F"\t" is to specify the separator as a tabular
```

space and `$4~/Human/` is to specify the matching pattern (within two slashes, `~` means matching)

```
[yyin@gaea book]$ less ../db/JGI/jgi-v3.5.list | awk -F"\t" '$4~/Human/{print $4}'  
| less
```

1.18 Installation of bioinformatics applications

A lot of bioinformatics tools and software have been developed to run on Unix. So it is essential to have some experience in installing, configuring and troubleshooting various bioinformatics software, which are most often written using C language and need to be compiled. Normally in their released packages (often as tarred and zipped files), all the bioinformatics tools have **README** or **INSTALL** or other install instruction files, which provide simple or detailed information on installation. So make sure to read these files before installation. There are also many bioinformatics programs written in *Java*, *perl* and *python* languages; these programs can run directly and do not need to be installed.

Since *lftp* is very useful for downloading tools and data from remote public websites, let's use *lftp* as an example to show how to install tools with source codes:

```
[yyin@gaea tools]$ wget -q http://ftp.yar.ru/pub/source/lftp/lftp-4.4.0.tar.gz  
[yyin@gaea tools]$ $ ll lftp-4.4.0.tar.gz  
-rw-rw-r-- 1 yyin yyin 2487662 Sep 27 08:17 lftp-4.4.0.tar.gz  
[yyin@gaea tools]$ gzip -dc lftp-4.4.0.tar.gz | tar xf -  
[yyin@gaea tools]$ cd lftp-4.4.0/  
[yyin@gaea lftp-4.4.0]$ less INSTALL
```

In the `INSTALL` file, we see the suggested procedure is, `configure`, `make` and then `make install`, so we will just follow it:

```
[yyin@gaea lftp-4.4.0]$ ./configure --prefix=/home/yyin/tools/lftp
```

Usually installing a program from the source code requires you have the right to write to the root folders (`/user/local/bin/` or `/user/bin/`), because by default the programs will be installed there. However, you often do not have the root right if you are using a Linux cluster. Therefore, the `-prefix` option is used to specify a folder (make sure to include the absolute path!) in my home where I have the right to write.

```
[yyin@gaea lftp-4.4.0]$ make  
[yyin@gaea lftp-4.4.0]$ make install
```

If we check `/home/yyin/tools/lftp`, the executable commands are there.

Now we will see how to install pre-compiled programs, which is much easier than installing source codes. For example, if you want to install BLAST, the most popular tool used in sequence analysis, first *lftp* to NCBI's ftp site:


```

[yyin@gaea tools]$ lftp ftp.ncbi.nih.gov
lftp ftp.ncbi.nih.gov:> cd blast/executables/LATEST/
cd ok, cwd=/blast/executables/LATEST
lftp ftp.ncbi.nih.gov:/blast/executables/LATEST>ls
-r--r--r--  1 ftp      anonymous   10822 Sep 11 20:35 ChangeLog
-r--r--r--  1 ftp      anonymous  165161376 Sep 11 20:35 ncbi-blast-2.2.27+-2.i686.rpm
-r--r--r--  1 ftp      anonymous  10878140 Sep 11 20:35 ncbi-blast-2.2.27+-2.src.rpm
-r--r--r--  1 ftp      anonymous  141809122 Sep 11 20:35 ncbi-blast-2.2.27+-2.x86_64.rpm
-r--r--r--  1 ftp      anonymous  165139074 Sep 11 20:35 ncbi-blast-2.2.27+-ia32-linux.tar.gz
-r--r--r--  1 ftp      anonymous  51694959 Sep 11 20:35 ncbi-blast-2.2.27+-ia32-win32.tar.gz
-r--r--r--  1 ftp      anonymous  137293007 Sep 11 20:35 ncbi-blast-2.2.27+-sparc64-solaris.tar.gz
-r--r--r--  1 ftp      anonymous  13038673 Sep 11 20:35 ncbi-blast-2.2.27+-src.tar.gz
-r--r--r--  1 ftp      anonymous  15977983 Sep 11 20:35 ncbi-blast-2.2.27+-src.zip
-r--r--r--  1 ftp      anonymous  267817605 Sep 11 20:36 ncbi-blast-2.2.27+-universal-macosx.tar.gz
-r--r--r--  1 ftp      anonymous  51768859 Sep 11 20:36 ncbi-blast-2.2.27+-win32.exe
-r--r--r--  1 ftp      anonymous  64604561 Sep 11 20:36 ncbi-blast-2.2.27+-win64.exe
-r--r--r--  1 ftp      anonymous  141796815 Sep 11 20:36 ncbi-blast-2.2.27+-x64-linux.tar.gz
-r--r--r--  1 ftp      anonymous  133863324 Sep 11 20:36 ncbi-blast-2.2.27+-x64-solaris.tar.gz
-r--r--r--  1 ftp      anonymous  64372601 Sep 11 20:36 ncbi-blast-2.2.27+-x64-win64.tar.gz
-r--r--r--  1 ftp      anonymous  269327941 Sep 11 20:36 ncbi-blast-2.2.27+.dmg

```

All we see above are different BLAST executable packages, pre-compiled for different computer platforms: Windows, MAC and different Unix platforms (e.g. Linux and Solaris). Besides, computer processor types also matter, e.g. x64 or ia32. To find out what our machine is:

```

[yyin@gaea tools]$ uname -a
Linux gaea.niu.edu 2.6.32-220.el6.x86_64 #1 SMP Wed Nov 9 08:03:13 EST 2011 x86_64
x86_64 x86_64 GNU/Linux

```

So we will need the **x64** version. We then get the right package accordingly:

```

lftp ftp.ncbi.nih.gov:/blast/executables/LATEST> get
ncbi-blast-2.2.27+-x64-linux.tar.gz
lftp ftp.ncbi.nih.gov:/blast/executables/LATEST> bye

```

Installing the BLAST executable is very easy:

```

[yyin@gaea tools]$ gzip -dc ncbi-blast-2.2.26+-x64-linux.tar.gz | tar xf -

```

We now have a new folder called **ncbi-blast-2.2.26+** and the BLAST commands will be in the folder:

```

[yyin@gaea tools]$ ll ncbi-blast-2.2.26+/bin/
total 431640
-rwxr-xr-x. 1 yyin petunia 16780504 Feb  9  2012 blastdb_aliastool
-rwxr-xr-x. 1 yyin petunia 16560056 Feb  9  2012 blastdbcheck
-rwxr-xr-x. 1 yyin petunia 23379544 Feb  9  2012 blastdbcmd

```

```

-rwxr-xr-x. 1 yyin petunia 26064088 Feb  9 2012 blast_formatter
-rwxr-xr-x. 1 yyin petunia 26059576 Feb  9 2012 blastn
-rwxr-xr-x. 1 yyin petunia 26055448 Feb  9 2012 blastp
-rwxr-xr-x. 1 yyin petunia 26055448 Feb  9 2012 blastx
-rwxr-xr-x. 1 yyin petunia 16697400 Feb  9 2012 convert2blastmask
-rwxr-xr-x. 1 yyin petunia 26204280 Feb  9 2012 deltablast
-rwxr-xr-x. 1 yyin petunia 16724304 Feb  9 2012 dustmasker
-rwxr-xr-x. 1 yyin petunia   51345 Jun 28 2010 legacy_blast.pl
-rwxr-xr-x. 1 yyin petunia 18155128 Feb  9 2012 makeblastdb
-rwxr-xr-x. 1 yyin petunia 17320328 Feb  9 2012 makemindex
-rwxr-xr-x. 1 yyin petunia 18234552 Feb  9 2012 makeprofiledb
-rwxr-xr-x. 1 yyin petunia 26117336 Feb  9 2012 psiblast
-rwxr-xr-x. 1 yyin petunia 26092632 Feb  9 2012 rpsblast
-rwxr-xr-x. 1 yyin petunia 26055448 Feb  9 2012 rpstblastn
-rwxr-xr-x. 1 yyin petunia 17134504 Feb  9 2012 segmasker
-rwxr-xr-x. 1 yyin petunia 26096824 Feb  9 2012 tblastn
-rwxr-xr-x. 1 yyin petunia 26055448 Feb  9 2012 tblastx
-rwxr-xr-x. 1 yyin petunia   11874 Dec 23 2011 update_blastdb.pl
-rwxr-xr-x. 1 yyin petunia 20045264 Feb  9 2012 windowmasker

```

Installation of another popular tool **HMMER** is very similar:

```

[yyin@gaea tools]$ wget -q
ftp://selab.janelia.org/pub/software/hmmer3/3.0/hmmer-3.0-linux-intel-x86_64.tar.gz
[yyin@gaea tools]$ gzip -dc hmmer-3.0-linux-intel-x86_64.tar.gz | tar xf -
[yyin@gaea tools]$ ls hmmer-3.0-linux-intel-x86_64/binaries/

```

Here is another very useful bioinformatics tool package: **EMBOSS**. We have to install from the source, following the same procedure as we did for lftp:

```

[yyin@gaea tools]$ wget ftp://emboss.open-bio.org/pub/EMBOSS/emboss-latest.tar.gz
[yyin@gaea EMBOSS-6.5.7]$ gzip -dc emboss-latest.tar.gz | tar xf -

```

We now have a new folder called *EMBOSS-6.5.7*

```

[yyin@gaea EMBOSS-6.5.7]$ cd EMBOSS-6.5.7/

```

Again we should find and check out the `INSTALL` or `README` or `readme` files, although normally we can follow the *configure*, *make* and then *make install* paradigm.

```

[yyin@gaea EMBOSS-6.5.7]$ ./configure --prefix=/home/yyin/tools/emboss

```

Again *--prefix* is to specify the installation folder.

```

[yyin@gaea EMBOSS-6.5.7]$ make
[yyin@gaea EMBOSS-6.5.7]$ make install

```

EMBOSS now is installed in `/home/yyin/tools/emboss/bin`.

Bioperl is another very commonly used tool. It is basically a collection perl object-oriented modules that are dedicated for biological sequence analysis. To install bioperl, there are multiple ways: http://www.bioperl.org/wiki/Installing_BioPerl_on_Unix. Here we show how to install from source code under your personal folder (i.e. without root).

```
[yyin@gaea tools]$ wget -q http://bioperl.org/DIST/BioPerl-1.6.1.tar.gz
[yyin@gaea tools]$ gzip -dc BioPerl-1.6.1.tar.gz | tar xf -
[yyin@gaea BioPerl-1.6.1]$ cd BioPerl-1.6.1
[yyin@gaea BioPerl-1.6.1]$ perl Build.PL --install_base /home/yyin/tools/bioperl
[yyin@gaea BioPerl-1.6.1]$ ./Build test
[yyin@gaea BioPerl-1.6.1]$ ./Build install
```

Note here we did not follow the *configure*, *make* and then *make install* paradigm, because we are installing perl packages (not C). That's why we always need to check the INSTALL or README files.

You may install many other bioinformatics tools in the similar ways, either from executable or source codes.

One last thing that you will need to do in order to efficiently call these tools is to add the paths to these tools to the `.bashrc` file. Otherwise, every time you run *lftp*, you have to provide the full path: `/home/yyin/tools/lftp/bin/lftp`, which is not necessary. In Unix, file names can start with a dot, so that they are hidden when you do *ls*. You may however list them using *ls -a* and you will see `.bashrc` is under your home directory. You may use *vi* to edit this file:

```
[yyin@gaea tools]$ vi ~/.bashrc
```

And add (and edit if needed) the following lines in the file:

```
export PERL5LIB=/home/yyin/tools/bioperl/lib/perl5/
PATH=$PATH:/home/yyin/tools/ncbi-blast-2.2.26+/bin
PATH=$PATH:/home/yyin/tools/blast-2.2.25/bin
PATH=$PATH:/home/yyin/tools/hmmer-3.0-linux-intel-x86_64/binaries/
PATH=$PATH:/home/yyin/tools/lftp/bin
PATH=$PATH:/home/yyin/tools/emboss/bin
export PATH
```

You will also need to log out and log in again or execute the `.bashrc` file to make it in effect:

```
[yyin@gaea tools]$ . ~/.bashrc
```

The first and last lines are to define the environmental variables `PERL5LIB` and `PATH`. By doing this, when you call bioperl modules in your perl scripts, the script will go to `/home/yyin/tools/bioperl/lib/perl5/` to find the called modules. Or when you call BLAST, HMMER or EMBOSS commands, the system will know the path to the folder where it can find the commands. This way, you do not need to type the full path to each program when you run them.

Note that if you install all programs in the default system folders e.g. `/usr/local/bin` or `/usr/bin` or `/bin`, you do not need to modify the `.bashrc` file, as these folders and system folders are already included in the `PATH` environment variable.

An alternate method for accomplishing this is to create a **symbolic link** (symlink) from your folder to the location of the program. For example, the command: `ln -s blastn /home/yyin/tools/blast-2.2.25/bin/blastn` creates a link that allows you to type `blastn` on the command line instead of having to write the whole path.

With `bioperl` installed, you can easily combine Unix command and **perl one-liner** capability for sequence analysis. Traditionally you write `perl` script inside a text editor and save it as a separate file. For `perl` one-liner, you use `perl` as a command such as `grep`, `sort`, `sed` and `awk` etc. The `perl` codes are put inside a pair of quotation marks and before the codes, you use `"perl -e"` to tell `perl` to interpret the followed content inside quotation marks as a one-liner.

Note that although this is very convenient for small tasks, it has one major problem: **it is very difficult to debug** the one-liner `perl` script. So you should always write the codes into a `perl` script as a separate file when you are developing complex programs.

However, for small tasks, this is very useful. Here is an example for combining Linux commands and `perl` one-liner script:

```
[yyin@gaea book]$ cat query.fa.out | cut -f2 | sort -u | perl -e 'use Bio::SeqIO;$new=Bio::SeqIO->new(-file=>"yeast.aa",-format=>"fasta");while($seq=$new->next_seq){$b{$seq->id}=$seq;}while(<>){chomp;print ">".$_. "\n".$b{$_}->seq. "\n";}' | less
```

The above command is used to parse the BLAST result to extract the FASTA sequences of hit proteins in **Figure 1.13**. The syntax for the `perl` script is detailed in the `perl` chapters.

1.19 An example to deal with many files with Shell loops

Sometimes you have multiple (could be even hundreds or thousands) similar data files and you need to run exactly the same command to process these files. For example, we will download all sequenced bacterial genomes in NCBI and we want to count how many chromosomes each bacterial genome has. How do we do that?

```
[yyin@gaea book]$ lftp ftp.ncbi.nih.gov
lftp ftp.ncbi.nih.gov:~> cd genomes/Bacteria
lftp ftp.ncbi.nih.gov:/genomes/Bacteria> ls | wc
  2199  19793  210975
```

There are 2199 items (folders and files).

```
lftp ftp.ncbi.nih.gov:/genomes/Bacteria> ls *.gz
```

```

-r--r--r-- 1 ftp      anonymous 433434845 Nov 27 08:02 all.GeneMark.tar.gz
-r--r--r-- 1 ftp      anonymous 97622797 Nov 27 08:04 all.Glimmer3.tar.gz
-r--r--r-- 1 ftp      anonymous 216848828 Nov 27 10:57 all.Prodigal.tar.gz
-r--r--r-- 1 ftp      anonymous 4304533838 Nov 27 08:15 all.asn.tar.gz
-r--r--r-- 1 ftp      anonymous 1439253077 Nov 27 08:28 all.faa.tar.gz
-r--r--r-- 1 ftp      anonymous 2081674048 Nov 27 08:34 all.ffn.tar.gz
-r--r--r-- 1 ftp      anonymous 2294630906 Nov 27 09:04 all.fna.tar.gz
-r--r--r-- 1 ftp      anonymous 8696506 Nov 27 09:36 all.frn.tar.gz
-r--r--r-- 1 ftp      anonymous 6389135416 Nov 27 09:46 all.gbk.tar.gz
-r--r--r-- 1 ftp      anonymous 519806234 Nov 27 10:36 all.gff.tar.gz
-r--r--r-- 1 ftp      anonymous 174516854 Nov 27 10:39 all.ptt.tar.gz
-r--r--r-- 1 ftp      anonymous 2490316 Nov 27 10:40 all.rnt.tar.gz
-r--r--r-- 1 ftp      anonymous 343017 Nov 27 10:40 all.rpt.tar.gz
-r--r--r-- 1 ftp      anonymous 3900691520 Nov 27 10:48 all.val.tar.gz

```

These tarred and zipped files contain relevant data of all sequenced bacteria. For example, all.faa.tar.gz contains FASTA format protein sequences and all.fna.tar.gz contains FASTA format genomic DNA sequences. Here we download the all.rpt.tar.gz file, which contains general statistics of every sequenced genome.

```

lftp ftp.ncbi.nih.gov:/genomes/Bacteria> get all.rpt.tar.gz
lftp ftp.ncbi.nih.gov:/genomes/Bacteria> bye
[yyin@gaea book]$ mkdir bac
[yyin@gaea book]$ mv all.rpt.tar.gz bac
[yyin@gaea book]$ cd bac
[yyin@gaea bac]$ gzip -dc all.rpt.tar.gz | tar xf -

```

If we list content in the bac folder, it contains 2177 folders.

```

[yyin@gaea bac]$ ls | wc
    2177    2177    87408

[yyin@gaea bac]$ ls | head -5
Acaryochloris_marina_MBIC11017_uid58167
Acetobacterium_woodii_DSM_1030_uid88073
Acetobacter_pasteurianus_IFO_3283_01_42C_uid158377
Acetobacter_pasteurianus_IFO_3283_01_uid59279
Acetobacter_pasteurianus_IFO_3283_03_uid158373

```

Each folder is named using the species name plus a unique ID given by NCBI. Inside the folder are the chromosome statistics files, one file one chromosome. In the following genome, it has 10 chromosomes: only one of them is the primary genome, others being plasmids)

```

[yyin@gaea bac]$ ll Acaryochloris_marina_MBIC11017_uid58167
total 40
-rw-r--r--. 1 yyin petunia 290 May 12 2011 NC_009925.rpt
-rw-r--r--. 1 yyin petunia 299 Oct 22 2010 NC_009926.rpt
-rw-r--r--. 1 yyin petunia 299 Oct 22 2010 NC_009927.rpt

```

```

-rw-r--r--. 1 yyin petunia 299 Oct 22 2010 NC_009928.rpt
-rw-r--r--. 1 yyin petunia 299 Oct 22 2010 NC_009929.rpt
-rw-r--r--. 1 yyin petunia 299 Oct 22 2010 NC_009930.rpt
-rw-r--r--. 1 yyin petunia 299 Oct 22 2010 NC_009931.rpt
-rw-r--r--. 1 yyin petunia 299 Oct 22 2010 NC_009932.rpt
-rw-r--r--. 1 yyin petunia 299 Oct 22 2010 NC_009933.rpt
-rw-r--r--. 1 yyin petunia 287 Oct 22 2010 NC_009934.rpt
[yyin@gaea bac]$ ls Acaryochloris_marina_MBIC11017_uid58167 | wc
    10     10     140

```

Then coming back to our original question: how do we find out how many chromosomes each bacterial genome has? We cannot do *wc* one by one, because there are 2177 folders. We can, however, use Shell loops to do the calculation for all folders in once:

```

[yyin@gaea bac]$ for i in `ls ./`; do echo $i; ls $i | wc -l; done | head
Acaryochloris_marina_MBIC11017_uid58167
10
Acetobacterium_woodii_DSM_1030_uid88073
1
Acetobacter_pasteurianus_IFO_3283_01_42C_uid158377
7
Acetobacter_pasteurianus_IFO_3283_01_uid59279
7
Acetobacter_pasteurianus_IFO_3283_03_uid158373
7

```

The syntax is very simple:

```

for i in `some command line`
do
some command line
done

```

Because we wrote the **for loop** as one-liner, we have to put ';' as a punctuation separator. Here `` (not ') is used to execute a command inside it. In the above example, the command is *ls ./*, used to list all items in the current *bac* folder, which are the bacterial folder names. Each line of the output (denoted as *\$i*, carrying one bacteria folder name each time) will be going through the loop and be used in the command: here actually two commands separated by ';': *echo \$i; ls \$i | wc -l*.

The output is in a format that two rows for a bacterial folder, first row being the folder name (from *echo \$i*) and second row being the chromosome number (*ls \$i | wc -l*). We can add a *perl* one-liner to further process the output to get an easier-to-read output:

```

[yyin@gaea bac]$ for i in `ls ./`; do echo $i;ls $i | wc -l; done | perl -e '@a=<>;chomp @a;for($i=0;$i<=$#a;$i=$i+2){print $a[$i]."\t".$a[$i+1]."\n";}' | head -5

```

```

Acaryochloris_marina_MBIC11017_uid58167 10
Acetobacterium_woodii_DSM_1030_uid88073 1
Acetobacter_pasteurianus_IFO_3283_01_42C_uid158377 7
Acetobacter_pasteurianus_IFO_3283_01_uid59279 7
Acetobacter_pasteurianus_IFO_3283_03_uid158373 7

```

Now the output has two columns, corresponding to the bacterial name and the chromosome number. We are not going to explain the *perl* one-liner part, as it will be covered in the *perl* chapters.

We can further sort the output according to the second column in a descending order:

```

[yyin@gaea bac]$ for i in `ls ./`; do echo $i;ls $i | wc -l; done | perl -e '@a=<>;chomp @a;for($i=0;$i<=$#a;$i=$i+2){print $a[$i]."\t".$a[$i+1]."\n";}' | sort -k 2,2nr | head
Salmonella_enterica_serovar_Weltevreden_2007_60_3289_1_uid178014 67
Borrelia_crocidurae_Achema_uid162335 40
Staphylococcus_aureus_uid178791 30
Borrelia_burgdorferi_B31_uid57581 22
Borrelia_burgdorferi_JD1_uid161197 21

```

We see that some bacteria even have 67 chromosomes!

1.20 Send jobs to the background and manage jobs

Often programs cannot finish quickly. You can send such jobs to the background; otherwise after you press *Enter*, you will just hang there until the program ends, and you will not be able to do other operations in the terminal. You can send the job to the **background** by simply adding “&” in the end of the command line. For example:

```

[yyin@gaea book]$ blastp -query query.fa -db yeast.aa -out query.fa.out -outfmt 6 &
Or
[yyin@gaea tools]$ wget -q http://bioperl.org/DIST/BioPerl-1.6.1.tar.gz &

```

Sometimes a program will not finish before you have to close the terminal session (or have it be automatically terminated by the server). If this happens, your program will be terminated **without finishing**. If you expect your program will run for a very long time, e.g. longer than 10 hours, you may put “**nohup**” before your command; this ensures that even if you close the terminal, the program will still run in the background until it is finished and you can log in again the next day to check the output. For example:

```

[yyin@gaea book]$ nohup blastp -query yeast.aa -db yeast.aa -out yeast.aa.ava.out -outfmt 6 &

```

You will get an additional file *nohup.out* in the working folder and this file will be empty if nothing wrong happened.

If you used *nohup* and *&* to have multiple jobs sent to the background, you can check the status by using *top* (Figure 1.20):

Figure 1.20 Run *top* command to check running jobs

```

top - 17:46:25 up 38 days, 8:24, 19 users, load average: 0.08, 0.02, 0.01
Tasks: 343 total, 2 running, 341 sleeping, 0 stopped, 0 zombie
Cpu(s): 13.3%us, 0.0%sy, 0.0%ni, 86.6%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 24593028k total, 22077148k used, 2515880k free, 266364k buffers
Swap: 26836984k total, 596k used, 26836388k free, 18708384k cached

  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM    TIME+  COMMAND
 25200 yyin      20   0 71840 23m  15m  R 99.9   0.1   0:07.50 blastp
 2083  root      20   0 9204  608  392  S   0.3   0.0  11:51.69 irqbalance
    1  root      20   0 21472 1356 1068  S   0.0   0.0   1:21.91 init
    2  root      20   0   0     0     0  S   0.0   0.0   0:08.97 kthreadd
    3  root      RT   0   0     0     0  S   0.0   0.0   0:01.28 migration/0
    4  root      20   0   0     0     0  S   0.0   0.0   0:00.37 ksoftirqd/0
    5  root      RT   0   0     0     0  S   0.0   0.0   0:00.00 migration/0
    6  root      RT   0   0     0     0  S   0.0   0.0   0:00.00 watchdog/0
    7  root      RT   0   0     0     0  S   0.0   0.0   0:00.80 migration/1
    8  root      RT   0   0     0     0  S   0.0   0.0   0:00.00 migration/1
    9  root      20   0   0     0     0  S   0.0   0.0   0:00.80 ksoftirqd/1
   10  root      RT   0   0     0     0  S   0.0   0.0   0:00.08 watchdog/1
   11  root      RT   0   0     0     0  S   0.0   0.0   0:05.39 migration/2
   12  root      RT   0   0     0     0  S   0.0   0.0   0:00.00 migration/2
   13  root      20   0   0     0     0  S   0.0   0.0   0:02.28 ksoftirqd/2
   14  root      RT   0   0     0     0  S   0.0   0.0   0:00.96 watchdog/2
   15  root      RT   0   0     0     0  S   0.0   0.0   0:07.53 migration/3

```

Or use *jobs* to check the status:

```

[yyin@gaea book]$ jobs
[1]+  Running                  blastp -query yeast.aa -db yeast.aa -outfmt 6 -out
yeast.aa.ava.out &

```

If there are multiple jobs running, you will see “[2]”, “[3]” and so on. If you want to terminate the first job, do *kill -9 %1*. You may also use *top* to find the job id (PID), in this case, 25200, and use *kill*:

```

[yyin@gaea book]$ kill -9 25200
[yyin@gaea book]$
[1]+  Killed                  blastp -query yeast.aa -db yeast.aa -outfmt 6 -out
yeast.aa.ava.out

```

Sometimes you want to recall a recently used command. You can press the *Up* arrow to go back to previous commands. You can also use the *history* command to print all the previous commands on the screen. For example:

```

[yyin@gaea book]$ history | tail -5
 573 perl doc Bio::SeqIO
 574 ll ../bioperl/lib/perl5/Bio/
 575 vi ~/.bashrc
 576 history | less
 577 history | tail

```

You may also use *w* to find out who is using the current node (here I used *grep* to only show jobs of user yyin):

```

[yyin@gaea book]$ w | grep yyin
yyin pts/0 :pts/22:S.2 Fri14 18:33m 0.66s 0.66s /bin/bash

```



```

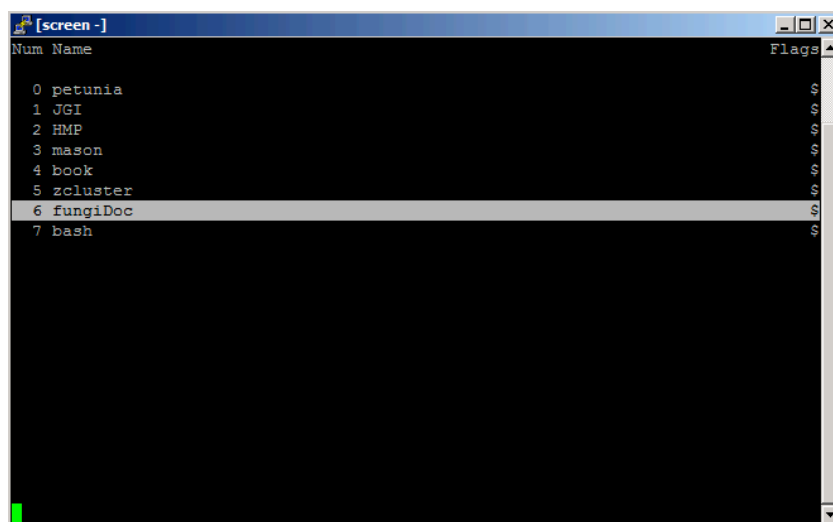
yyin pts/3 :pts/22:S.0 Fri14 13days 0.08s 0.08s /bin/bash
yyin pts/8 131.156.211.10 11:13 0.00s 0.06s 0.04s screen -x
yyin pts/9 :pts/22:S.1 Fri14 19:12m 0.42s 0.42s /bin/bash
yyin pts/10 :pts/22:S.4 Fri14 0.00s 0.37s 0.01s w
yyin pts/12 :pts/22:S.5 Fri14 11:16 2.47s 2.46s ssh yinyb@zclus
yyin pts/13 :pts/22:S.3 Fri14 16days 0.03s 0.03s /bin/bash
yyin pts/19 :pts/22:S.6 Fri14 7:57 0.61s 0.61s /bin/bash
yyin pts/21 :pts/22:S.7 Fri14 2days 0.00s 0.00s /bin/bash
yyin pts/22 131.156.41.238 Fri14 2days 0.50s 0.48s screen -x

```

1.21 Screen: having more screens/tabs in a single terminal window

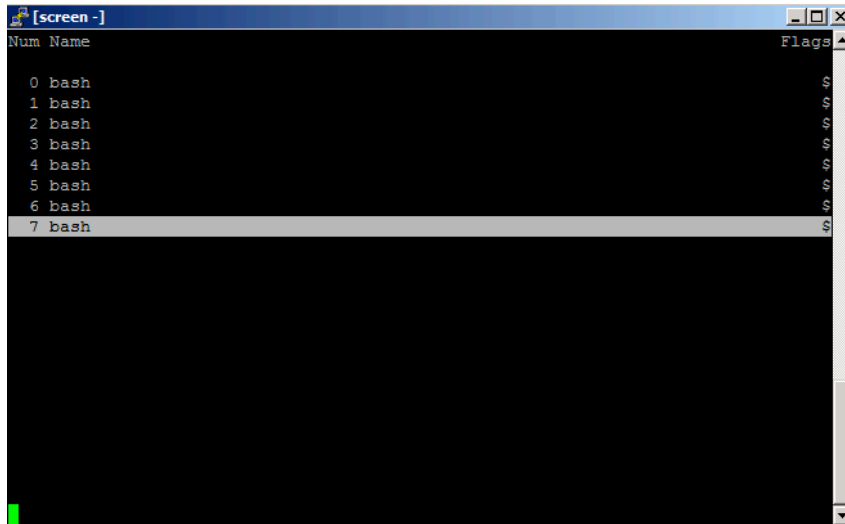
When you become more advanced in Unix and bioinformatics, you often found that you have to open multiple terminals to work on them simultaneously. This is just like when you surf on the Internet, you often open multiple windows of IE or Firefox and you don't want to close one window to open another. Most of the web browsers now support multiple tabs in one window. And Unix command line terminal also allows the similar thing with the `screen` command (**Figure 1.21**).

Figure 1.21 An example of a list tabs in screen session



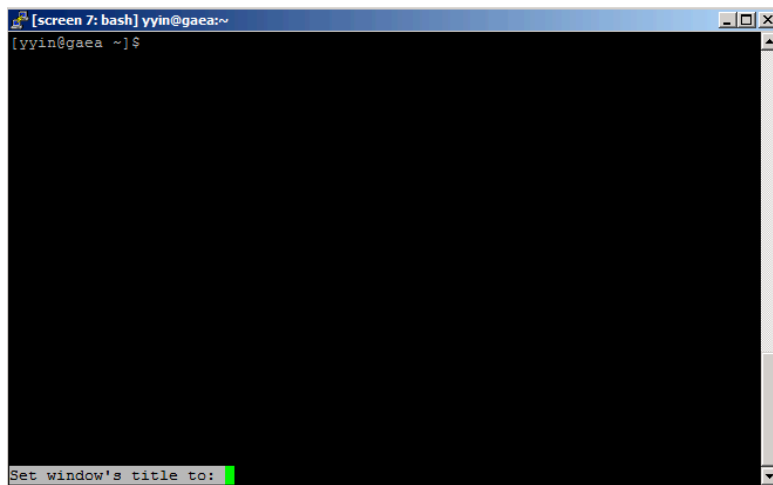
Here we have created seven screens (or seven tabs) within one terminal. What we did is that we typed `screen` and then we are in the screen session. We then pressed `Ctrl+a` followed by `c` to open tabs in the screen session. We did this seven times to have seven tabs. Note that every time when you do `Ctrl+a` and then `c`, the screen just blinks and you will not notice that you are opening a new tab. However, if you do `Ctrl+a` and then “ (or `shift+’`) and then you switch to the tab list as shown in (**Figure 1.22**):

Figure 1.22 Tab list before change the names



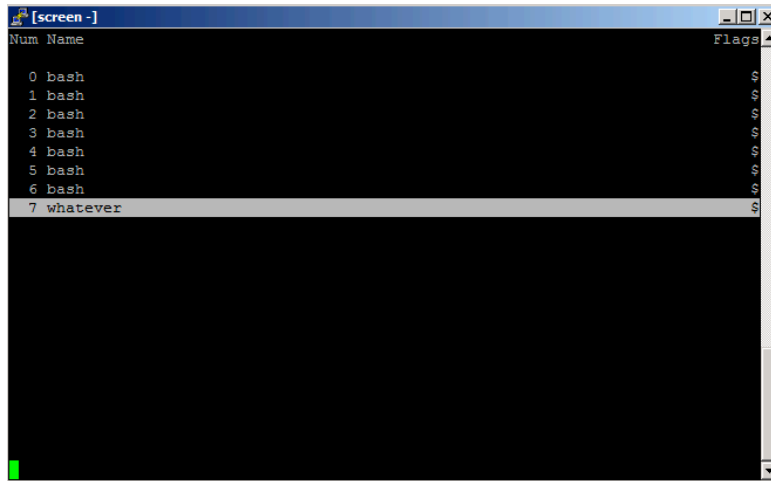
Here you see that all the seven tabs have the same name “*bash*”. You can move the highlighted band by using *up* and *down* arrows. To change the name of each tab, you have to go into one tab and then do *Ctrl+a* and then *Shift+a*; and you will see a highlighted band in the left-bottom and then you can remove “*bash*” and type in the name you want to call the tab and do *Enter* (**Figure 1.23**).

Figure 1.23 Change the name of each tab



You then can check if the change was made successfully by going to the tab list again using *Ctrl+a* followed by “” (**Figure 1.24**):

Figure 1.24 After the name is changed



Here we changed the tab 7 to “*whatever*”. You can move the highlighted band to other tab and change the name similarly. You then will have seven tabs (terminal windows) in one screen so that you can do different things simultaneously (e.g. *ssh* to other remote Linux servers that you have access to) and you can switch between different tabs by going to the tab list.

If you are leaving, you can do *Ctrl+a* followed by *d* to **detach** from the screen. The screen session is not ended and all your seven tabs are still up but in the background. If you are running different jobs in some tabs, they will not be terminated either (this is like using *nohup* but we think it is much better). The greatest thing is that, if you go back home and open a PuTTY terminal in your home computer and log in, you can attach the screen back again by doing *screen -x*; all tabs are still there and active. Of course you can also reattach the screen on your office computer on the second day you come to work.

Actually you can have multiple screen sessions up and use *screen -ls* to list them:

```
[yyin@gaea ~]$ screen -ls
There are screens on:
    23162.pts-20.gaea      (Detached)
    6621.pts-2.gaea      (Attached)
2 Sockets in /var/run/screen/S-yyin.
```

If you have multiple screen sessions, you have to do *screen -x 6621* to attach to the specific screen session *6621*. You can have the same screen session open in any numbers of computers.

The bad thing about screen is that if the computer where your screen session is started is down or rebooted, all the embedded tabs in the screen session will be lost. So always start a screen session on a Unix server that is not often rebooted and then you can have screen sessions lasting for a long time and basically you do not need to log out from any tabs. Every time you go back home, detach the screen (*Ctrl+a* and then *d*) and the second day you come back, just reattach it (*screen -x*).