Linux for Beginners – Part 2

3CPG Workshop

Robert Bukowski
Computational Biology Service Unit

http://cbsu.tc.cornell.edu/lab/doc/Linux_workshop_Part2_Nov_2011.pdf

Topics

CBSU/3CPG Lab

cbsuwrkst2,3,4 (Linux)
3 “interactive” machines with nice consoles (also accessible remotely)

cbsum1c1b00n (Linux)
cbsum1c2b00n (Linux)
31 “remote” machines

cbsulm01,cbsulm02
(Linux, 64 and 500 GB RAM)

Part 1: (March Nov. 7, 2011)

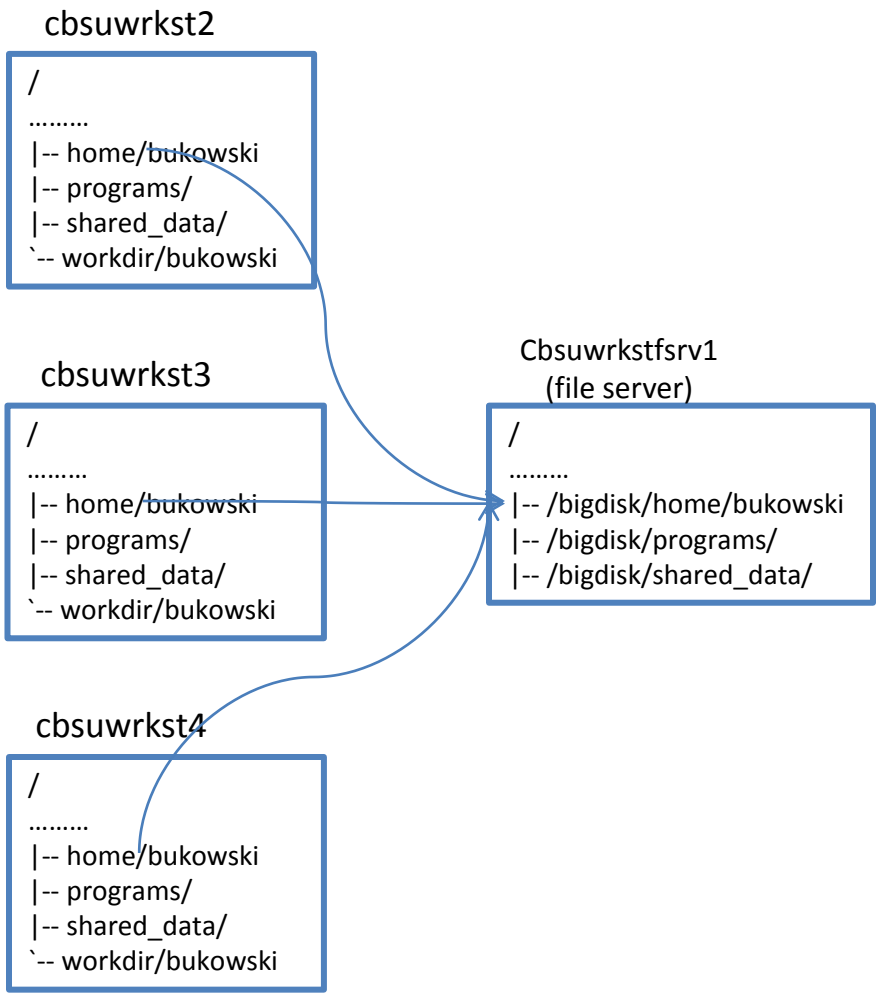
- ❑ Reserving time on 3CPG Lab workstations
- ❑ Logging in to a Linux workstation
- ❑ Terminal window and tricks
- ❑ Linux directory structure
- ❑ Working with files
- ❑ Working with text files

Part 2: (today)

- ❑ Transferring files to/from workstations
- ❑ Running applications
  - ➢ Note: this will only cover the Linux aspect of running applications; the functionality and the biological aspect will be covered in workshop **Using BioHPC Lab Software** on Nov. 28, 2011.
- ❑ Basics of scripting (shell and Perl)
  - ➢ Note: this will <u>not</u> teach you scripting – just get you started. We are planning a series of workshops on Perl in the fall – stay tuned. In the meantime - use multiple resources online (google “Perl tutorial”, for example).

Disk usage guidelines: Local vs. network directories

(3CPG LAB – specific)



<u>Network directories</u>

/home, /programs, /shared_data (with all subdirectories)

- Physically located on the file server
- Visible from all workstations
- Relatively SLOW access – DO NOT run any calculations there, avoid transferring large files there

<u>Local directories:</u>

/workdir (with all subdirectories), all other directories

- Physically attached to “its own” workstation
- Not visible from other workstations
- Fast access – all calculations should be run in **/workdir**

Disk usage guidelines

(3CPG lab specific)

Your home directory (e.g., /home/bukowski)

- Is network-mounted and therefore access to it is slow
- Visible from each workstation – no matter which one you log in to
- 200 GB quota will be imposed (may change depending on conditions)
- Use it to store files which you use frequently (reference genomes, index files) or which are small and hard to replace (scripts and executables)
- Never run any disk intensive applications (all Next-Gen tools are disk intensive) with your home directory (or any of its subdirectories) as the "current directory". Work on **/workdir** instead.

The **/workdir** directory

- Is local to its workstation (located on disks physically attached to the machine's controller)
- Not visible from other workstations
- Temporary – the content of /workdir may be erased after you log out. When you log in again, your files may be no longer there
- After you log in, create your own subdirectory in /workdir (if not already there)
- All the files to be used in processing have to be moved/copied to that subdirectory
- Applications have to be started in that subdirectory
- Important output files have to be copied back to the home directory or (better yet) out of the machine.

Checking disk space

<u>How much disk space is taken by my files?</u>

du –hs . *(displays combined size of all files in the current directory and recursively in all its subdirectories)*

du –h --max-depth=1 . *(as above, but sizes of each subdirectory are also displayed)*

How much disk space is available?

df -h

```
Filesystem            Size  Used Avail Use% Mounted on
/dev/mapper/VolGroup00-LogVol00
                      3.6T  236G  3.1T   7% /
/dev/sda1              99M   19M   75M  21% /boot
tmpfs                  12G     0   12G   0% /dev/shm
cbsuwrkstfsrv1.tc.cornell.edu:/bigdisk/home
                      6.9T  246G  6.3T   4% /home
cbsuwrkstfsrv1.tc.cornell.edu:/bigdisk/programs
                      6.9T  246G  6.3T   4% /programs
cbsuwrkstfsrv1.tc.cornell.edu:/bigdisk/shared_data
                      6.9T  246G  6.3T   4% /shared_data
```

/workdir is a part of it – this space is yours during your reservation

Your home directory is a part of it – you share space with other users

File transfer

between PC or Mac and a lab workstation

On Windows PC: install and use your favorite **sftp client** program, such as

- **winscp**: http://winscp.net/eng/index.php
- **CoreFTP LE**: http://www.coreftp.com/
- **FileZilla** (client): http://filezilla-project.org/
- … others…
- When connecting to Lab workstations from a client, use the **sftp** protocol. You will be asked for your user name and password (the same you use to log in to the lab workstations).
- Transfer text file in text mode, binary files in binary mode (the "default" not always right).
- All clients feature
  - File explorer-like graphical interface to files on both the PC and on the Linux machine
  - Drag-and-drop functionality

On a Mac: file transfer program is **fetch** (recommended by Cornell CIT)

- http://www2.cit.cornell.edu/services/systems_support/filefetch.html#fetchinst
- graphical user interface
- Drag-and-drop functionality

Large files (> 1 GB) should be transferred to your subdirectory under /workdir (e.g., /workdir/bukowski). Avoid storing such files in your home directory. Never process such files in home directory (or any subdirectory of your home)

File transfer

fixing Windows/Mac – Linux text file conversion problems

unix2dos my_file (convert a text file in linux format my_file to Windows/Mac format, i.e., change line endings)

dos2unix my_file (convert a text file my_file in Windows/Mac format to Linux format, i.e., change line endings)

File transfer

between a lab workstation and another Linux machine

Suppose we want to transfer a file from **cbsuss04.tc.cornell.edu** (another Linux machine; substitute "your" Linux machine here) and **cbsuwrkst2** lab workstation.

Option 1: when logged in to **cbsuwrkst2**, sftp to **cbsuss04** by running the following commands:

cd /workdir/bukowski *(this is where we want the file to be on cbsuwrkst2)*
sftp bukowski@cbsuss04.tc.cornell.edu *(instead of "bukowski", use your own user name on cbsuss04; you will be asked for password)*
cd /data/bukowski/reads *(on cbsuss04, go to the directory where the file is)*
get my_read.fastq *(transfer, or "get" the file from cbsuss04)*
quit *(exit sftp client and disconnect from cbsuss04 – we are back on cbsuwrkst2)*

Option 2: when logged in to **cbsuss04**, sftp to **cbsuwrkst2** by running the following commands:

cd /data/bukowski/reads *(this is where the file is on cbsuss04)*
sftp bukowski@cbsuwrkst2.tc.cornell.edu *(instead of "bukowski", use your own user name on cbsuss04; you will be asked your lab password)*
cd /workdir/bukowski *(on cbsuwrkst2, go to the directory where the file is supposed to be stored)*
put my_read.fastq *(transfer, or "put" the file on cbsuwrkst2)*
quit *(exit sftp client and disconnect from cbsuwrkst2– we are back on cbsuss04)*

File transfer

from web- and ftp sites to lab workstations

<u>Option 1: download using a web browser on workstation</u>. While logged in to the workstation, execute the following:

- **firefox** *(this will start the Firefox browser on the workstation)*
  - *If you are working remotely from a PC **and not using VNC**, you will need to have Xming running. Note: Firefox browser you just started is **running on Linux workstation**, your PC is just displaying the browser's window. May be slow on slow networks…*
- Navigate to the site you want to download the file from, click on download link. The browser will ask for destination directory (on the workstation !) to put the file in. Select a directory (should be in /workdir if the file is large) and let the browser complete the download.
- Close Firefox browser if no longer needed.

File transfer

from web- and ftp sites to lab workstations

<u>Option 2: run **wget** command on the workstation (if you know the URL of the file)</u>

- **Example:**

wget ftp://ftp.ncbi.nih.gov/blast/matrices/BLOSUM100

(will download the file BLOSUM100 from the NCBI FTP site and deposit it in the current directory under the name BLOSUM100)

- **another Example (the following should be typed on one line):**

wget -O e_coli_1000_1.fq
"http://cbsuapps.tc.cornell.edu/Sequencing/showseqfile.aspx?cntrl=646698859&laneid=487&mode=http&file=e_coli_1000_1.fq"

(the command above can be used to download files given by complicated URLs; note the **""** *marks around the link and the* **–O** *option which specifies the name you want to give the downloaded file)*

More about commands

- Each command is, in fact, an executable program stored somewhere on disk, usually in places like **/bin**, **/usr/bin**, or **/usr/local/bin**
  - **which mv** *(tells us where on disk the command mv is located)*
- Why can we just use **mv** rather than the full name **/bin/mv** ? Because of the <u>search path</u> environment variable which is defined for everybody. The you type **mv**, the system tries each directory on the search path one by one until it finds the corresponding executable.
  - **echo $PATH** *(displays the search path)*
  - Note: the current directory **./** is NOT in the search path. If you need to run a program located, say in your home directory, you need to precede it with **./**, for example, **./my_program**
- The next-gen analysis applications installed on the workstations are also in your **$PATH**. Thus, you can launch them using just the name rather than the full path:
  - Example: command **samtools** is equivalent to **/programs/bin/samtools/samtools**

Example project

Objective: align Illumina reads to D. Melanogaster genome

- Download data from FTP server (using Firefox or wget command)
- Extract files: reference genome (FASTA) and Illumina reads (FASTQ)
- Index reference genome (i.e., prepare it for use with BWA aligner program)
- Align reads using BWA aligner
- Convert alignments to SAM format
- Convert alignments in SAM format to BAM format

Download/unpack project data

Create a directory on local scratch disk (/workdir)

```
cd /workdir/bukowski
mkdir d_melanodaster
cd d_melanogaster
```

Download the tgz archive with sample files

```
wget ftp://cbsuftp.tc.cornell.edu/software/CBSUtools/Linux2workshop/fly_example.tgz
```

Unpack the tgz archive

```
tar -xzvf fly_example.tgz
```

Running applications. Example: genome indexing

Very general syntax for launching applications: **<path_to_application_executable> <options>**

First, cd to work directory and create a subdirectory for indexed genome

cd /workdir/bukowski/d_melanogaster
mkdir bwaindex

Now launch the actual indexing program

- We will run the program in directory under **/workdir/bukowski/d_melanogaster**
- We need the FASTA file with the genome, ***flygenome.fa***, in that directory
- We want the index files to end up in **/workdir/bukowski/d_melanogaster/bwaindex** and we want their names to start with "drosophila"
- Study the BWA manual (http://bio-bwa.sourceforge.net/bwa.shtml) to learn more about this program's options

bwa index –p bwaindex/drosophila flygenome.fa

Path to application executable

Program options

- After 2-3 minutes, the index files will be written to **bwaindex** (check this by doing "ls –al" in bwaindex directory!). Any information messages (the program "log") will be written to the screen.
- For larger genomes, the indexing step will take longer.

Running applications, cnt.

Saving log messages

- Normally, log messages are printed to the screen (and then lost)
  - Two streams: “standard output” (STDOUT), “standard error” (STDERR)
- To save them, redirect to files on disk – you can examine them later

```
bwa index –p bwaindex/drosophila flygenome.fa 1> run.log 2> run.err
```

Redirect screen output

Running a program in the background

- Normally, the program will run to completion (or crash), blocking the terminal window
- By putting an “&” at the end of command, we can send the program to the background
  - Terminal will return to prompt immediately – you will be able to continue working
  - Good for long-running programs (most programs of interest…)
  - Can run multiple programs simultaneously if more then 1 processor available on a machine and if there is enough memory

```
bwa index –p bwaindex/drosophila flygenome.fa 1> run.log 2> run.err &
```

Run in the background

Running applications

Checking on your application: the **top** command

To exit – just type **q**

```
top - 17:13:49 up 81 days,  2:13,  3 users,  load average: 0.81, 0.27, 0.09
Tasks: 136 total,   2 running, 134 sleeping,   0 stopped,   0 zombie
Cpu(s): 25.0%us,  0.1%sy,  0.0%ni, 75.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  24692152k total, 24528008k used,   164144k free,   182100k buffers
Swap: 26738680k total,      192k used, 26738488k free, 22737208k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 3108 bukowski  25   0  815m 805m  608 R 100.2  3.3   1:14.91 bwa
    1 root      15   0 10352  704  588 S  0.0  0.0   0:02.95 init
    2 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 migration/0
    3 root      34  19     0    0    0 S  0.0  0.0   0:00.07 ksoftirqd/0
    4 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 watchdog/0
    5 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 migration/1
    6 root      34  19     0    0    0 S  0.0  0.0   0:59.15 ksoftirqd/1
    7 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 watchdog/1
    8 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 migration/2
    9 root      34  19     0    0    0 S  0.0  0.0   0:24.62 ksoftirqd/2
   10 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 watchdog/2
   11 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 migration/3
   12 root      34  19     0    0    0 S  0.0  0.0   0:00.16 ksoftirqd/3
   13 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 watchdog/3
   14 root      10  -5     0    0    0 S  0.0  0.0   0:00.33 events/0
   15 root      10  -5     0    0    0 S  0.0  0.0   1:28.92 events/1
   16 root      10  -5     0    0    0 S  0.0  0.0   3:25.27 events/2
   17 root      10  -5     0    0    0 S  0.0  0.0   1:15.45 events/3
   18 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 khelper
  214 root      10  -5     0    0    0 S  0.0  0.0   0:00.01 kthread
  221 root      10  -5     0    0    0 S  0.0  0.0   0:00.01 kblockd/0
```

Running applications

<u>Checking on your application:</u>

the **ps** command – display info about all your processes – one of them should be **bwa**

ps –ef | grep bukowski

```
bukowski  3159 14936 99 17:26 pts/1    00:00:25 bwa index -p bwaindex/drosophila flygeome
.fa
bukowski  3162 14936  0 17:26 pts/1    00:00:00 ps -ef
bukowski  3163 14936  0 17:26 pts/1    00:00:00 grep bukowski
root     14769  3484  0 Feb18 ?        00:00:00 sshd: bukowski [priv]
bukowski 14771 14769  0 Feb18 ?        00:00:00 sshd: bukowski@notty
bukowski 14772 14771  0 Feb18 ?        00:00:00 /usr/libexec/openssh/sftp-server
root     14933  3484  0 Feb18 ?        00:00:00 sshd: bukowski [priv]
bukowski 14935 14933  0 Feb18 ?        00:00:01 sshd: bukowski@pts/1
bukowski 14936 14935  0 Feb18 pts/1    00:00:00 -bash
[bukowski@cbsuwrkst2 d_melanogaster]$
```

Process ID (PID)

Running time

Try **man ps** for more info about the **ps** command.

Running applications

<u>Stopping applications</u>

- If the application is running in the foreground (i.e., without “&”), it can be stopped with Ctrl-C (press and hold the Ctrl key, then press the “C” key) issued from the window (terminal) it is running in.
- If the application is running in the background (i.e., with “&”), it can be stopped with the **kill** command

kill -9 <PID>

Where <PID> is the process id obtained rom the **ps** command. For example, to terminate the **bwa** process form the previous slide, we would use

kill -9 3159

Try **man kill** for more info about the **kill** command.

Running applications: BWA alignment and conversion to BAM format

Three steps (commands) are required:

```
bwa  aln  bwaindex/drosophila   short_reads.fastq   1> aln.sai    2> log

bwa   samse   bwaindex/drosophila   aln.sai   short_reads.fastq   1> aln.sam   2>> log

samtools   view  -bS  -o  aln.bam   aln.sam   2>> log
```

This is simple example of a "**pipeline**" – several commands run in succession, so that output from one command is input to the next one.

Running applications

basic shell scripting

The three commands from previous slide may be put in a text file, e.g., **bwascript.sh**, created with a text editor, which looks like this:

The script may then be executed:

chmod u+x bwascript.sh *(make the file executable; needs to be done only once, right after the script is created and saved)*

./bwascript.sh 1> script.log 2> script.err & *(run script in the background)*

Note: use “&” for the whole script rather than in each command (why?)

Running applications

basic shell scripting

Slightly more complicated (and more useful) script (call it **bwascript1.sh**)

```
#!/bin/bash

INFILE=$1

# Run alignment
bwa aln bwaindex/drosophila ${INFILE} 1> aln.sai  2> log

# Produce alignment in SAM format
bwa samse bwaindex/drosophila aln.sai ${INFILE} 1> aln.sam 2>> log

# Remove the intermediate sai file to save space
rm aln.sai

# Run samtools to generate the BAM file, name it after the input
file
samtools view -bS -o ${INFILE}_aln.bam aln.sam 2>> log

# Remove the SAM file to save space
rm aln.sam
```

This script:

- Runs the **bwa** and **samtools** commands
- Input file with reads is specified as an <u>argument</u>
- After each step, **removes** intermediate files to save space

Run the script with the following commands:

chmod u+x bwascript1.sh *(make the file executable; needs to be done only once, after the script is created)*

./bwascript1.sh short_read.fastq 1> script.log 2> script.err &

The result will be the file called **short_read.fastq_aln.bam**

Running applications

basic shell scripting

Even more complicated (and more useful) script (call it **bwascript2.sh**)

```
#!/bin/bash

INFILE=$1

# Run alignment
bwa aln bwaindex/drosophila ${INFILE} 1> aln.sai 2> log
if [ ! -e aln.sai ]; then
   echo "Alignment failed"
   exit
else
   echo "Alignment completed at"
   date
fi
# Produce alignment in SAM format
bwa samse bwaindex/drosophila  aln.sai ${INFILE} 1> aln.sam  2>> log
if [ ! -e aln.sam ]; then
    echo "Conversion to SAM format failed"
    exit
else
    echo "Conversion to SAM completed at"
    date
fi
# Remove the intermediate sai file to save space
rm aln.sai
# Run samtools to generate the BAM file, name it after the input file
samtools view -bS -o ${INFILE}_aln.bam aln.sam  2>> log
if [ ! -e ${INFILE}_aln.bam ]; then
     echo "Conversion to BAM format failed"
     exit
else
     echo "Conversion to SAM completed at"
     date
fi
# Remove the SAM file to save space
rm   aln.sam
```

This script:

- Runs the **bwa** and **samtools** commands
- Input read file specified as an argument
- Performs simple **checks of the completion status** of each command (by verifying the existence of this step's output file)
- **removes intermediate files** to save space (or terminates if a step failed)
- Prints current date and time after each step

Run the script with the following commands:

chmod u+x bwascript2.sh *(make the file executable; needs to be done only once)*

./bwascript2.sh short_read.fastq 1> script.log 2> script.err &

Similar script in perl

way too complicated – to show possibilities

```perl
#!/usr/local/bin/perl

$infile="short_reads";
$index = "bwaindex/drosophila";

# Collect all commands in an array
@commands = ();
push(@commands,"bwa aln $index  $infile.fastq  1> aln.sai  2> log");
push(@commands,"bwa samse $index  aln.sai $infile.fastq  1> aln.sam    2>> log");
push(@commands,"samtools  view  -bS  -o  aln.bam   aln.sam   2>> log");
push(@commands,"samtools sort aln.bam ${infile}_sorted");
push(@commands,"samtools index ${infile}_sorted.bam");

# Collect output files in an array
@outfiles = ();
push(@outfiles,"aln.sai");
push(@outfiles,"aln.sam");
push(@outfiles,"aln.bam");
push(@outfiles,"${infile}_sorted.bam");
push(@outfiles,"${infile}_sorted.bam.bai");

# Which file can be removed after being used
@toremove = ();
push(@toremove,1);
push(@toremove,1);
push(@toremove,1);
push(@toremove,0);
push(@toremove,0);

# Now we can use a loop to do the processing

for($i=0;$i<=$#commands;$i++)
{
        # Run the command
        system($commands[$i]);
        # validate and remove intermediate files
        validate($i);
}
```

```perl
# Result validation function - will delete the intermediate file
# if required, or exit if error detected
#
sub validate
{
        ($loopind) = @_;
        if( -e $outfiles[$loopind] && -s  $outfiles[$loopind])
        {
                print "Command\n\n $commands[$loopind]\n\n completed\n";
                if($loopind >0)
                {
                        if($toremove[$loopind-1])
                        {
                                print "Removing intermediate file $outfiles[$loopind-1]\n";
                                `rm -f $outfiles[$loopind-1]`;
                        }
                }
        }
        else
        {
                print "Something wrong with command\n\n $commands[$loopind]\n\n Exiting....\n";
                exit;
        }
}
```

Perl

Pros:

- Available on all platforms
- Rich but intuitive syntax
  - Each task can be programmed in many different, equivalent ways
- Arrays, loops, conditional statements, etc. – full programming infrastructure
- Access to operating system commands and programs (through **system** function)
- Functions
  - write your own
  - Use extensive library of perl modules written by others (http://www.cpan.org)
    - BioPerl http://www.bioperl.org/wiki/Main_Page - also a list of resources for learning perl
- Efficient for text parsing/processing (not shown in our examples)

Cons:

- Perl is an **interpreted** language – code 10-100 times slower than **compiled** languages (C, C++, Java)
- Syntax flexibility may lead to confusion and hard-to-find bugs
- Poor memory management (i.e., objects take much more memory than really needed; some memory leaks)

More about scripting

<u>Multiple scripting tools available</u>

- **shell** (bash, tcsh – good for stitching together shell commands)
- **perl** (probably the most popular in biology, due to BioPerl module package)
- **python** (good numerical analysis tools – NumPy, SciPy packages)
- **awk** (mostly text parsing and processing)
- **sed** (mostly text parsing and processing)
- **R** (rich library of numerical analysis and statistical functions)

A separate course on Perl scripting is planned in the fall.