Museu de Zoologia da USP

MZUSP01: User Manual

Denis Jacob Machado

September 12, 2018

HIGH PERFORMANCE COMPUTING AT THE MUSEUM OF ZOOLOGY

User Guide.

September 12, 2018

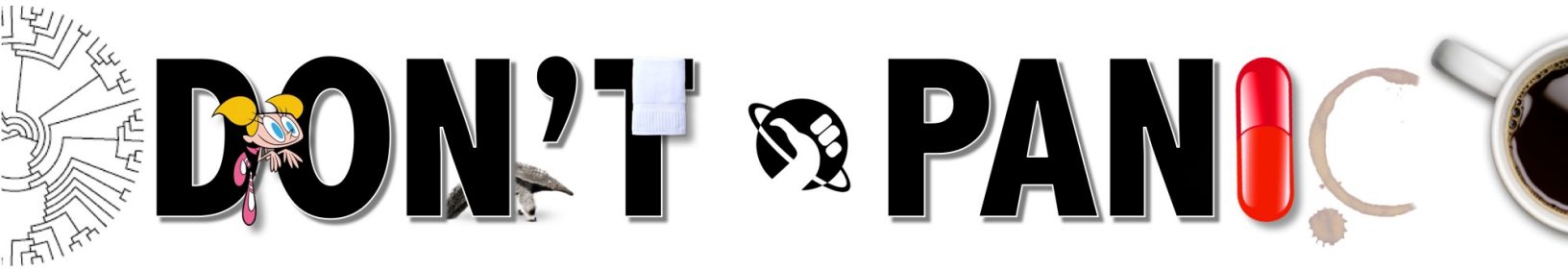
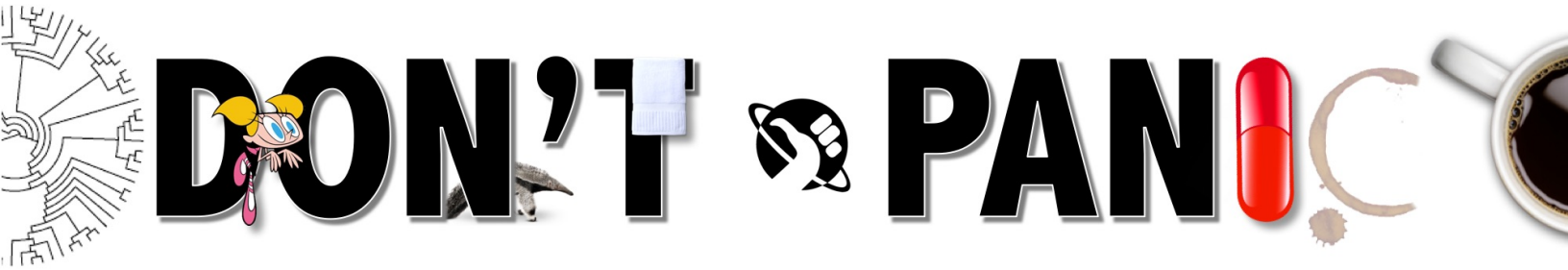


Table of contents

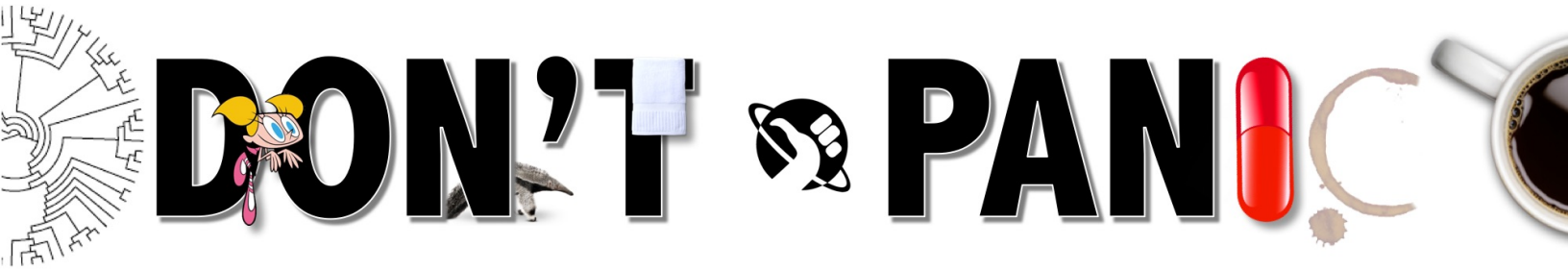


1. Computational resources

High performance computing (HPC) has become an integral aspect of modern comparative biology. The dedicated HPC cluster "MZUPS01" housed in the Museu de Zoologia da USP is essential research tools that enable us to carry out large comparative genomic and phylogenetic analyses.

1.1 MZUSP01

MZUSP01 is a FAPESP-funded SGI cluster housed in the Museu de Zoologia da USP that entered production in October 2013. It is composed of 11 quad-socket AMD Opteron 6376 16-core 2.3-GHz CPU, 16MB cache, 6.4 GT/s compute nodes (= 704 cores total) with 128GB RAM DDR3 1600 MHz (16 x 8GB) and QDR 4x InfiniBand (32 Gb/s) networking.



2. Basics

2.1 Access information

The headnode of MZUSP01 is only accessible via the Gatekeeper. To get to it, be sure the adminstrator gave you a user account and password information not just to MZUSP01, but to the Gatekeeper as well. Them use your account information to log into the Gatekeeper using the Secure Shell (SSH) like this

```
$ ssh <username>@143.107.38.43
```

Note that the dollar sign ($) simply indicates that you should execute this command as user, not as root. You should not type it. Also note that you are not to type "<username>." The greater-than and

smaller-than signs are simply enclosing a variable named “username” so you know you should put your own username there.

After typing the command above on your terminal Window, enter your password. Remember that nothing will show on the screen when you type the password. Press return/ enter.

From the Gatekeeper, you can get to MZUSP01 in a similar way:

```
$ ssh <username>@mzusp01
```

Enter your account’s password and done!

PLEASE, PLEASE, PLEASE!

At some point after getting in, you will want to leave. Please type `exit` to do that and try to avoid simply closing the terminal window or shutting down your computer or any of the reckless stuff you kids do nowadays.

Oh, before I forget... I don’t know who you are. I don’t know what you want. If you’re looking for money, remember to share it when you find it. But what I do have are a very particular set of skills. Skills that involve mainly making coffee but that is hardly the point. If you follow security protocol and play by the rules, that will be the end of it - I will not look for you, I will not pursue you... but if you don’t, I will look for you, I will find you... and I will be very disappointed!

2.2 Data transfer

The easiest way to upload and download data from MZUSP01 and Gatekeeper is using `scp`. The basic syntax of `scp` is the following:

```
$ scp <files> <username>@<host address>:</path/to/directory>
```

Remember that, for most applications, you can use the `help` argument to get more information about how to use that program:

```
$ scp --help
```

If you need to resume an `scp` transfer from local to remote, try with `rsync`.

In general, the order of arguments for rsync is the following:

```
rsync [options] SRC DEST
```

For example:

```
$ rsync --partial --progress --rsh=ssh <local_file> <user>@<host>:<remote_file>
```

You may also try this short version, if you are lazy:

```
$ rsync -P -e ssh local_file user@host:remote_file
```

2.3 Basic administrative operations

You are just an user and we sincerely hope you'll not break anything. Still, you might need to know a few things regarding the cluster and how it works. The following are a few common operations that just might save the day.

2.3.1 Check temperature of a given node

The CPUs have sensors and sometimes it is not a bad idea to check, specially after a power outage that might have turned off the air conditioners.

```
$ ipmitool -H <vnode>-bmc -U admin -P admin sensor
```

Note that vnode stands for n001 to n011 and correspond to the 11 compute nodes in MZUSP01.

2.3.2 See if vnodes are energized

Now, here is something you will use VERY often. Use the following commands to check which nodes are energized:

```
$ pm -q
```

Note that energized nods are not necessarily online and accessible. Also, if you are a user, don't do switching node on and off. Talk to the administrator for that.

2.3.3 Check if vnodes are accessible

The command above lists compute nodes that are on, off, or inaccessible. The following command, however, tell you if they are online:

```
$ pdsh -a w | sort -n
```

If everything is ok, you should see this after executing the command above

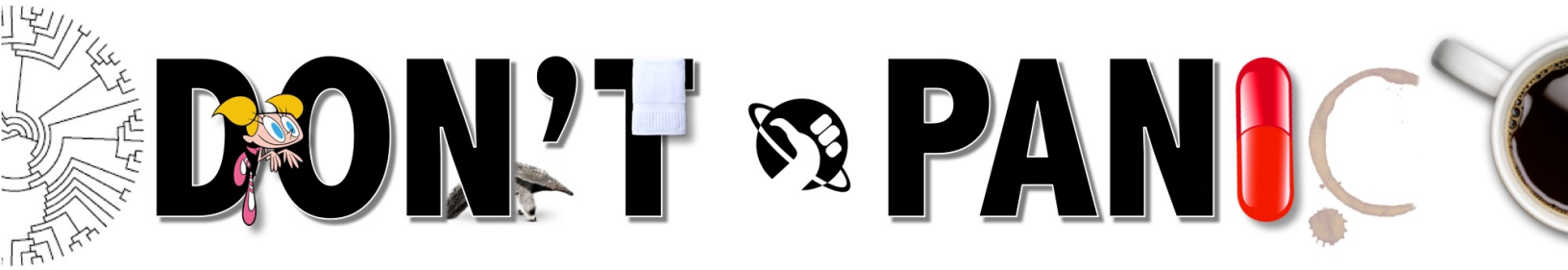
```
n001:  06:45:33 up 14 days, 23:37,  0 users,  load average: 0.00, 0.01, 0.05
n001: USER     TTY        LOGIN@   IDLE   JCPU   PCPU WHAT
n002:  06:46:17 up 14 days, 23:37,  0 users,  load average: 0.00, 0.01, 0.05
n002: USER     TTY        LOGIN@   IDLE   JCPU   PCPU WHAT
n003:  06:45:33 up 14 days, 23:37,  0 users,  load average: 0.00, 0.01, 0.05
n003: USER     TTY        LOGIN@   IDLE   JCPU   PCPU WHAT
n004:  06:45:33 up 14 days, 23:36,  0 users,  load average: 0.00, 0.01, 0.05
n004: USER     TTY        LOGIN@   IDLE   JCPU   PCPU WHAT
n005:  06:45:33 up 14 days, 23:37,  0 users,  load average: 0.00, 0.01, 0.05
n005: USER     TTY        LOGIN@   IDLE   JCPU   PCPU WHAT
n006:  06:45:33 up 14 days, 23:37,  0 users,  load average: 0.00, 0.01, 0.05
n006: USER     TTY        LOGIN@   IDLE   JCPU   PCPU WHAT
n007:  06:45:33 up 14 days, 23:37,  0 users,  load average: 0.01, 0.03, 0.05
n007: USER     TTY        LOGIN@   IDLE   JCPU   PCPU WHAT
n008:  06:45:33 up 14 days, 23:37,  0 users,  load average: 0.00, 0.01, 0.05
n008: USER     TTY        LOGIN@   IDLE   JCPU   PCPU WHAT
n009:  06:45:33 up 14 days, 23:37,  0 users,  load average: 0.01, 0.02, 0.05
n009: USER     TTY        LOGIN@   IDLE   JCPU   PCPU WHAT
n010:  06:46:28 up 14 days, 23:38,  0 users,  load average: 0.01, 0.02, 0.05
n010: USER     TTY        LOGIN@   IDLE   JCPU   PCPU WHAT
n011:  06:45:33 up 14 days, 23:37,  0 users,  load average: 0.01, 0.02, 0.05
n011: USER     TTY        LOGIN@   IDLE   JCPU   PCPU WHAT
```

2.3.4 List compute nodes that have potential problems

A final check that you can do before executing jobs is to try to see if there are any obvious problems on compute nodes that the PBS server is aware off (not that you should trust it blindly, machines can be a pain):

```
$ pbsnodes -l
```

If the PBS server can't see no evil, the command above will return nothing at all.



3. PBS

3.1 Templates

The **Portable Batch System** (or simply "PBS") is the name of computer software that performs job scheduling. You will ALWAYS use it to execute tasks on MZUSP01 unless you are told otherwise by the adminstrator.

The primary task of PBS is to allocate computational tasks, *i.e.*, batch jobs, among the available computing resources. PBS is often used in conjunction with UNIX cluster environments.

The `/home/sgi/JOB_EXAMPLE` has an example PBS script to test if PBS is working properly. There are copies of this directory in the home directory of each user (ask a copy for the adminstrator if you don't have it).

General template

```
#!/bin/sh

# MZUSP01

# PBS VARIABLES

#PBS -V
#PBS -N <JOB NAME>
#PBS -e <STANDARD ERROR FILE>
#PBS -o <STANDARD OUTPUT FILE>
#PBS -r n
#PBS -q <regular|workq>
#PBS -l place=<free|pack|scatter|vscatter>
#PBS -l select=<1-11>:ncpus=<1-64>:mpiprocs=<1-64>:mem=<gb>
#PBS -l host=<n001-n011>
#PBS -l walltime=100:00:00
# FABRIC=<shm|sock|ssm|rdma|rdssm>
FABRIC=rdma
CORES=$[ `cat $PBS_NODEFILE | wc -l` ]
NODES=$[ `uniq $PBS_NODEFILE | wc -l` ]
cd $PBS_O_WORKDIR

# LOG AND MODULES

. /etc/profile.d/modules.sh
module load mpt && echo "Successfully load modules"
printf "mpiexec_mpt run command location is: `which mpiexec_mpt`\n";
printf "\n[START] qstat -f $PBS_JOBID\n"
printf "`qstat -f $PBS_JOBID`\n[END]\n"
TBEGIN=`echo "print time();" | perl`

# COMMAND EXAMPLES

mpiexec_mpt -n $CORES <program> <arguments> > <stdout> 2> <stderr>

# FINISH

TEND=`echo "print time();" | perl`
printf "Job finished: `date`\n";
printf "Job walltime: `expr $TEND - $TBEGIN`\n";
```

Note that the file after the ">" on the example command line will receive the standard output while the file after "2>" will receive the standard error. Specifying these files is optional.

Parallel execution of POY using mpirun

```
#!/bin/sh

# MZUSP01

#PBS -V
#PBS -N template1
#PBS -e template1.err
#PBS -o template1.out
#PBS -q workq
#PBS -l place=scatter
#PBS -l select=11:ncpus=64:mpiprocs=64
#PBS -l walltime=100:00:00

# shm, sock, ssm, rdma, rdssm
FABRIC=rdma

CORES=$[ `cat $PBS_NODEFILE | wc -l` ]
NODES=$[ `uniq $PBS_NODEFILE | wc -l` ]

cd $PBS_O_WORKDIR

. /etc/profile.d/modules.sh

module load mpt

TBEGIN=`echo "print time();" | perl`

MPI_HOSTS=$(sort $PBS_NODEFILE | uniq -c | \
        awk '{print $2 " " $1}' | \
    tr "\n" "," | \
    sed 's/.$//')

mpirun  $MPI_HOSTS /apps/poy -e script.poy > stdout 2> stderr
```

Check the available installations of POY on /apps/. This directory is available to all compute nodes at it houses most applications that have been installed by the adminstrator for all users.

Parallel execution of POY using mpiexec_mpt

```
#!/bin/sh

#PBS -N template2
#PBS -e template2.err
#PBS -o template2.out
#PBS -q workq
#PBS -l place=scatter
#PBS -l select=1:ncpus=1:mpiprocs=1
# #PBS -l select=6:ncpus=64:mpiprocs=64
# #PBS -l select=11:ncpus=64:mpiprocs=64

# shm, sock, ssm, rdma, rdssm
FABRIC=rdma

CORES=$[ `cat $PBS_NODEFILE | wc -l` ]
NODES=$[ `uniq $PBS_NODEFILE | wc -l` ]

cd $PBS_O_WORKDIR

. /etc/profile.d/modules.sh

module load mpt

printf "mpiexec_mpt location is: `which mpiexec_mpt`\n";

mpiexec_mpt -n $CORES \
        /apps/poy511 \
            -e scriptSearch.3.0.poy > stdout 2> stderr
```

This alternative template uses `mpiexec_mpt` instead of `mpirun`.

3.2 Queues

For now, users of MZUSP01 have the following two queues available to them:

- `regular`: default queue, point towards `exec_regular` and ives access to all resources (n[001–011])
- `workq`: same as above, but with a different named. Kept due historical reasons.

3.3 Submit a job

Use `qsub` to submit a job:

```
$ qsub </path/to/pbs.script>
```

3.4 After job submission

3.4.1 Check the status of a job

To see if job is in the queue (**Q**), running (**R**), hanging (**H**), or exiting (**E**):

```
$ qstat <job number>
```

3.4.2 Check job details

```
$ qstat -f <job number>
```

This will output many details that you might be unfamiliar with. Don't worry, simply send the output to the administrator if you have any problems.

3.4.3 Kill a job

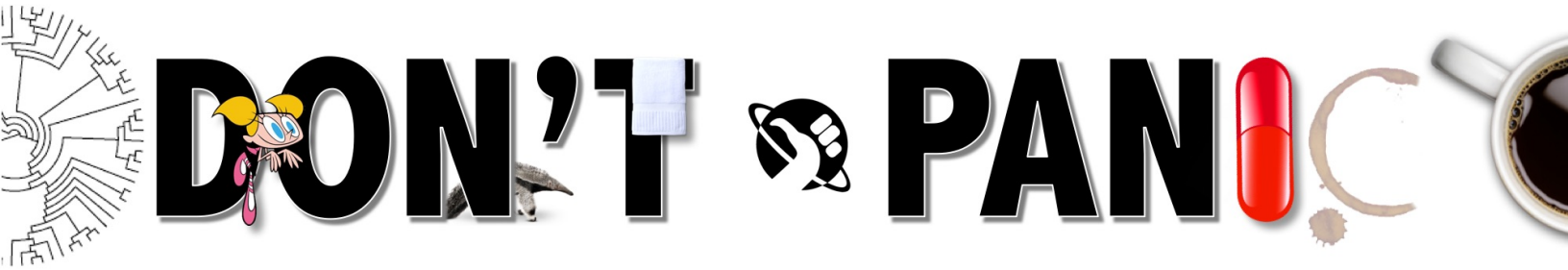
This will only work for jobs that your user have submitted (hopefully). But, just in case, keep clear of other user's jobs.

```
$ qdel <job number>
```

3.4.4 Count how many jobs a user is running

Not sure if you will ever need this, but it is so fun I wanted to include it.

```
$ qstat | grep <user name> | grep R | wc -l
```



4. Tips for software installation

The ~ symbol indicates the path to your home directory. Nope, not your secrete bunker at 10 Cloverfield Lane. Not the 221B Baker Street neither, you rich scum! Stop buying property with your CAPES' scholarship and try donating some of that money to charity, you heartless maniac!

Your home directory is /home/<username>. Therefore, ~ = `/home/<username>`. That is were you should install software, mainly through the usage of source codes inside tarballs.

Installing programs locally using their source codes is painful to say the least. Here are a few tips:

- Arora, H. (2012) "How to Compile and Install Software from Source Code on Linux." Retrieved from https://www.thegeekstuff.com/2012/06/install-from-source/. Last accessed on September 12, 2018.
- Hoffman, C. (2012) "How To Compile and Install from Source on Ubuntu." Retrieved from https://www.howtogeek.com/105413/how-to-compile-and-install-from-source-on-ubuntu/. Last accessed on September 12, 2018.
- Brocklehurst, G. (2015) "The magic behind configure, make, make install ." Retrieved from https://robots.thoughtbot.com/the-magic-behind-configure-make-make-install. Last accessed on September 12, 2018.
- SathiyaMoorthy (2010) "The Ultimate Tar Command Tutorial with 10 Practical Examples." Retrieved from https://www.thegeekstuff.com/2010/04/unix-tar-command-examples/. Last accessed on September 12, 2018.

4.1 The just of it

If you are luck, this step-by-step procedure may be all you need.

Fist of all, get the URL for the tarball you want and download it using wget, like so:

```
$ wget <link to the tarball>
```

Next you needs to unpack the tarball in order to get access to the source code and other files. Depending on the extension, use one of the following commands:

```
$ tar -xvfz <name of tarball with .tar.gz extension>
```

...or...

```
$ tar -xvfj <name of tarball with tar.bz2 extension>
```

If everything worked out until here, taxonomists would say you are a *Lukius bastardus* (Lucky: Son: Of: A: *Canis canis* ♀)[1]. Seriously though, prepare for things to do not work 'cause they often don't.

As I was saying, if everything worked out until here, you want to move into the extracted directory and compile and install the software, like this:

```
$ ./compile --prefix=<path somewhere inside your home directory>
$ make
$ make install
```

4.2 Add new directories to the system PATH

The `PATH` is an environment variable on Unix-like operating systems, DOS, OS/2, and Microsoft Windows, specifying a set of directories where executable programs are located. In general, each executing process or user session has its own PATH setting.

Imagine that we want to add `/home/username/bin` to the `PATH`.

```
export PATH="${PATH}:/home/username/bin"
export PATH="/home/username/bin:${PATH}"
```

Choosing among the two options above depends on whether you want to add `/home/username/bin` at the end (to be searched after all other directories, in case there is a program by the same name in multiple directories) or at the beginning (to be searched before all other directories). You can also add multiple entries at the same time, for exeample:

```
export PATH="${PATH}:home/username/bin:~/opt/bin:~/opt/node/bin"
```

Please note that you don't need export if the variable is already in the environment: any change of the value of the variable is reflected in the environment

4.2.1 Where to put it?

Ideally, you should not define environment variables in `~/.bashrc`. The right place to define environment variables such as `PATH` is `~/.profile` (or `~/.bash_profile` if you don't care about shells other than Bash).

[1] Ha! Ha! And you though that bioinformaticians knew nothing about taxonomy! In your face!

4.3 How to set library paths

New paths can be added into `/etc/bash.bashrc.local`, which will be read upon reboot. You can also export the path directly just for the current section. For example:

```
$ export LD_LIBRARY_PATH=.:/usr/lib/:/usr/lib64/:/usr/local/lib/:/usr/local/lib64/:
    /usr/local/lib/ncl/:/usr/local/boost/:/usr/lib64/gcc/x86_64-suse-linux/4.3/:
    /opt/sgi/mpt/mpt-2.07/lib/:/apps/GSL/lib/:/apps/GSL/include/gsl/:
    ${LD_LIBRARY_PATH}
```

To check the variable `LD_LIBRARY_PATH` execute:

```
$ echo $LD_LIBRARY_PATH
```

To unset a `LD_LIBRARY_PATH` execute:

```
$ unset LD_LIBRARY_PATH
```

In some cases you may also need to add the library so it will be recognized by the `ldconfig` command. In order to do that, creat a text file in `/etc/bash/bashrc.local.d/` with some name and the `.conf` extension. Inside it, paste just the path of the library folder. Refresh `ldcongid` by executing this:

```
$ sudo /sbin/ldconfig
```

Check if the new libraries are there:

```
$ sudo /sbin/ldconfig -v
```

Do this before compiling programs that requires new libraries.
TIP: you can see if the executable nows where everything he need is by executing `ldd`:

```
$ ldd <executable>
```

But even if it finds everything, sometimes it will not work if that library was in a declared path that was not recognized by `ldconfig` before compilation.

4.4 Your friend ldd

The `ldd` command can be used to find a program's shared libraries, which is very useful if you are suspicious that the program is not working properly due to problems with its dependencies or compilation errors.

4.4.1 The ldd Command Syntax

This is the proper syntax when using the `ldd` command:

```
ldd [OPTION]... FILE...
```

Here are the available `ldd` command switches that can be inserted into the `[OPTION]` spot in the above command:

```
  --help print this help and exit
  --version print version information and exit
  -d, --data-relocs process data relocations
  -r, --function-relocs process data and function relocations
  -u, --unused print unused direct dependencies
  -v, --verbose print all information
```

How to Use the `ldd` Command

You can use the following command to get more information from any `ldd` command:

```
$ ldd -v /path/to/program/executable
```

The output shows version information as well as the paths and addresses to the shared libraries, like this:

```
ldd libshared.so
linux-vdso.so.1 => (0x00007fff26ac8000)
libc.so.6 => /lib/libc.so.6 0x00007ff1df55a000)
/lib64/ld-linux-x86-64.so.2 (0x00007ff1dfafe000)
```

If the SO file doesn't exist at all, you can find the missing libraries using the following command:

```
$ ldd -d path/to/program
```

The output is similar to the following:

```
linux-vdso.so.1 (0x00007ffc2936b000)
/home/gary/demo/garylib.so => not foundlibc.so.6 => usr/lib/libc.so.6 (0x00007fd0c6259000)
/lib64/ld-linux-x86-64.so.2 (0x00007fd0c65fd000)
```

Important: Never run `ldd` against an untrusted program since the command might actually execute it. This is a safer alternative that shows just the direct dependencies and not the whole dependency tree: `objdump -p /path/to/program | grep NEEDED`.

4.4.2 How to Find the Path to an Application

You have to provide the full path to an application if you want to find its dependencies with `ldd`, which you can do a number of ways.

For example, this is how you'd find the path to `Firefox` (this is just an example, there is no Firefox in the cluster):

```
$ find / -name firefox
```

The problem with the find command, however, is that it will not only list the executable but everywhere that Firefox is located, like this:

```
/etc/skel/.mozilla/firefox
/home//cache/mozilla/firefox
/home//.mozilla/firefox
/usr/bin/Firefox
/usr/lib/Firefox
/usr/lib/Firefox/Firefox
```

This approach is a bit of an overkill and you may need to use the `sudo` command to elevate your privileges, else you're likely to get lots of permission denied errors.

It's instead much easier to use the `whereis` command to find an application's path:

```
$ whereis firefox
```

This time the output might look like this:

```
/usr/bin/firefox
/etc/firefox
/usr/lib/firefox
```

All you have to do now to find the shared libraries for Firefox is type the following command:

```
$ ldd /usr/bin/firefox
```

The output from the command will be something like this:

```
linux-vdso.so.1 (0x00007ffff8364000)
libpthread.so.0 => /usr/lib/libpthread.so.0 (0x00007feb9917a000)
libdl.so.2 => /usr/lib/libdl.so.2 (0x00007feb98f76000)
libstdc++.so.6 => /usr/lib/libstdc++.so.6 (0x00007feb98bf4000)
libm.so.6 => /usr/lib/libm.so.6 (0x00007feb988f6000)
libgcc_s.so.1 => /usr/lib/libgcc_s.so.1 (0x00007feb986e0000)
libc.so.6 => /usr/lib/libc.so.6 (0x00007feb9833c000)
/lib64/ld-linux-x86-64.so.2 (0x00007feb99397000)
```

The `linux-vdso.so.1` is the name of the library and the hex number is the address where the library will be loaded to in memory.

You'll notice on many of the other lines that the right arrow ($\Rightarrow$) symbol is followed by a path. This is the path to the physical binary; the hex number is the address where the library will be loaded.

Table 4.1: Relation of text values to octal and binary representation

Text Values	Octal Values	Binaries Equivalents	Description of the Privileges
—	0	000	No privileges set
–x	1	001	Executionbitisset
-w-	2	010	Writebitisset
-wx	3	011	Writeandexecution
r–	4	100	Readonlyaccess
r-x	5	101	Readandexecutionpossible
rw-	6	110	Readandwritepossible
rwx	7	111	Read,writeandexecutionpossible

4.5 Your friend chmod

A file's permissions can be changed by using the change mode command `chmod`. The mode can be changed by changing the character flags (*e.g.* **r**, **w**, **x** etc.) or by using an octal number. The octal format uses the digits 0 to 7 to set the permissions for each set: **owner**, **group**, **others**.

Each octal value translates to a 3-bit binary equivalent where each bit either sets or unsets the permission flags. The following table shows how the text values relate to the octal and binary representations:

File permissions

File permissions are represented by positions two through ten of the `ls ?l` display. The nine character positions consist of three groups of three characters. Each three character group indicates read (**r**), write (**w**), and execute (**x**) permissions.

The three groups indicate permissions for the **owner**, **group**, and **other users** respectively.

4.5.1 Add single permission to a file/directory

Changing permission to a single set. + symbol means adding permission. For example, do the following to give execute permission for the user irrespective of anything else:

```
$ chmod u+x filename
```

4.5.2 Add multiple permission to a file/directory

Use comma to separate the multiple permission sets as shown below.

```
$ chmod u+r,g+x filename
```

4.5.3 Remove permission from a file/directory

Following example removes read and write permission for the user.

```
$ chmod u-rx filename
```

4.5.4 Change permission for all roles on a file/directory

Following example assigns execute privilege to user, group and others (basically anybody can execute this file).

```
$ chmod a+x filename
```

4.5.5 Make permission for a file same as another file (using reference)

If you want to change a file permission same as another file, use the reference option as shown below. In this example, file2's permission will be set exactly same as file1's permission.

```
$ chmod --reference=file1 file2
```

4.5.6 Apply the permission to all the files under a directory recursively

Use option -R to change the permission recursively as shown below.

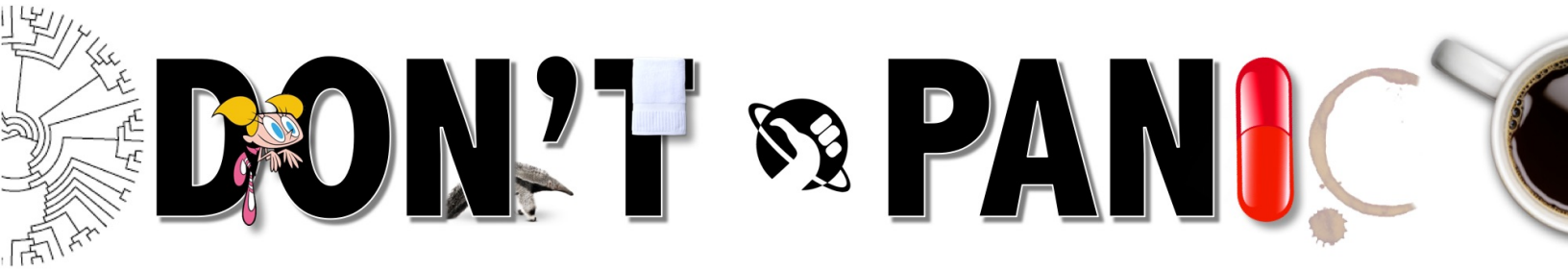
```
$ chmod -R 755 directory-name/
```

4.5.7 Change execute permission only on the directories (files are not affected)

On a particular directory if you have multiple sub-directories and files, the following command will assign execute permission only to all the sub-directories in the current directory (not the files in the current directory).

```
$ chmod u+X *
```

Note: If the files has execute permission already for either the group or others, the above command will assign the execute permission to the user



5. Bioinformatics tips

5.1 Useful Bash one-liners

Sum the values on the first column of `file.txt`:

```
$ awk '{sum+=$1} END {print sum}' file.txt
```

Number each line in file.txt:

```
$ sed = file.txt | sed 'N;s/\n/ /'
```

Get unique entries on the first column of `file.txt` (takes only the first instance):

```
$ awk '!arr[$2]++' file.txt
```

Print each line where the 5^{th} field is equal to "abc123":

```
$ awk '$5 == "abc123"' file.txt
```

Print each line where the 5^{th} field is not equal to "abc123":

```
$ awk '$5 == "abc123"' file.txt
```

Print each line whose 7^{th} field matches the regular expression:

```
$ awk '$7 ~ /^[a-f]/' file.txt
```

Print each line whose 7^{th} field does not match the regular expression:

```
$ awk '$7 ~ /^[a-f]/' file.txt
```

Replace all occurances of "foo" with bar in `file.txt`:

```
$ sed 's/foo/bar/g' file.txt
```

Convert a FASTQ file (`file.fq`) to FASTA (`file.fa`):

```
$ sed -n '1~4s/^@/>/p;2~4p' file.fq > file.fa
```

Extract every 4^{th} line starting at the second line of a FASTQ file (`file.fq`):

```
$ sed -n '2~4p' file.fq
```

Print total number of reads, total number unique reads, percentage of unique reads, most abundant sequence, its frequency, and percentage of total in `file.fq`:

```
$ cat myfile.fq | awk '((NR-2)%4==0){read=$1;total++;count[read]++}
  END{for(read in count){if(!max||count[read]>max)
  {max=count[read];axRead=read};if(count[read]==1){unique++}};
  printtotal,unique,unique*100/total,maxRead,count[maxRead],
  count[maxRead]*100/total}'
```

Convert `file.bam` back to `file.fastq`:

```
$ samtools view file.bam | awk 'BEGIN {FS="\t"}
  {print "@" $1 "\n"$10 "\n+\n" $11}' > file.fq
```

Keep only top bit scores in Blast hits (best bit score only):

```
$ awk '{ if(!x[$1]++) {print $0; bitscore=($14-1)}
  else{ if($14>bitscore) print $0} }' blastout.txt
```

Keep only top bit scores in blast hits (5 less than the top):

```
$ awk '{ if(!x[$1]++) {print $0; bitscore=($14-6)}
  else{ if($14>bitscore) print $0} }' blastout.txt
```

Trim leading whitespace in `file.txt`:

```
$ sed 's/^[ \t]*//' file.txt
```

Trim trailing whitespace in `file.txt`:

```
$ sed 's/[ \t]*$//' file.txt
```

Trim leading and trailing whitespace in `file.txt`:

```
$ sed 's/^[ \t]*//;s/[ \t]*$//' file.txt
```

Delete blank lines in `file.txt`:

```
$ sed '/^$/d' file.txt
```

Count the number of unique lines in `file.txt`:

```
$ cat file.txt | sort | uniq | wc -l
```

Find number of lines shared by two files:

```
$ sort file1 file2 | uniq -d
```

Find the most common strings in the second column:

```
$ cut -f2 file.txt | sort | uniq -c | sort -k1nr | head
```

Pick 10 random lines from `file.txt`:

```
$ shuf file.txt | head -n 10
```

Print rows where column 3 is larger than column 5 in `file.txt`:

```
$ awk '$3>$5' file.txt
```

Compute the mean of column 2:

```
$ awk '{x+=$2}END{print x/NR}' file.txt
```

Extract fields 2, 4, and 5 from `file.txt`:

```
$ awk '{print $2,$4,$5}' input.txt
```

Print all possible 3-mer DNA sequence combinations:

```
$ echo {A,C,T,G}{A,C,T,G}{A,C,T,G}
```

Untangle an interleaved paired-end FASTQ file (`interleaved.fq`) If a FASTQ file has paired-end reads intermingled, and you want to separate them into separate "/1" and "/2" files, and assuming the /1 reads precede the /2 reads:

```
$ cat interleaved.fq |paste - - - - - - - - | \
  tee >(cut -f 1-4 | tr "\t" "\n" > deinterleaved_1.fq) \
  | cut -f 5-8 | tr "\t" "\n" > deinterleaved_2.fq
```

Untangle an interleaved paired-end FASTQ file. If a FASTQ file has paired-end reads intermingled, and you want to separate them into separate /1 and /2 files, and assuming the /1 reads precede the /2 reads:

```
$ seqtk seq -l0 interleaved.fq | awk '{if ((NR-1) % 8 < 4) print \
  >> "deinterleaved_1.fq"; else print >> "deinterleaved_2.fq"}'
```

Search for files in BAM format (`.bam`) anywhere in the current directory recursively:

```
$ find . -name "*.bam"
```

Delete all `.bam` files:

```
$ find . -name "*.bam" | xargs rm
```

Rename all `.txt` files to `.bak` (backup `*.txt` before doing something else to them, for example):

```
$ find . -name "*.txt" | sed "s/\.txt$//" | \
  xargs -i echo mv {}.txt {}.bak | sh
```

Chastity filter raw Illumina data (`grep` reads containing ":N:", append (`-A`) the three lines after the match containing the sequence and quality info, and write a new filtered FASTQ file):

```
$ find *fq | parallel "cat {} | \
  grep -A 3 '^@.*[^:]*:N:[^:]*:' | \
  grep -v '^\-\-$' > {}.filt.fq"
```

Run FASTQC in parallel 12 jobs at a time:

```
$ find *.fq | parallel -j 12 "fastqc {} --outdir ."
```

Index your bam files in parallel, but only echo the commands (`-dry-run`) rather than actually running them:

```
$ find *.bam | parallel --dry-run 'samtools index {}' seqtk
```

Convert a file from FASTQ to FASTA format:

```
$ seqtk seq -a in.fq.gz > out.fa
```

Convert ILLUMINA 1.3+ FASTQ to FASTA and mask bases with quality lower than 20 to lowercases (the 1^{st} command line) or to "N" (the 2^{nd}):

```
$ seqtk seq -aQ64 -q20 in.fq > out.fa
$ seqtk seq -aQ64 -q20 -n N in.fq > out.fa
```

Fold long FASTA/Q lines and remove FASTA/Q comments:

```
$ seqtk seq -C160 in.fa > out.fa
```

Convert multi-line FASTQ to 4-line FASTQ:

```
$ seqtk seq -l0 in.fq > out.fq
```

Reverse complement FASTA/Q:

```
$ seqtk seq -r in.fq > out.fq
```

Extract sequences with names in file `name.lst`, one sequence name per line:

```
$ seqtk subseq in.fq name.lst > out.fq
```

Extract sequences in regions contained in file `reg.bed`:

```
$ seqtk subseq in.fa reg.bed > out.fa
```

Mask regions in `reg.bed` to lowercases:

```
$ seqtk seq -M reg.bed in.fa > out.fa
```

Subsample 10000 read pairs from two large paired FASTQ files (remember to use the same random seed to keep pairing):

```
$ seqtk sample -s100 read1.fq 10000 > sub1.fq
$ seqtk sample -s100 read2.fq 10000 > sub2.fq
```

Trim low-quality bases from both ends using the Phred algorithm: s

```
$ eqtk trimfq in.fq > out.fq
```

Trim 5bp from the left end of each read and 10bp from the right end:

```
$ seqtk trimfq -b 5 -e 10 in.fa > out.fa
```

Print all sequences annotated in a GFF3 file.

```
$ cut -s -f 1,9 yourannots.gff3 | grep $'\t' | cut -f 1 | sort | uniq
```

Determine all feature types annotated in a GFF3 file:

```
$ grep -v '^#' yourannots.gff3 | cut -s -f 3 | sort | uniq
```

Determine the number of genes annotated in a GFF3 file:

```
$ grep -c $'\tgene\t' yourannots.gff3
```

Extract all gene IDs from a GFF3 file:

```
$ grep $'\tgene\t' yourannots.gff3 | \
  perl -ne '/ID=([^;]+)/ and printf("%s\n", $1)'
```

Print length of each gene in a GFF3 file:

```
$ grep $'\tgene\t' yourannots.gff3 | cut -s -f 4,5 | \
  perl -ne '@v = split(/\t/); printf("%d\n", $v[1] - $v[0] + 1)'
```

FASTA header lines to GFF format (assuming the length is in the header as an appended "_length" as in Velvet assembled transcripts):

```
$ grep '>' file.fasta | awk -F "_" \
  'BEGIN{i=1; print "##gff-version3"}
  { print $0"\tBLAT\tEXON\t1\t"$10"\t95\t+\t.\tgene_id="$0";
  transcript_id=Transcript_"i;i++ }' > file.gff
```

5.2 Customize your terminal prompt

Get a prompt that looks like `user@hostname:/full/path/cwd/:$`, add the following line to `~/.bashrc`:

```
export PS1="\u@\h:\w\\$ "
```

Now, source your `.bashrc` file:

```
$ source ~/.bashrc
```

5.3 Aliases

Here are a few useful aliases for your `.bashrc`. First, how about never typing cd ../../.. again? Add the following lines to `.bashrc`:

```
alias ..='cd ..'
alias ...='cd ../../'
alias ....='cd ../../../'
alias .....='cd ../../../../'
alias ......='cd ../../../../../'
```

Some `ls` aliases:

```
alias ls="ls -1p --color=auto"
alias l="ls -lhGgo"
alias ll="ls -lh"
alias la="ls -lhGgoA"
alias lt="ls -lhGgotr"
alias lS="ls -lhGgoSr"
alias l.="ls -lhGgod .*"
alias lhead="ls -lhGgo | head"
alias ltail="ls -lhGgo | tail"
alias lmore='ls -lhGgo | more'
```

Aliases to ssk before removing or overwriting files:

```
alias mv="mv -i"
alias cp="cp -i"
alias rm="rm -i"
```

Aliases to use cut on space- or comma- delimited files:

```
alias cuts="cut -d \" \""
alias cutc="cut -d \",\""
```

Aliases to pack and unpack `tar.gz` files:

```
alias tarup="tar -zcf"
alias tardown="tar -zxf"
```

Aliases to use `mcd` to create a directory and `cd` to it simultaneously:

```
function mcd { mkdir -p "$1" && cd "$1";}
```

Aliases to go up to the parent directory and list it's contents:

```
alias u="cd ..;ls"
```

Aliases to make `grep` pretty:

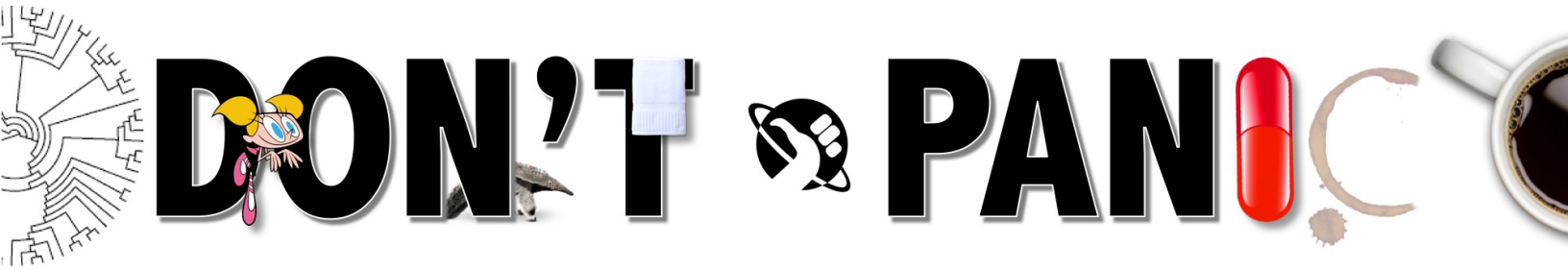
```
alias grep="grep --color=auto"
```

Common typos:

```
alias mf="mv -i"
alias mroe="more"
```

All the aliases above go into your `.bashrc` and you can even include an alias to refresh (`source`) it:

```
alias refresh="source ~/.bashrc"
```



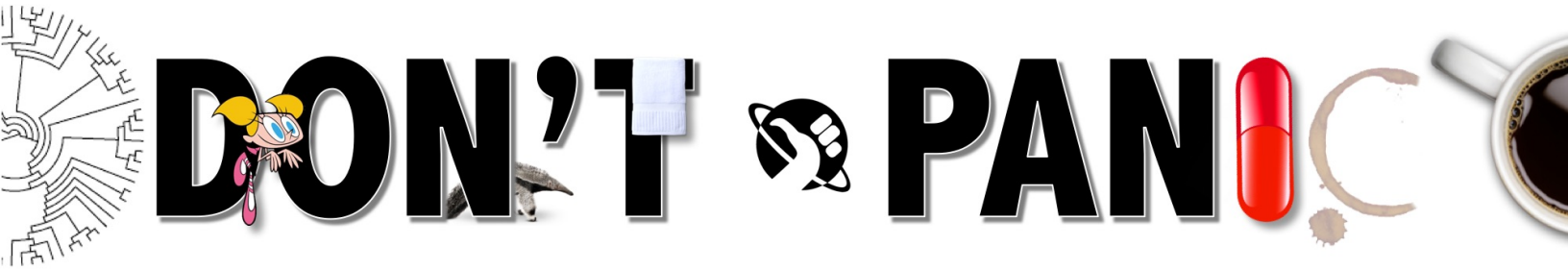
6. Contact information

6.1 Talk to Denis (and be prepared not to blame him of anything)

- Name: Dr. Denis Jacob Machado
- Email: machadodj@alumni.usp.br
- Alternative email: machadodj@usp.br
- Homepage: about.me/machadodj
- Skype name: denisjmachado

6.2 Talk to Luciana (but remember it's probably your fault)

- Name: Luciana dos Santos Regina Lemos
- Email: lulemos@usp.br
- Work phone: +55-11-2065-6672



7. Supporting digital information

SGI factory document's, PBS guides, Altair's license, and administration notes (including tips to create and manage queues) are available at www.ib.usp.br/grant/SecureDownloads/hpc-docs-and-notes.zip. Please contact Denis (contact information above) to get the user name and password needed to download this file.