Linux for Biologists

Robert Bukowski, Qi Sun
Institute of Biotechnology
Bioinformatics Facility
(aka Computational Biology Service Unit - **CBSU**)

Workshop website: https://biohpc.cornell.edu/ww/1/Default.aspx?wid=138

Contact: brc_bioinformatics@cornell.edu

Topics

Week 1

- ❑ What is Linux?
- ❑ Logging in to (and out of) a Linux workstation using ssh client
- ❑ Terminal window tricks
- ❑ Linux directory structure
- ❑ Working with files and directories
- ❑ Persistent multiple shells
- ❑ Graphical applications on Linux

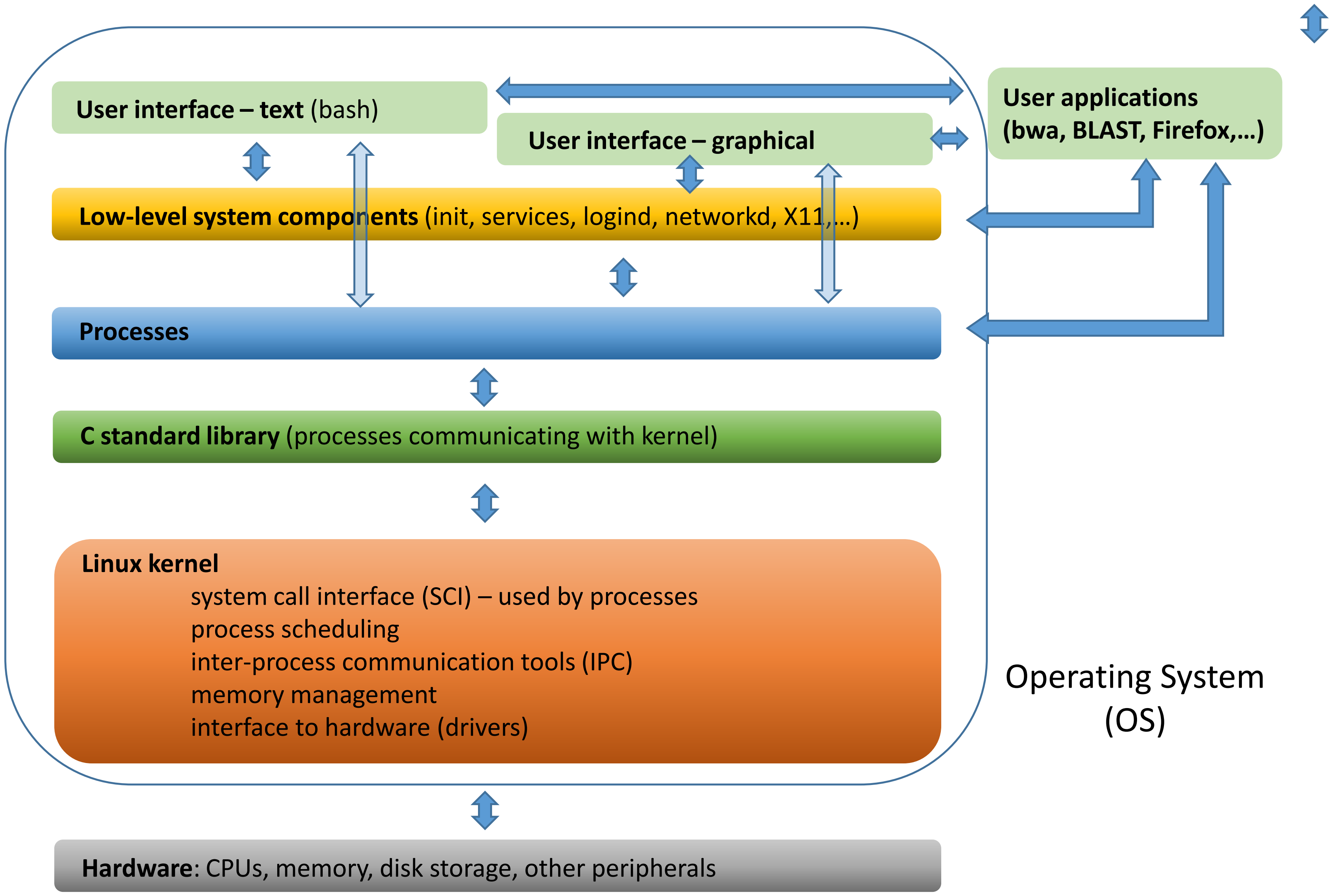
Week 2

- ❑ File transfer between a Linux computer and the world
- ❑ Running programs (non-biological aspects)
- ❑ Very basics of shell scripting
- ❑ Harnessing the power of multiple processors

Week 3

- ❑ Linux in action: processing of large text files common in bioinformatics

What is an operating system?


User interface – text (bash)
User interface – graphical
User applications (bwa, BLAST, Firefox,…)
Low-level system components (init, services, logind, networkd, X11,…)
Processes
C standard library (processes communicating with kernel)
Linux kernel
system call interface (SCI) – used by processes
process scheduling
inter-process communication tools (IPC)
memory management
interface to hardware (drivers)
Operating System (OS)
Hardware: CPUs, memory, disk storage, other peripherals

Operating Systems

- Windows

- Mac OS (distant cousin to Linux)

- Android

- iOS

- Linux OS (Linux kernel + GNU software)
    - open source
    - developed by community (started by Linus Torvalds in 1991)
    - 500+ various ‘distributions’ (customized software collections working with Linux kernel with own package management tools)
        - RedHat (commercial – pay for support)
        - CentOS (free – community RedHat) – that’s what’s installed on BioHPC
        - Ubuntu
        - Debian
        - ….

Why Linux?

- ❑ Majority of bioinformatics/computational biology software developed only for Linux
- ❑ Most programs are command-line (i.e., launched by entering a command in a **terminal window** rather than through GUI)
- ❑ While various graphical and/or web user interfaces exist (e.g., Galaxy, CyVerse Discovery Environment, ~~BioHPC Web~~), but often struggle to provide level of flexibility needed in cutting-edge research
- ❑ Versatile scripting and system tools readily available on Linux allow customization of any analysis, including big data (Week 3)
- ❑ Learning Linux is a good investment

Logging in to a Linux machine

What you need:

- **network name** of the Linux machine (e.g., **cbsum1c2b007.biohpc.cornell.edu**)
- an <u>account</u>, i.e., **user ID** and **password** valid on the Linux machine
- on your laptop: <u>remote access software </u>(typically: **ssh client** or **VNC client**)

- (legal) way to circumvent **firewalls** likely to be present between your laptop and the Linux machine you want to reach

ssh: Secure Shell – provides access to alphanumeric terminal
VNC: Virtual Network Connection - provides access to graphical features (Desktop, GUIs, File Manager, Firefox, …)

Network obstructions: how to reach workshop machines in BioHPC Cloud

- Be on Cornell campus in Ithaca and physically connect laptop to campus network

- If off-campus, install and launch **Cornell VPN** (Virtual Private Network) connection on laptop
    - have to have **Cornell NetID** - for eligibility and instructions check https://it.cornell.edu/cuweblogin-netids-policy/who-eligible-netid
    - info about Cornell VPN: https://it.cornell.edu/cuvpn

- If off-campus and no NetID: connection still possible – more about it later…

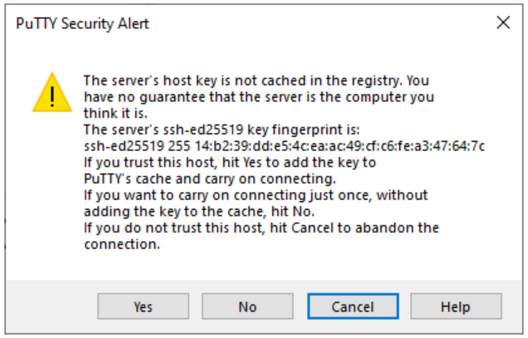
SSH - Windows

- Install PuTTY – open source SSH package for Windows

- Start PuTTY (double-click)

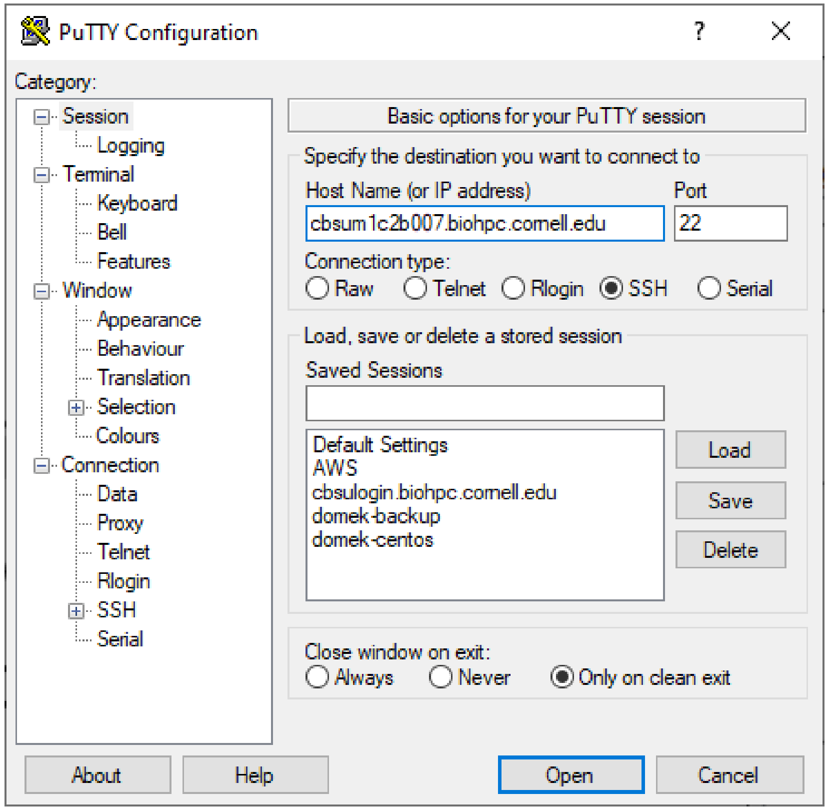
- Type fully qualified server name you want to connect to, e.g. cbsu1c2b007.biohpc.cornell.edu

- Click "Open"



PuTTY Security Alert

```
The server's host key is not cached in the registry. You
have no guarantee that the server is the computer you
think it is.
The server's ssh-ed25519 key fingerprint is:
ssh-ed25519 255 14:b2:39:dd:e5:4c:ea:ac:49:cf:c6:fe:a3:47:64:7c
If you trust this host, hit Yes to add the key to
PuTTY's cache and carry on connecting.
If you want to carry on connecting just once, without
adding the key to the cache, hit No.
If you do not trust this host, hit Cancel to abandon the
connection.
```



PuTTY Configuration

Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): cbsum1c2b007.biohpc.cornell.edu

Port: 22

Connection type: Raw Telnet Rlogin [✓ SSH] Serial

Load, save or delete a stored session

Saved Sessions:

Default Settings
AWS
cbsulogin.biohpc.cornell.edu
domek-backup
domek-centos

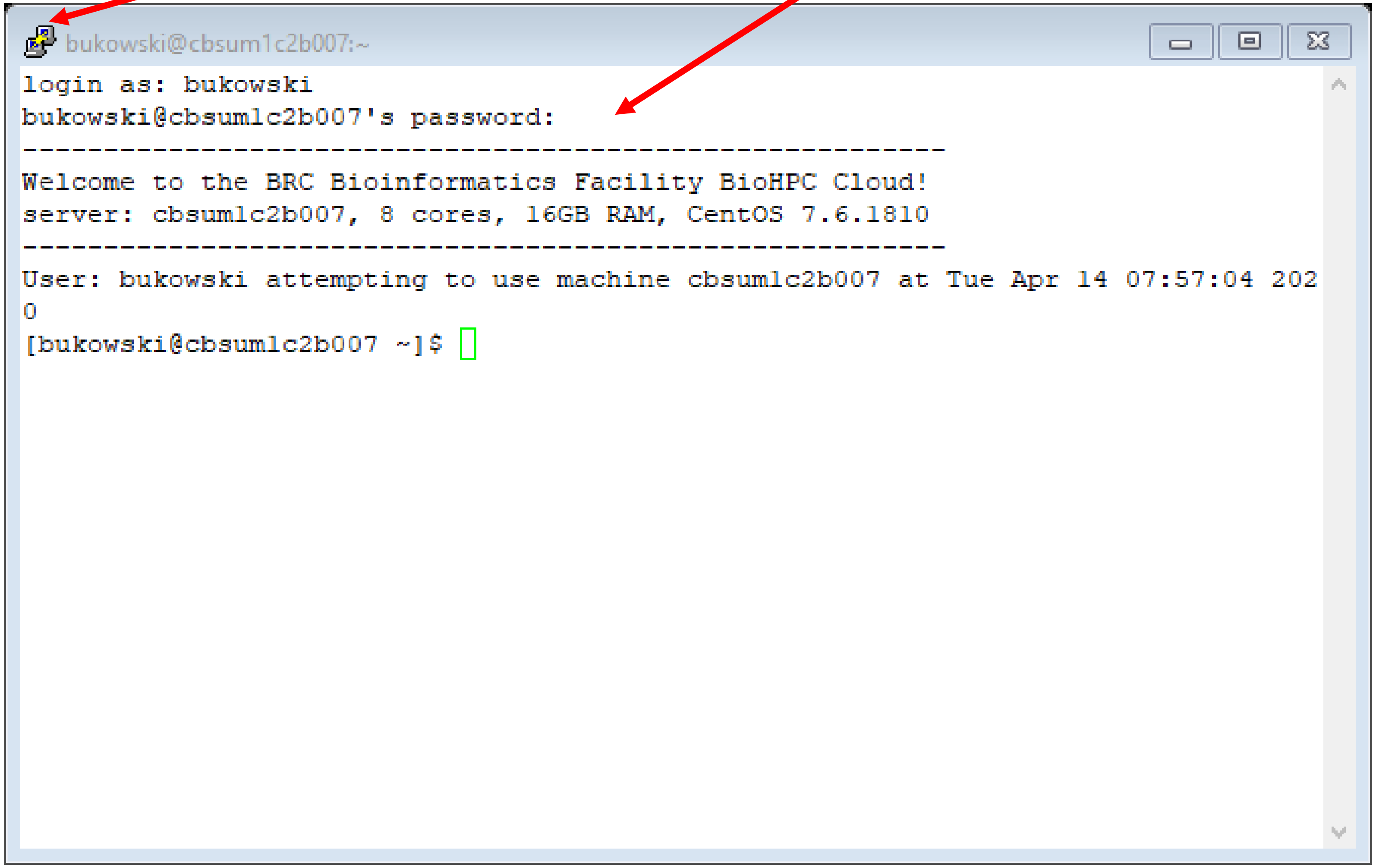
Close window on exit: Always Never [✓ Only on clean exit]

- You can open several terminal windows, if needed (i.e., log in several times)

SSH - Windows

Click for many useful options, e.g., 'Duplicate Session'

Password won't show up when typed



```
login as: bukowski
bukowski@cbsumlc2b007's password:
----------------------------------------------------------
Welcome to the BRC Bioinformatics Facility BioHPC Cloud!
server: cbsumlc2b007, 8 cores, 16GB RAM, CentOS 7.6.1810
----------------------------------------------------------
User: bukowski attempting to use machine cbsumlc2b007 at Tue Apr 14 07:57:04 202
0
[bukowski@cbsumlc2b007 ~]$
```

Logging in via ssh from Mac (or other Linux box)

Use native **ssh client** (already there - no need to install anything)

- Launch the Mac's terminal window and type

```
ssh -Y bukowski@cbsum1c2b007.biohpc.cornell.edu
```

(replace the **cbsum1c2b007** with your reserved workstation, and "**bukowski**" with your own user ID). Enter the password when prompted.

  - When connecting for the first time, a message will appear about "caching server hostkey" – answer "Yes". The message will not appear next time around
  - while you are typing your password, the terminal will appear frozen – this is on purpose!
  - -Y is optional – it enables X11 forwarding –important if you intend to run graphical applications

- You may open several terminal windows, if needed, and log in to the workstation from each of them.

Off-campus and no VPN

1. ssh from your laptop to **cbsulogin.biohpc.cornell.edu**, **cbsulogin2.biohpc.cornell.edu**, or **cbsulogin3.biohpc.cornell.edu**

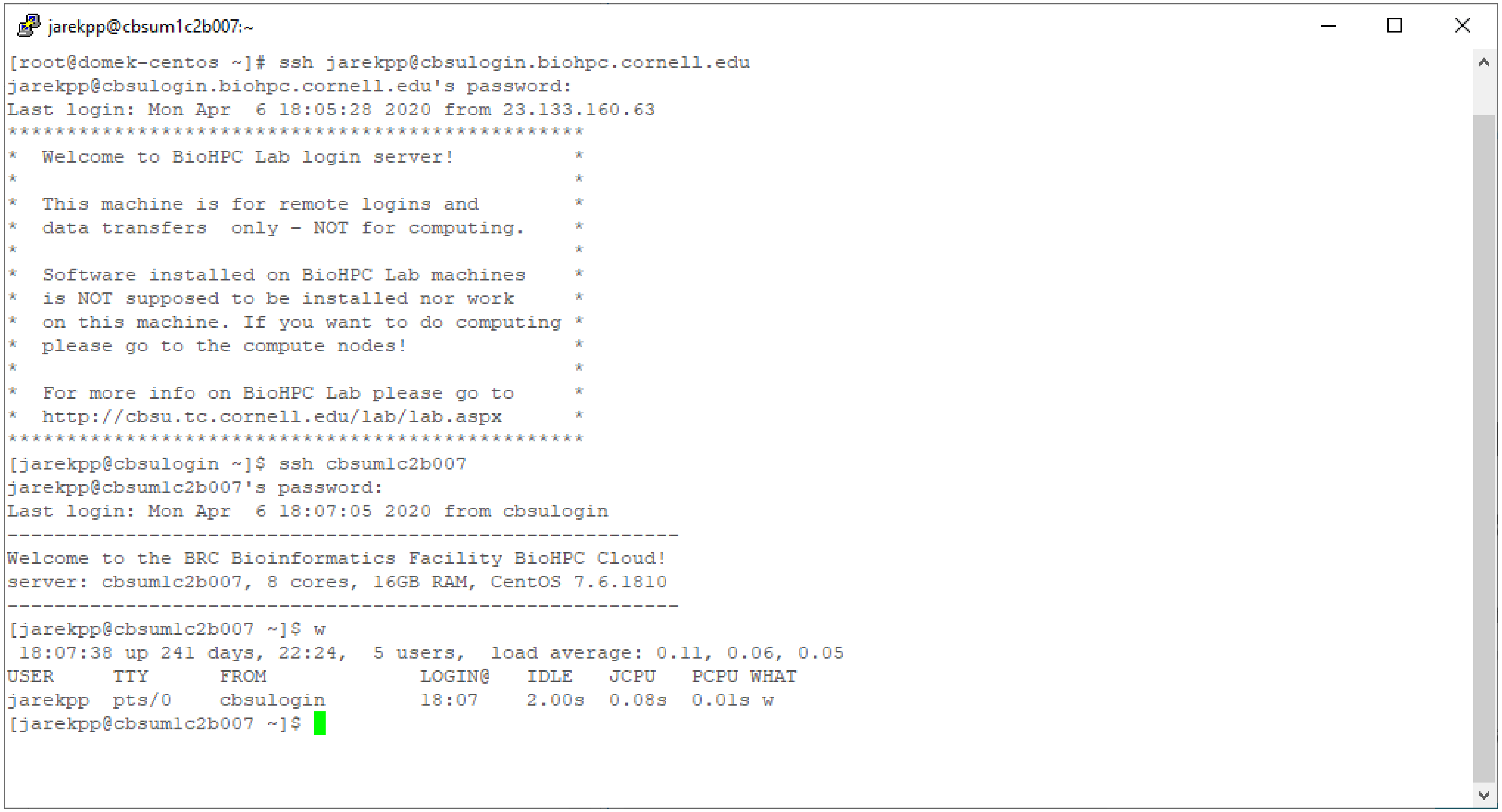
2. From the terminal you just opened on **cbsulogin** (or **cbsulogin2** or **cbsulogin3**), ssh to your reserved BioHPC machine using the **Linux/Mac procedure**, e.g.,

```
ssh userID@cbsum1c2b007.biohpc.cornell.edu
```

or just

```
ssh cbsum1c2b007
```

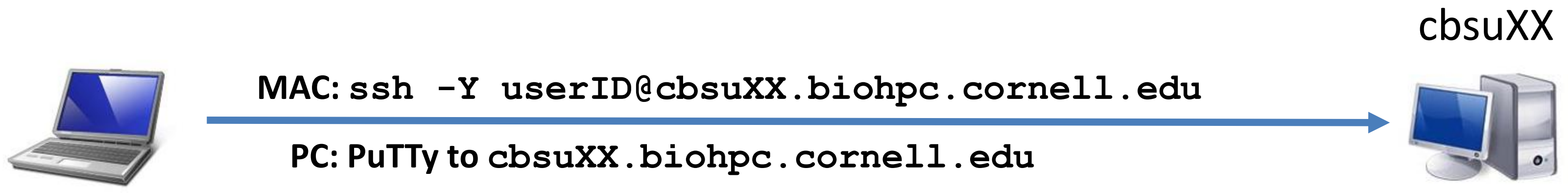
SSH – connect from outside without VPN
(Mac/Linux version)



```
[root@domek-centos ~]# ssh jarekpp@cbsulogin.biohpc.cornell.edu
jarekpp@cbsulogin.biohpc.cornell.edu's password:
Last login: Mon Apr  6 18:05:28 2020 from 23.133.160.63
*************************************************
*  Welcome to BioHPC Lab login server!          *
*                                               *
*  This machine is for remote logins and        *
*  data transfers  only - NOT for computing.    *
*                                               *
*  Software installed on BioHPC Lab machines    *
*  is NOT supposed to be installed nor work     *
*  on this machine. If you want to do computing *
*  please go to the compute nodes!              *
*                                               *
*  For more info on BioHPC Lab please go to     *
*  http://cbsu.tc.cornell.edu/lab/lab.aspx      *
*************************************************
[jarekpp@cbsulogin ~]$ ssh cbsum1c2b007
jarekpp@cbsum1c2b007's password:
Last login: Mon Apr  6 18:07:05 2020 from cbsulogin
---------------------------------------------------------
Welcome to the BRC Bioinformatics Facility BioHPC Cloud!
server: cbsum1c2b007, 8 cores, 16GB RAM, CentOS 7.6.1810
---------------------------------------------------------
[jarekpp@cbsum1c2b007 ~]$ w
 18:07:38 up 241 days, 22:24,  5 users,  load average: 0.11, 0.06, 0.05
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
jarekpp  pts/0    cbsulogin        18:07    2.00s  0.08s  0.01s w
[jarekpp@cbsum1c2b007 ~]$
```

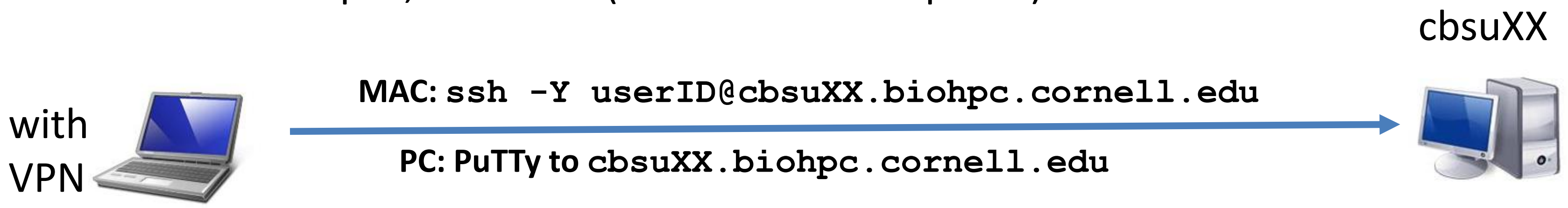
ssh access to BioHPC: summary

On Ithaca campus, with NetID



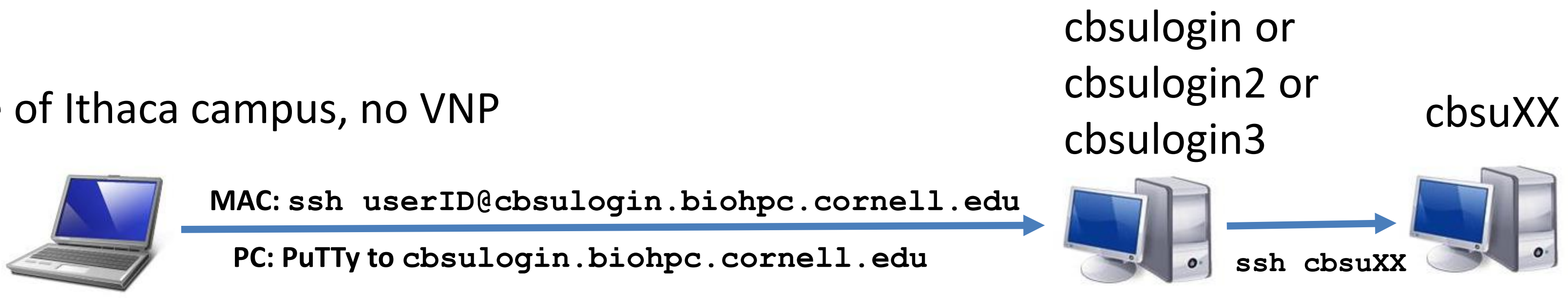
```
MAC: ssh -Y userID@cbsuXX.biohpc.cornell.edu
PC: PuTTy to cbsuXX.biohpc.cornell.edu
```

Outside of Ithaca campus, with VPN (Cornell NetID required)



```
MAC: ssh -Y userID@cbsuXX.biohpc.cornell.edu
PC: PuTTy to cbsuXX.biohpc.cornell.edu
```

Outside of Ithaca campus, no VNP



```
MAC: ssh userID@cbsulogin.biohpc.cornell.edu
PC: PuTTy to cbsulogin.biohpc.cornell.edu
ssh cbsuXX
```

Logging out of an ssh session

While in terminal window, type **exit** or **Ctrl-d** **-** this will close the current terminal window

- If logged in via **cbsulogin** – need to hit Ctrl-d **twice**: first - to exit your machine (e.g., cbsum1c2b007), and second – to exit cbsulogin

One machine, multiple users

Linux is a **multi-access, multi-tasking** system: multiple users may be logged in and run multiple tasks on one machine at the same time, sharing resources (CPUs, memory, disk space)

- This is what is happening during this workshop
- After workshop: when using BioHPC machines for real work, you <u>reserve</u> it all for yourself. You can chose to allow a few other users (collaborators) or not
- BioHPC <u>reservation system</u> is not a part of Linux – it is an add-on we created to better manage access of multiple users to multiple machines

How to access BioHPC machines in the future (after workshop)

BioHPC User's Guide

http://biohpc.cornell.edu/lab/userguide.aspx

Slides from workshop "Introduction to BioHPC"

https://biohpc.cornell.edu/lab/doc/Introduction_to_BioHPC_v9.pdf

Interacting with Linux in terminal window

- User communicates with Linux machine via **commands** typed in the **terminal window**
  - Commands are interpreted by a program referred to as **shell** – an interface between Linux and the user. We will be using the shell called **bash** (another popular shell is **tcsh**).
  - Typically, each command is typed in one line and "**entered**" by hitting the **Enter** key on the keyboard.
  - Commands deal with **files** and **processes**, e.g.,
    - request information (e.g., list user's files)
    - launch a simple task (e.g., rename a file)
    - start an application (e.g., Firefox web browser, BWA aligner, IGV viewer, …)
    - stop an application
  - In this part of the workshop we'll learn mostly about file management commands

Try a few simple commands:

List files and directories (more about it in a minute):

```
ls
ls -al
```

What kind of machine am I on (name, operating system, kernel version, etc.)?

```
uname -a
```

Where on disk am I now (i.e., Print Working Directory)?

```
pwd
```

Who else is logged in? For how long?

```
w
who
```

Use **Manual Pages** to learn more about each command – see all possible **command options**

```
man ls
```

```
man uname
```

Useful tricks

(may not work on all ssh or VNC clients…)

- Helpful tricks to avoid excessive command typing
  - Use **copy/paste**. Any text "mouse-selected" while holding the left mouse button is copied to clipboard. It may then be pasted, e.g., into a command, by clicking the **right** mouse button (PuTTy) or the **middle** button (when working through the console in 625 Rhodes).
  - Use **Up/Down arrow keys** – this will cycle through recently executed commands.
  - Use the **TAB key** – this will often present you with a list of choices after typing a part of a command – no need to remember everything.

Useful tricks

- Helpful tricks to avoid excessive command typing

`history` command: list all recently used commands – can copy a desired command and paste it to execute again, or refer to a command by its index

Examples:

```
history
```

(list all remembered commands)

```
history | less
```

(list all remembered commands page by page)

```
history | grep workdir
```

(list all remembered commands containing string "workdir")

Screen output from a command may be saved to disk

Each command produces two output streams: **standard output** (STDOUT) and **standard error** (STDERR). Normally, they both are displayed on the screen.

But they can be saved on disk ("redirected")

Save to separate files (file names are arbitrary) …

```
who  > OUT.log  2> ERR.log
```

… or save to a single file

```
who >& OUTERR.log
```

These files are text files and can be looked at with any text processing tool (more about it later)

`less OUTERR.log`	page through the file (use **`more`** to page forward)
`cat OUTERR.log`	print the file on screen
`nano OUTERR.log`	open file in text editor

Files and directory tree

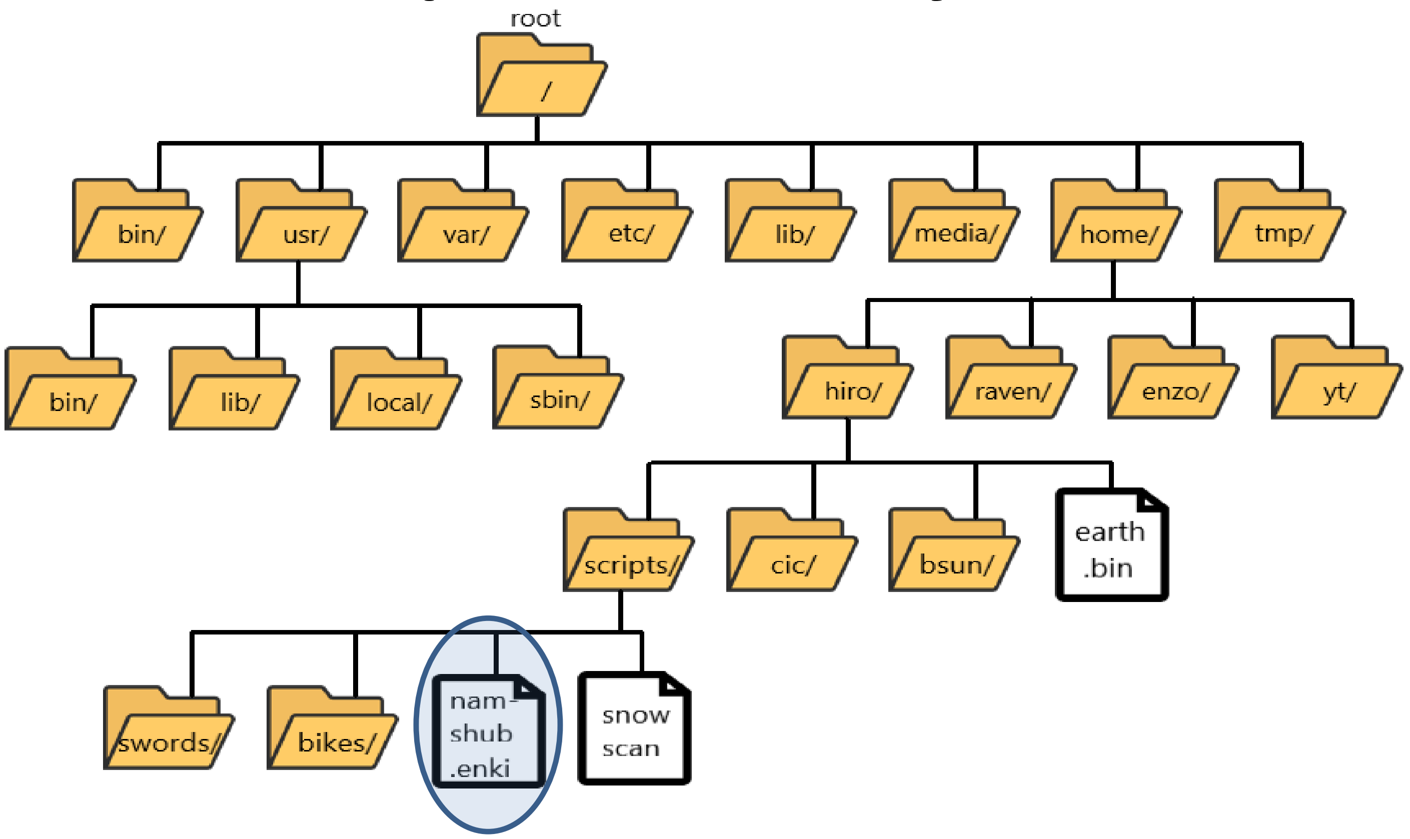
Files and directory tree

Data and **programs** are stored in **files** on **disk storage**

Each **file** has a **name** and is located in a **directory** (a.k.a. **folder**)

> **directory** – a logical location on disk
>
> **(directory, name)** pair uniquely specifies the file
>
> a **directory** may hold **files** and/or **other directories**
>
> > directories form tree structure

Example of directory tree



How to refer to a file?

Full path (starts with **/):**	`/home/hiro/scripts/nam-shub.enki`
Relative path (to `/home/hiro`)	`scripts/nam-shub.enki`
Relative path (to `/home/hiro/scripts`)	`nam-shub.enki`
Relative path (to `/home/enzo`)	`../hiro/scripts/nam-shub.enki`

‘**current directory**’

`pwd` command will show what it is
relative paths will be assumed relative to it
ls command (with no arguments) will show content of it

home directory

typically: **`/home/userID`**
user’s private (logical) space on disk storage
becomes ‘current directory’ right after logging in

Traversing directory tree

Right after logging in or opening a terminal window, "you are" in your **home directory** (e.g., **`/home/bukowski`**).

Where am I?

```
pwd
```

*(**p**rint **w**orking **d**irectory) – show the **current directory**; any **relative path** you specify will be relative to this place*

Navigating through directories

```
cd
```

C**hange (current) **d**irectory; without additional arguments, this command will take you to your **home directory

```
cd /workdir/bukowski/indexes
```

***C**hange (current) **d**irectory from wherever to /workdir/bukowski/indexes.*

```
cd indexes
```

***C**hange (current) **d**irectory to indexes (will work if the current directory contains "indexes")*

```
cd ../
```

***C**hange (current) **d**irectory one level back (closer to the root)*

```
cd ../../../
```

***C**hange (current) **d**irectory three levels back (closer to the root)*

```
cd ./
```

***C**hange (current) **d**irectory to the **same one** (i.e., do nothing). Note: **./** or just **.** refers to the current directory.*

Working with Directories

Creating directories

```
mkdir  /home/bukowski/my_new_dir
```

Make a new directory called "my_new_dir" in /home/bukowski

```
mkdir my_new_dir
```

Make a new directory called "my_new_dir" in the current directory

Removing directories

```
rmdir  /home/bukowski/my_new_dir
```

Remove directory called "my_new_dir" in /home/bukowski – will fail if the directory is not empty

```
rm -Rf  /home/bukowski/my_new_dir
```

Remove directory called "my_new_dir" in /home/bukowski with all its content (i.e. all files and subdirectories will be gone)

```
rm -Rf  my_new_dir
```

Remove directory called "my_new_dir" in current directory with all its content (i.e. all files and subdirectories will be gone)

Listing content (files and subdirectories) of a directory

ls

(list)

```
ls
```

List files and directories in current directory (in short) format

```
ls -al
```

List all files and directories in current directory in long format

```
ls -al /home/bukowski/tst
```

List content of /home/bukowski/tst (which does not have to be the current directory)

```
ls -alt *.txt
```

Lists all files and directories with names ending with ".txt" in the current directory, sorted according to modification time (use **ls –altr** *to sort in reverse)*

```
ls -alS
```

Lists content of the current directory sorted according to size (use **ls –alSr** *to sort in reverse)*

```
ls -al | less
```

Lists content of the current directory using pagination – useful if the file list is long (SPACE bar will take you to the next page, "q" will exit)

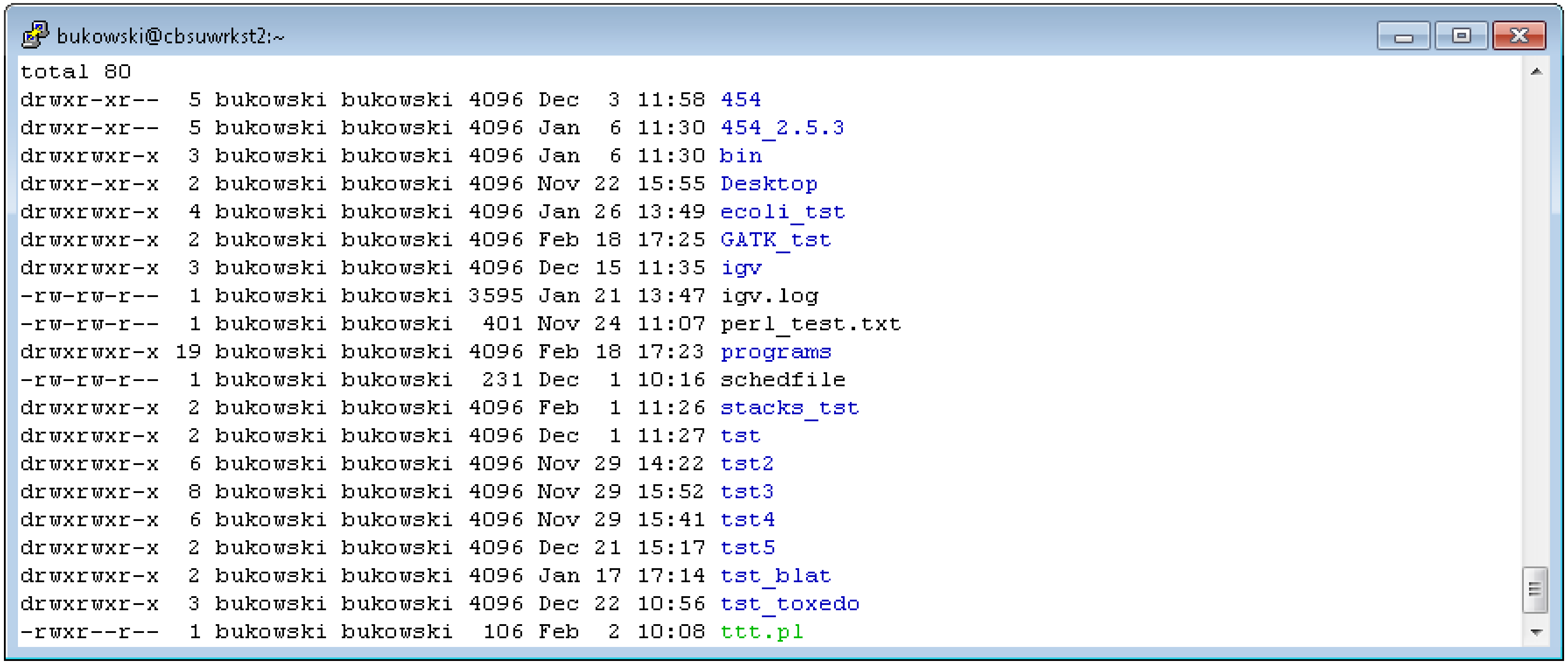
pipe

Output from first command is "piped" as input to the second

LOTS more options for ls – try `man ls` to see them all (may be intimidating).

Listing content of a directory

```
ls -al
```



```
total 80
drwxr-xr--   5 bukowski bukowski 4096 Dec  3 11:58 454
drwxr-xr--   5 bukowski bukowski 4096 Jan  6 11:30 454_2.5.3
drwxrwxr-x   3 bukowski bukowski 4096 Jan  6 11:30 bin
drwxr-xr-x   2 bukowski bukowski 4096 Nov 22 15:55 Desktop
drwxrwxr-x   4 bukowski bukowski 4096 Jan 26 13:49 ecoli_tst
drwxrwxr-x   2 bukowski bukowski 4096 Feb 18 17:25 GATK_tst
drwxrwxr-x   3 bukowski bukowski 4096 Dec 15 11:35 igv
-rw-rw-r--   1 bukowski bukowski 3595 Jan 21 13:47 igv.log
-rw-rw-r--   1 bukowski bukowski  401 Nov 24 11:07 perl_test.txt
drwxrwxr-x  19 bukowski bukowski 4096 Feb 18 17:23 programs
-rw-rw-r--   1 bukowski bukowski  231 Dec  1 10:16 schedfile
drwxrwxr-x   2 bukowski bukowski 4096 Feb  1 11:26 stacks_tst
drwxrwxr-x   2 bukowski bukowski 4096 Dec  1 11:27 tst
drwxrwxr-x   6 bukowski bukowski 4096 Nov 29 14:22 tst2
drwxrwxr-x   8 bukowski bukowski 4096 Nov 29 15:52 tst3
drwxrwxr-x   6 bukowski bukowski 4096 Nov 29 15:41 tst4
drwxrwxr-x   2 bukowski bukowski 4096 Dec 21 15:17 tst5
drwxrwxr-x   2 bukowski bukowski 4096 Jan 17 17:14 tst_blat
drwxrwxr-x   3 bukowski bukowski 4096 Dec 22 10:56 tst_toxedo
-rwxr--r--   1 bukowski bukowski  106 Feb  2 10:08 ttt.pl
```

File permissions ("d" means this is a directory)

Owner and group

Size (in bytes) – meaningful for files, but not directories

Last modification time

File name (directories in blue, executable files in green)

Storage

Linux directory structure is *continuous*, i.e. regardless of the physical location of storage, logically it all appears as part of single directory tree starting from root (/).

But differents parts of the tree may have different physical locations (local or network)

affects storage access speed

Not easy to tell which storage is local and which network just by directory name. Remember the setup on BioHPC machines:

- **Networked storage**

  /home

  /shared_data

  /programs

- **Local storage**

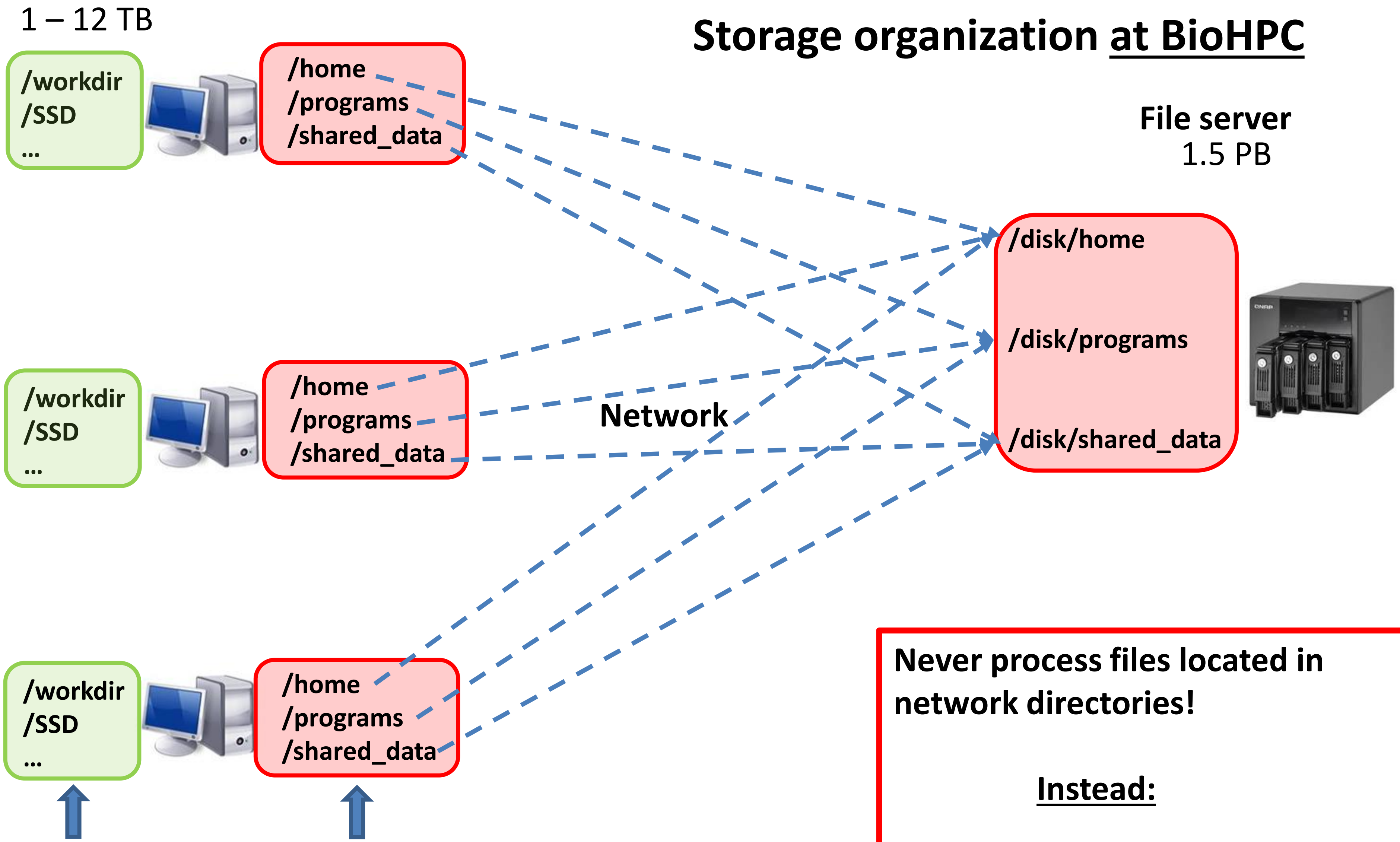
  /workdir

  /SSD

  /local_data

Will look different on other machines or centers – always check description!

Storage organization at BioHPC



Local file systems (fast, few users, temporary)

Network file systems (slow, many users, permanent)

Never process files located in network directories!

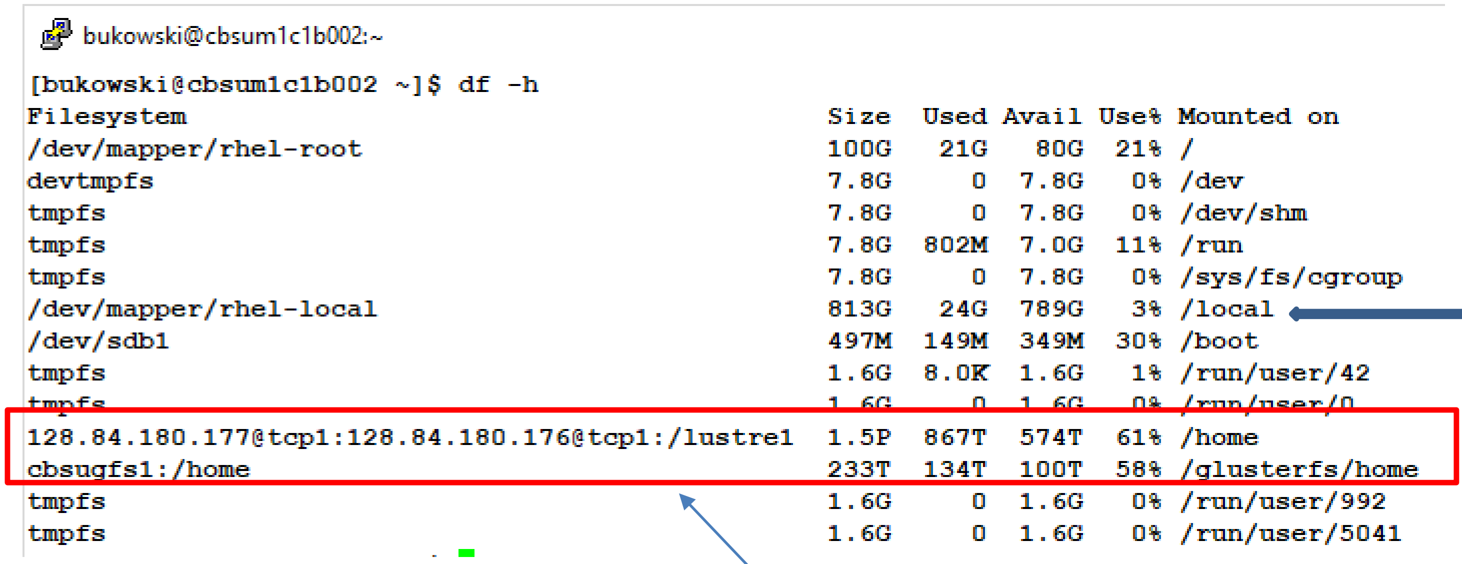
Instead:

Copy files to `/workdir` or `/SSD` and process them there. When finished, copy results back to `/home/yourID`

`df` command…

… tells how much disk space is available on various file systems:

```
df -h
```



```
[bukowski@cbsum1c1b002 ~]$ df -h
Filesystem                                       Size  Used Avail Use% Mounted on
/dev/mapper/rhel-root                            100G   21G   80G  21% /
devtmpfs                                         7.8G     0  7.8G   0% /dev
tmpfs                                            7.8G     0  7.8G   0% /dev/shm
tmpfs                                            7.8G  802M  7.0G  11% /run
tmpfs                                            7.8G     0  7.8G   0% /sys/fs/cgroup
/dev/mapper/rhel-local                           813G   24G  789G   3% /local
/dev/sdb1                                        497M  149M  349M  30% /boot
tmpfs                                            1.6G  8.0K  1.6G   1% /run/user/42
tmpfs                                            1.6G     0  1.6G   0% /run/user/0
128.84.180.177@tcp1:128.84.180.176@tcp1:/lustre1 1.5P  867T  574T  61% /home
cbsugfs1:/home                                   233T  134T  100T  58% /glusterfs/home
tmpfs                                            1.6G     0  1.6G   0% /run/user/992
tmpfs                                            1.6G     0  1.6G   0% /run/user/5041
```

<u>local</u>
/workdir,
/local_data

<u>networked</u>
/home
/programs
/shared_data

These are network devices – starting with "computername:/"

Checking my disk space

How much disk space is taken by my files?

```
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)

May take some time if you have a lot of small files

Working with files

There are many types of files. Here are the most important:

❑ **Text files** (human-readable; can be viewed and modified using a text editor)

- Text documents (e.g., README files)
- Data in text format (e.g., FASTA, FASTQ, VCF, …)
- Scripts:
  - Shell scripts (usually **`*.sh`** or **`*.csh`**)
  - Perl scripts (usually **`*.pl`**)
  - Python scripts (usually **`*.py`**)
  - …

Working with files

❑ **Binary files** (not human-readable; cannot be viewed using a text editor)

- Executables (e.g., samtools, bwa, bowtie, firefox)
- Data in binary format (e.g, BAM files, index files for BWA or Bowtie, formatted BLAST databases)
- Compressed files (usually `*.gz`, `*.zip`, `*.bz2`,…, but extensions not necessary) – often text files re-formatted to save space on disk or packaged directory trees

Working with files

There are many types of files. Here are the most important:

- **Symbolic links**: pointers to other files or directories.

```
cd /programs/bin/samtools
ls -al samtools
```

```
lrwxrwxrwx 1 root root 30 Apr 16  2013 samtools -> ../../samtools-1.2/samtools
```

In the example above, file **`/programs/bin/samtools/samtools`** is a symbolic link to **`/programs/samtools-1.2/samtools`**.

Note the “l” character in the first column of output from “**`ls -al`**”.

Working with files

Where do files come from?

They are created by various programs, e.g.,

- Text editors
- File compression tools
- Aligners
- Assemblers
- …
- System commands (copy, move, rename, etc.)
- Screen output redirection (>, >&)
- Remote copy tools (scp, sftp, wget, Firefox)

Creating an empty file (zero size):

```
touch my_file
```

`my_file` is empty (so one can't say if it is a text file or binary file…)

Working with files

File and directory names – best practices

- ❑ Names are case-sensitive (`MyFile`, `myfile`, `myFile` are all different!)
- ❑ Use only **letters** (upper- and lower-case), **numbers** from 0 to 9, a **dot** (.), an **underscore** (_), a dash (-) [good example: `This_is-myFile99.abc`]
- ❑ Avoid other characters, as they may have special meaning to either Linux, or to the application you are trying to run. **Do not use "space"** or other special characters [bad example: `This is my&File#^99.abc`]
- ❑ Use of special characters in file names is possible if absolutely necessary, but will lead to problems if done incorrectly.
- ❑ "Extensions" (like `.zip`, `.gz`, `.ps`, **.sam**, **.bam**, **.fastq**., **.fa**, **.gff**…) are commonly used to denote the type of file, but are typically not necessary to "open" or use a file. While working in command line terminal you always explicitly specify a program which is supposed to work with (open) this file.

Basic operations on files - summary

Listing

```
ls
ls -al
```

Copying

```
cp  <path_to_source>  <path_to_destination>
```

Moving and/or renaming

```
mv <path_to_source> <path_to_destination>
```

Deleting

```
rm <path_to_file>
```

Deleting whole directory with all its content

```
rm -Rf <path_to_directory>
```

Working with files

Copying a file

```
cp    <source file>    <destination file>
```

Examples:

```
cp sample_data.fa /workdir/bukowski/sample.fa
```

(copy file sample_data.fa from the current directory to /workdir/bukowski and give the copy a name sample.fa; destination directory must exist)

```
cp /workdir/bukowski/my_script.sh .
```

(copy file myscript.sh from /workdir/bukowski to the current directory under the same file name)

```
cp /home/bukowski/*.fastq  /workdir/bukowski
```

(copy all files with file names ending with ".fastq" from /home/bukowski to /workdir/bukowski; destination directory must exist)

```
cp –R /workdir/bukowski/tst5 /home/bukowski
```

(if tst5 is a directory, it will be copied with all its files and subdirectories to directory /home/bukowski/tst5; if /home/bukowski/tst5 did not exist, it will be created).

Try `man cp` for all options to the **`cp`** command.

Working with files

Moving and renaming files

```
mv <source_file>    <destination_file>
```

Examples:

```
mv my_file_one my_file_two
```

(change the name of the file my_file_one in the current directory)

```
mv my_file_one /workdir/bukowski
```

(move the file my_file_one from the current directory to /workdir/bukowski without changing file name; the file will be removed from the current directory)

```
mv /workdir/bukowski/my_file_two ./my_file_three
```

(move the file my_file_two from /workdir/bukowski to the current directory changing the name to my_file_three; the file will be removed from /workdir/bukowski)

Try `man mv` for all options to the **`mv`** command….

Working with files

Removing (deleting) files

```
rm    <file_name>
```

Examples:

```
rm my_file_one
```

(delete file my_file_one from the current directory)

```
rm /workdir/bukowski/my_file_two
```

(delete file my_file_two from directory /workdir/bukowski)

```
rm -Rf ./tst5
```

(if tst5 is a subdirectory in the current directory, it will be removed with all its files and directories)

Try `man rm` for all options to the **`rm`** command….

Working with files

What kind of file is this?

Since there are no strict naming conventions for various file types, it is not always clear what kind of file we deal with. When in doubt, try the `file` command:

```
cd  /programs/samtools-0.1.11
file samtools
```

this is an executable program….

```
samtools: ELF 64-bit LSB executable, AMD x86-64, version 1
(SYSV), for GNU/Linux 2.6.9, dynamically linked (uses shared
libs), for GNU/Linux 2.6.9, not stripped
```

… which uses "shared" libraries", i.e., may not work if moved to other machine where these libraries are absent

Working with files

<u>Looking for a file</u>

```
find . -name  PHG47_sorted.bam -print
```

(look for all files called **`PHG47_sorted.bam`** *in the current directory and recursively in all its subdirectories)*

```
find /data1 -name "*PHG47*" -print
```

*(look for all files having "***`PHG47`***" in the name, located in /data1 or recursively in its subdirectories)*

Try `man find` for many more options

Working with files: archiving and compression

To save disk space, we can **compress** large files if we do not intend to use them for a while. Files downloaded from the web are typically compressed and sometimes need to be uncompressed before processing can take place.

Common compressed formats and compression/decompression tools:

Format (extension)	Tool	Function
gz	gzip	Compress a single file
bz2	bzip2	Compress a single file
zip	zip	Make compressed archive (single file) of a directory structure; same as on Windows
tar	tar	Make an archive (single file) of a directory structure
tgz (tar.gz)	tar	Make a compressed archive (single file) of a directory structure

Compression works best (i.e., saves most disk space) for text files (e.g., large FASTQ files).

Getting help about compression tools:

- **`gzip -h, bzip2 --help, zip, tar --help`**
- **`man gzip, man bzip2, man zip, man tar`** (may be intimidating…)

File compression: examples

- **gzip (gz)**

```
gzip my_file
```

(compresses file my_file, *producing its compressed version,* `my_file.gz`*)*

```
gzip -d my_file.gz
```

(decompress my_file.gz, *producing its original version* `my_file`*)*

- **bzip2**

```
bzip2 my_file
```

(compresses file my_file, *producing its compressed version,* `my_file.bz2`*)*

```
bunzip2 my_file.bz2
```

(decompress my_file.bz2, *producing its original version* `my_file`*)*

Archiving and compression: examples

- **zip**

```
zip  my_file.zip  my_file1  my_file2  my_file3
```

(create a compressed archive called my_files.zip*, containing three files:* my_file1, my_file2, my_file3*)*

```
zip -r my file.zip  my file1  my dir
```

(if my_dir *is a directory, create an archive* my_file.zip *containing the file* my_file1 *and the directory* my_dir *with all its content)*

```
zip –l my_file.zip
```

(list contents of the zip archive my_file.zip*)*

```
unzip my_files.zip
```

(decompress the archive into the constituent files and directories

Archiving with tar: examples

- **tar**

```
tar -cvf  my_file.tar  my_file1  my_file2  my_dir
```

(create a compressed archive called my_files.tar*, containing files* my_file1, my_file2 *and the directory* my_dir *with all its content)*

```
tar -tvf   my_file.tar
```

(list contents of the tar archive my_file.tar*)*

```
tar -xvf  my_files.tar
```

(decompress the archive into the constituent files and directories)

Archiving and compression with tar: examples

- **tgz** (also, tar.gz – essentially a combo of “tar” and “gzip”)

```
tar -czvf my_file.tgz  my_file1  my_file2  my_dir
```

(create a compressed archive called my_files.tgz*, containing files* my_file1, my_file2 *and the directory* my_dir *with all its content)*

```
tar -tzvf my_file.tgz
```

(list contents of the tar archive my_file.tar*)*

```
tar -xzvf my_files.tgz
```

(decompress the archive into the constituent files and directories)

Working with text files

Linux features standard tools for text file processing:

Function	tool
Text editing	`vi, nano, gedit,` …
Page through the file	`less, more`
Select lines from top, bottom, or middle of file	`head, tail`
Select lines containing a string	`grep`
Select columns	`cut`
Append rows to a file	`cat`
Append columns to a file	`paste`
Sort a file over column(s)	`sort`
Count lines, words, characters	`wc`
Advanced, text-focused scripting tools	`awk, sed`
General scripting tools (not only in Linux)	`perl, python`

Working with text files: editors

vim

- Available on all UNIX-like systems (Linux included), i.e., also on BioHPC workstations (type **vi** or **vi file_name**)
- Free Windows implementation available (once you learn vi, you can just use one editor everywhere)
- Runs locally on Linux machine (no network transfers)
- User interface rather peculiar (no nice buttons to click, need to remember quite a few keyboard commands instead)
- Some love it, some hate it

nano

- Available on most Linux machines (our workstations included; type **nano** or **nano file_name**)
- Intuitive user interface. Keyboard commands-driven, but help always displayed on bottom bar (unlike in vi).
- Runs locally on Linux machine (no network transfers during editing)

gedit (installed on BioHPC workstations; just type **gedit** or **gedit file_name** to invoke)

- X-windows application – need to have X-manager running on client PC.
- May be slow on slow networks…

vi basics

Opening a file:

vi my_reads.fastq (open the file my_reads.fastq in the current directory for editing; if the file does not exist, it will be created)

Command mode: typing will issue commands to the editor (rather than change text itself)

Edit mode: typing will enter/change text in the document

<Esc>	exit edit mode and enter command mode (this is the most important key – use it whenever you are lost)

The following commands will **take you to edit mode**:

i	enter insert mode
r	single replace
R	multiple replace
a	move one character right and enter insert mode
o	start a new line under current line
O	start a new line above the current line

The following commands **operate in command mode (hit <Esc> before using them)**

x	delete one character at cursor position
dd	delete the current line
G	go to end of file
1G	go to beginning of file
154G	go to line 154
$	go to end of line
1	go to beginning of line
:q!	exit without saving
:w	save (but not exit)
:wq!	save and exit

Arrow keys: move cursor around (in both modes)

Working with text files

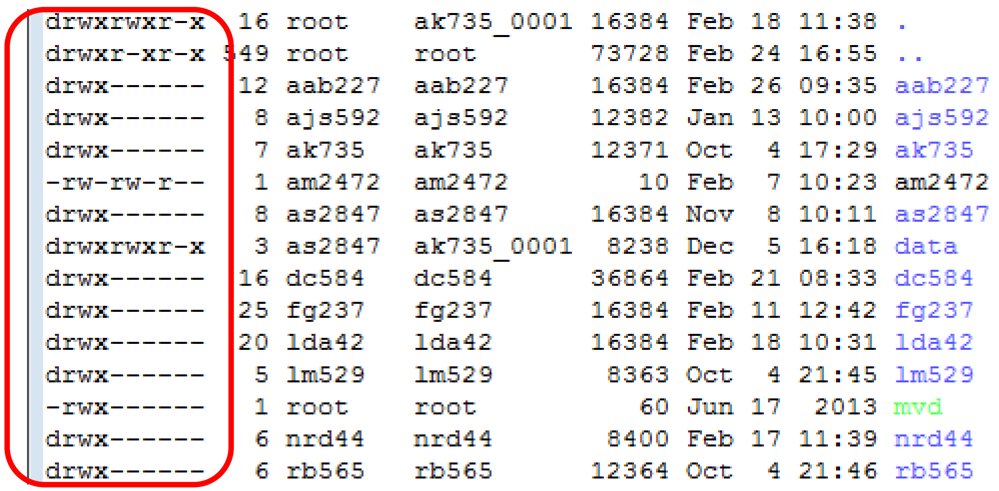
NOTE: Text files prepared using advanced text processors (e.g., MS Word) will cause problems when used as input to Linux applications.

If you have to use such files on Linux – always save as “**Plain Text**”

Controlling file access: user groups in Linux

- On a Linux system, users may be organized in groups
- Be default, a group is created for each user (with group name the same as user ID)
  - Example: a group **`bukowski`** is created along with user **`bukowski`**
- Other users may belong to a given user's group
  - Example: user **`jarekp`** may belong to group **`bukowski`**
- Other groups may be also defined (not named after any user IDs) and contain multiple users
- A user may belong to multiple (up to 15) groups (one of them is <u>primary)</u>
- Groups are set up by an administrator
- User's membership in groups determine this user's access to files
  - Each file and directory has an **owner** and a **group**, each with separate set of access permissions, and another set of permissions for **everybody else**

File permissions



```
drwxrwxr-x  16 root    ak735_0001 16384 Feb 18 11:38 .
drwxr-xr-x 549 root    root       73728 Feb 24 16:55 ..
drwx------  12 aab227  aab227     16384 Feb 26 09:35 aab227
drwx------   8 ajs592  ajs592     12382 Jan 13 10:00 ajs592
drwx------   7 ak735   ak735      12371 Oct  4 17:29 ak735
-rw-rw-r--   1 am2472  am2472        10 Feb  7 10:23 am2472
drwx------   8 as2847  as2847     16384 Nov  8 10:11 as2847
drwxrwxr-x   3 as2847  ak735_0001  8238 Dec  5 16:18 data
drwx------  16 dc584   dc584      36864 Feb 21 08:33 dc584
drwx------  25 fg237   fg237      16384 Feb 11 12:42 fg237
drwx------  20 lda42   lda42      16384 Feb 18 10:31 lda42
drwx------   5 lm529   lm529       8363 Oct  4 21:45 lm529
-rwx------   1 root    root          60 Jun 17  2013 mvd
drwx------   6 nrd44   nrd44       8400 Feb 17 11:39 nrd44
drwx------   6 rb565   rb565      12364 Oct  4 21:46 rb565
```

d r w x r w x r w x: User (owner), Group,   Others

"d": directory (or "-" if file); "r": read permission;   "w": write permission;   "x": execute permission (or permission to "cd" if it is a directory);   "-": no permission

<u>Examples:</u>

`data`:

- is a directory ("d" in the first column)
- everybody can read and "cd" to it, but not write ("`r-x`" in the last three columns)
- owner (**`as2847`**) and everybody in the group (**`ak735_0001`**) can also write to it

`am2472`:

- is a file readable by everybody and writable by owner and his group
- the file is not executable by anyone

`rb565`:

- is a directory accessible only by owner

Changing file permissions

chmod command – some examples

```
chmod  o-rwx /home/bukowski
```

make my home directory inaccessible to others ("o")

```
chmod  ug+x  my script.pl
```

make the file my_script.pl (in the current directory) executable by the <u>owner</u> ("u") and the members of the <u>group</u> ("g").

```
chmod  a-w /workdir/bukowski/my_file
```

deny <u>all</u> ("a"), including the owner, the right to write to the file my_file *(in* /workdir/bukowski*)*

Try **`man chmod`** for more information (may be somewhat intimidating!)

Want to make your files accessible to some (but not all) other users? Contact us!

- we would need to make sure that you and those other users are in the same user groups

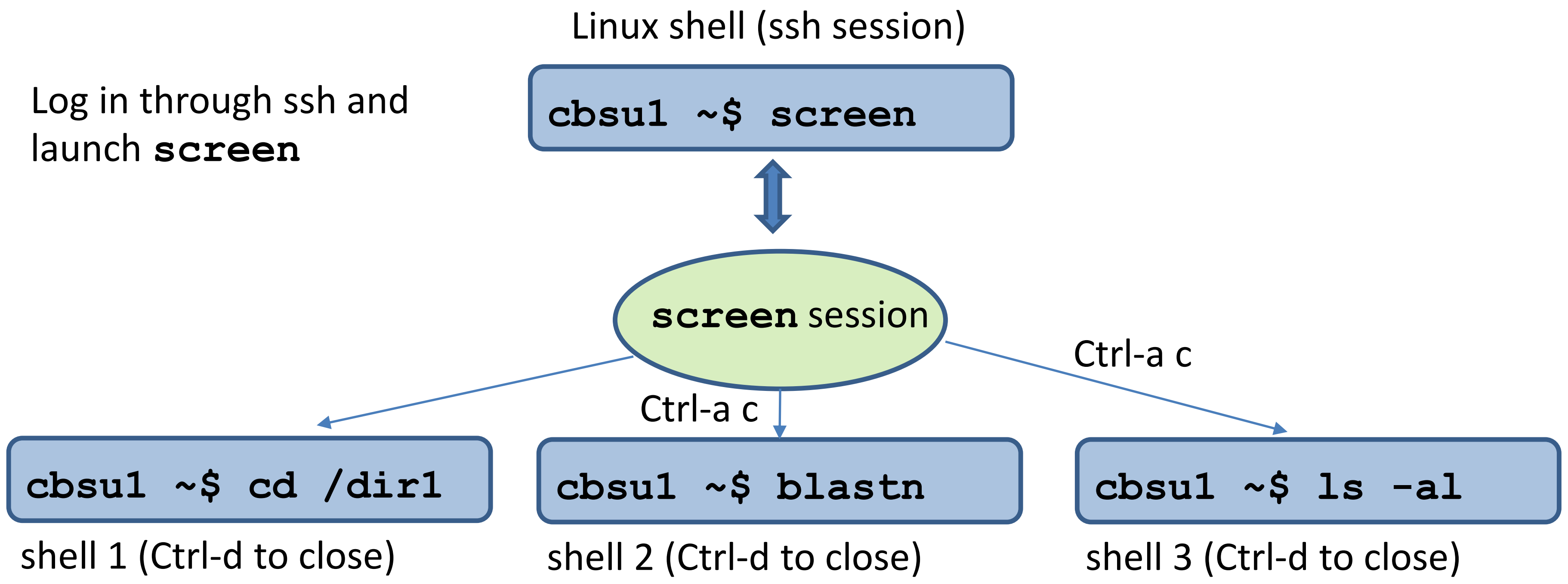
Multiple shells and graphics

Running multiple shells at the same time

- ❑ Start a few **separate ssh sessions** (e.g., can use “Duplicate session” function in PuTTy)
    - Separate window for each shell
- ❑ **screen**: a program which allows running **multiple shells** within **one “screen session” in a single terminal window**
    - All shells run in a single window (which can be divided, but not too convenient)
    - can **switch** between the shells with a few keystrokes
    - can **detach** the whole screen session (with all shells running) and **re-attach** it later
    - Screen session **survives connection/laptop crashes** – perfect way of keeping long jobs running

Using `screen`

Log in through ssh and launch **`screen`**


Linux shell (ssh session)
cbsu1 ~$ screen
screen session
Ctrl-a c
Ctrl-a c
cbsu1 ~$ cd /dir1
cbsu1 ~$ blastn
cbsu1 ~$ ls -al
shell 1 (Ctrl-d to close)
shell 2 (Ctrl-d to close)
shell 3 (Ctrl-d to close)

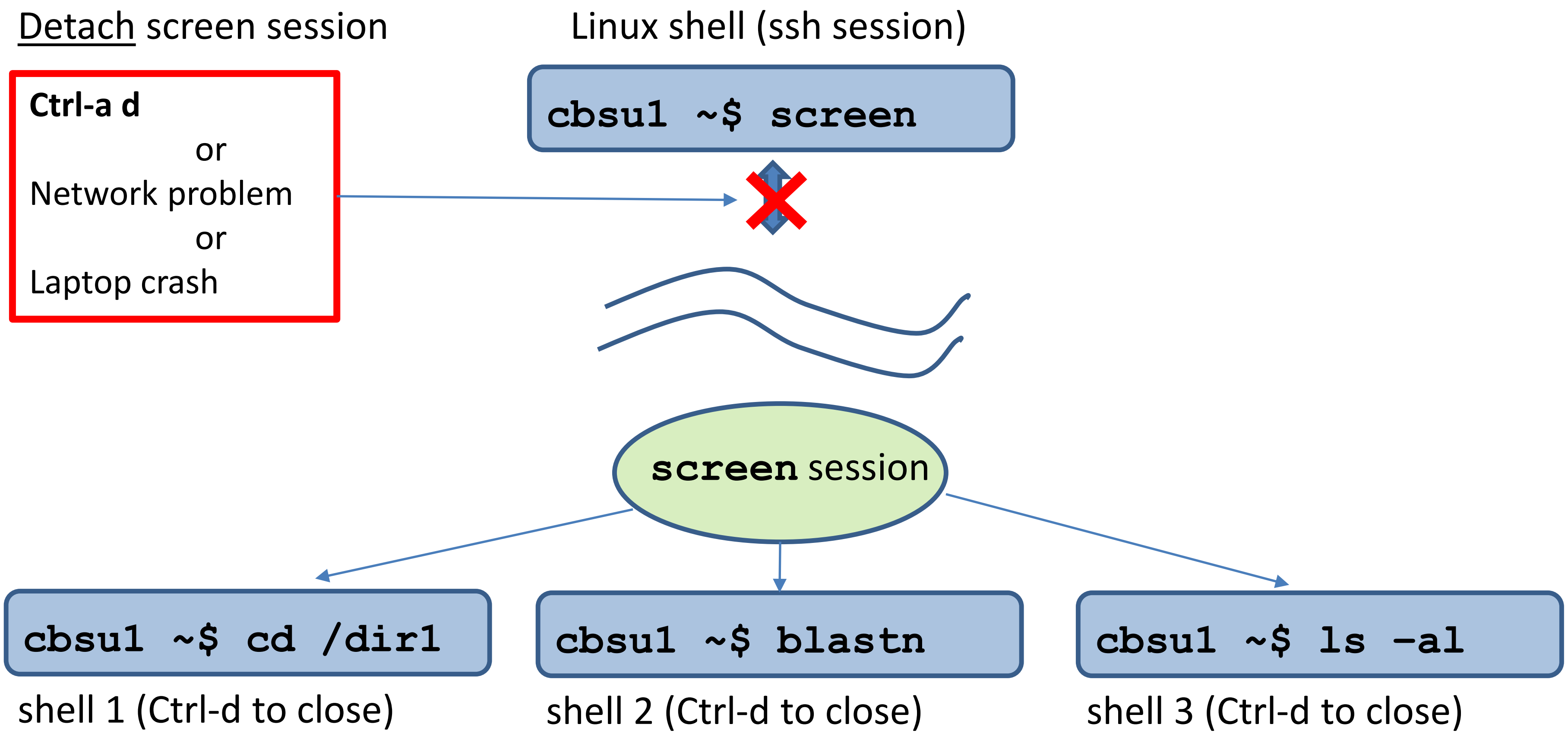
Ctrl-a c creates a new shell within the screen session

Ctrl-a p and **Ctrl-a n** switch back-and-forth between the shells

Can do different things in each shell, in different directories, etc.

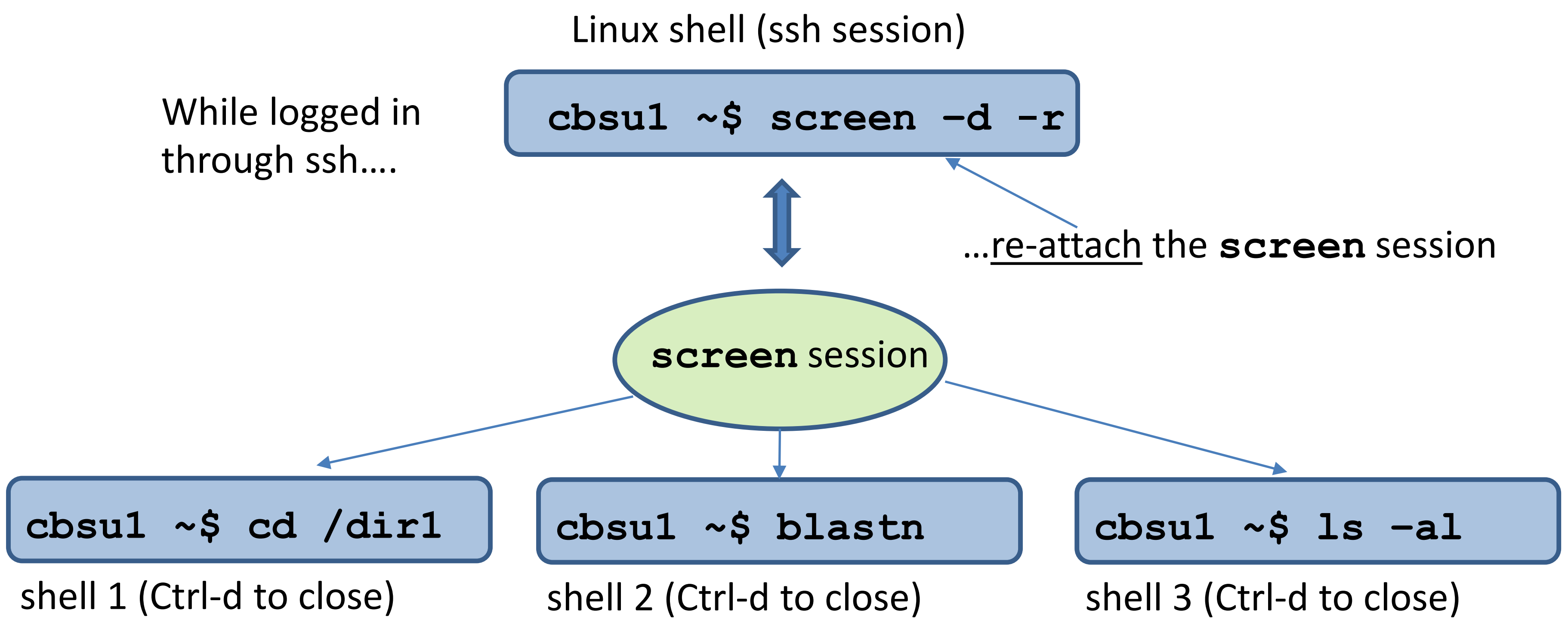
Ctrl-d closes the <u>current</u> shell (i.e., the one currently displayed); others remain active

Using `screen`



Disconnected screen session **keeps running** on its own, with everything within it.

Using `screen`


Linux shell (ssh session)
cbsu1 ~$ screen -d -r
While logged in through ssh….
…re-attach the screen session
screen session
cbsu1 ~$ cd /dir1
shell 1 (Ctrl-d to close)
cbsu1 ~$ blastn
shell 2 (Ctrl-d to close)
cbsu1 ~$ ls -al
shell 3 (Ctrl-d to close)

Re-attach the screen session using **`screen -d -r`**

Prior to re-attaching, verify the session is running: **`screen -list`**

Will see all shells as we left them, and progress of any programs we left running

screen: running multiple shells in one window

After logging in, type `screen`

Most useful `screen` commands:

Screen command	What it does
screen -S ABC	Start a new session named ABC ('-S ABC' optional)
screen -list	List all your screen sessions
screen -d -r screen -d -r [sessionID]	Re-attach previously detached (or unintentionally disconnected) session – can be done upon next login
Ctrl-a c	Create a new shell in a session; can be repeated multiple times
Ctrl-a n Ctrl-a p Ctrl-a N	Switch to next (n), previous (p), or N-th shell within a session
Ctrl-a "	List all shells in a session, switch to one (arrows, ENTER)
Ctrl-a S Ctrl-a |	Split window horizontally (S) or vertically (|), then 'Ctrl-a TAB' to jump to new split and 'Ctrl-a c', 'Ctrl-a n', 'Ctrl-a p', or 'Ctrl-a N' to create or import a shell to it; this will show 2 or more shells in one window
Ctrl-a TAB	Jump between shells in a split window
Ctrl-a X Ctrl-a Q	Remove current shell (X), or all shells except current (Q) from split window; removed shells will keep running (use Ctrl-a N, Ctrl-a n, or Ctrl-a p to access)
Ctrl-a d	Detach a session (all shells will continue running)
Ctrl-d	Exit form current shell (or from whole session, if in last shell)
screen -X -S [name] quit	Kill session "name" (obtained from screen -list)

For more features/functionality – type screen –h or Ctrl-a ? (within session)

Sessions are persistent – will survive connection problems, turning off laptop, etc.

Graphics on Linux workstations

http://biohpc.cornell.edu/lab/doc/Remote_access.pdf

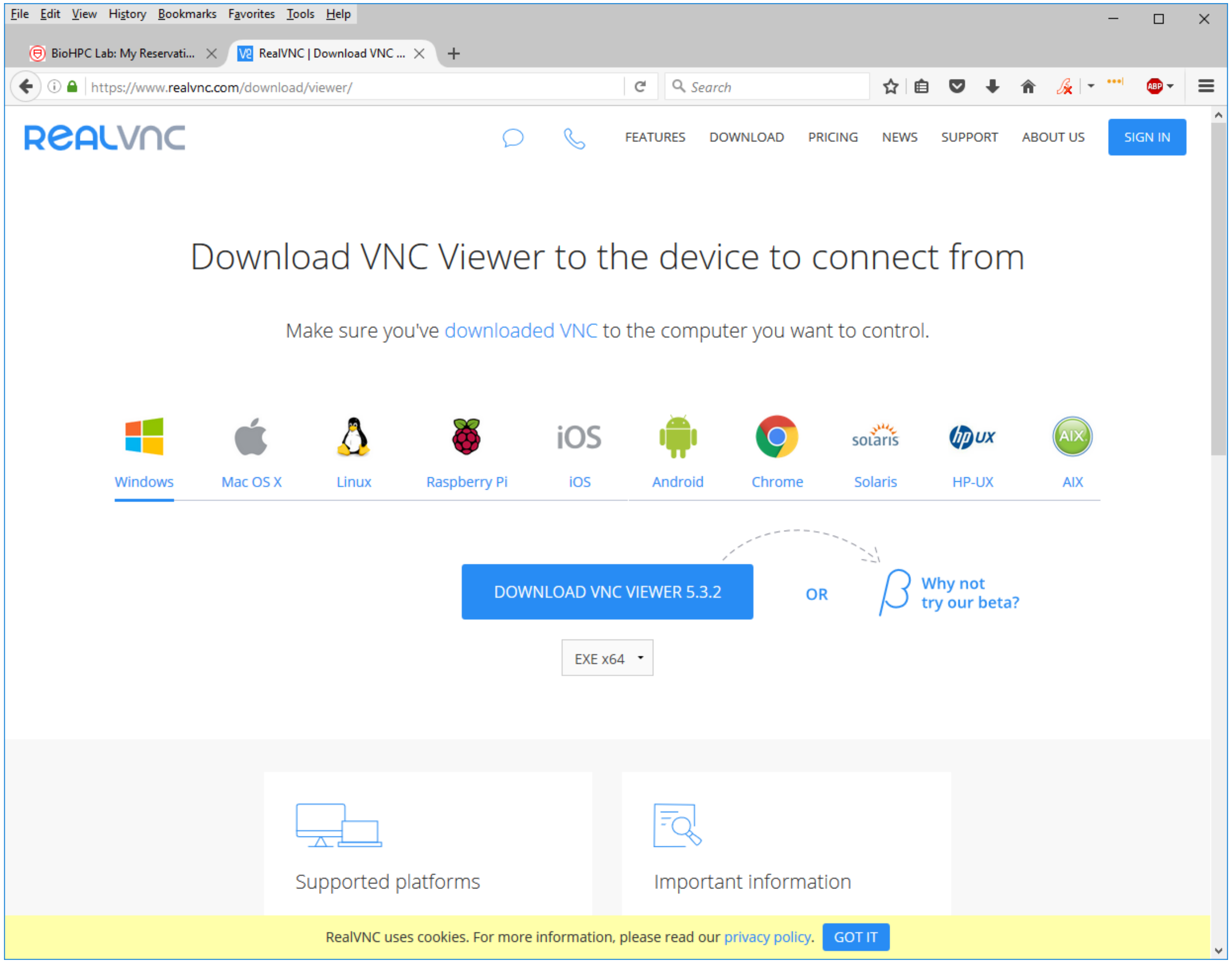
In short, there are two options:

- Log in through ssh with **X11 forwarding** (check option in PuTTy, or **`ssh -Y`** on a Mac). The laptop must be running an **X-windows manager**. Start GUI application in ssh terminal, and the GUI window will appear on your laptop screen. Individual GUI windows are rendered this way.

- Log in to a Linux **graphical mode** using **VNC** (Virtual Network Computing)
  - Start a **VNC server** on Linux machine (typically installed by default)
  - Download and start a **VNC client** on your laptop, connect to VNC server on Linux machine
  - Your laptop will display **whole Linux graphical desktop** (similar to a Windows or Mac desktop)
  - VNC session is **persistent**, just like ‘screen’ session

Logging in to a Linux workstation (GUI)

You need software client to connect to your machine via VNC.
We recommend **RealVNC VNC Viewer** for all platforms.


File Edit View History Bookmarks Favorites Tools Help
BioHPC Lab: My Reservati...
RealVNC | Download VNC ...
https://www.realvnc.com/download/viewer/
Search
REALVNC
FEATURES DOWNLOAD PRICING NEWS SUPPORT ABOUT US
SIGN IN

Download VNC Viewer to the device to connect from

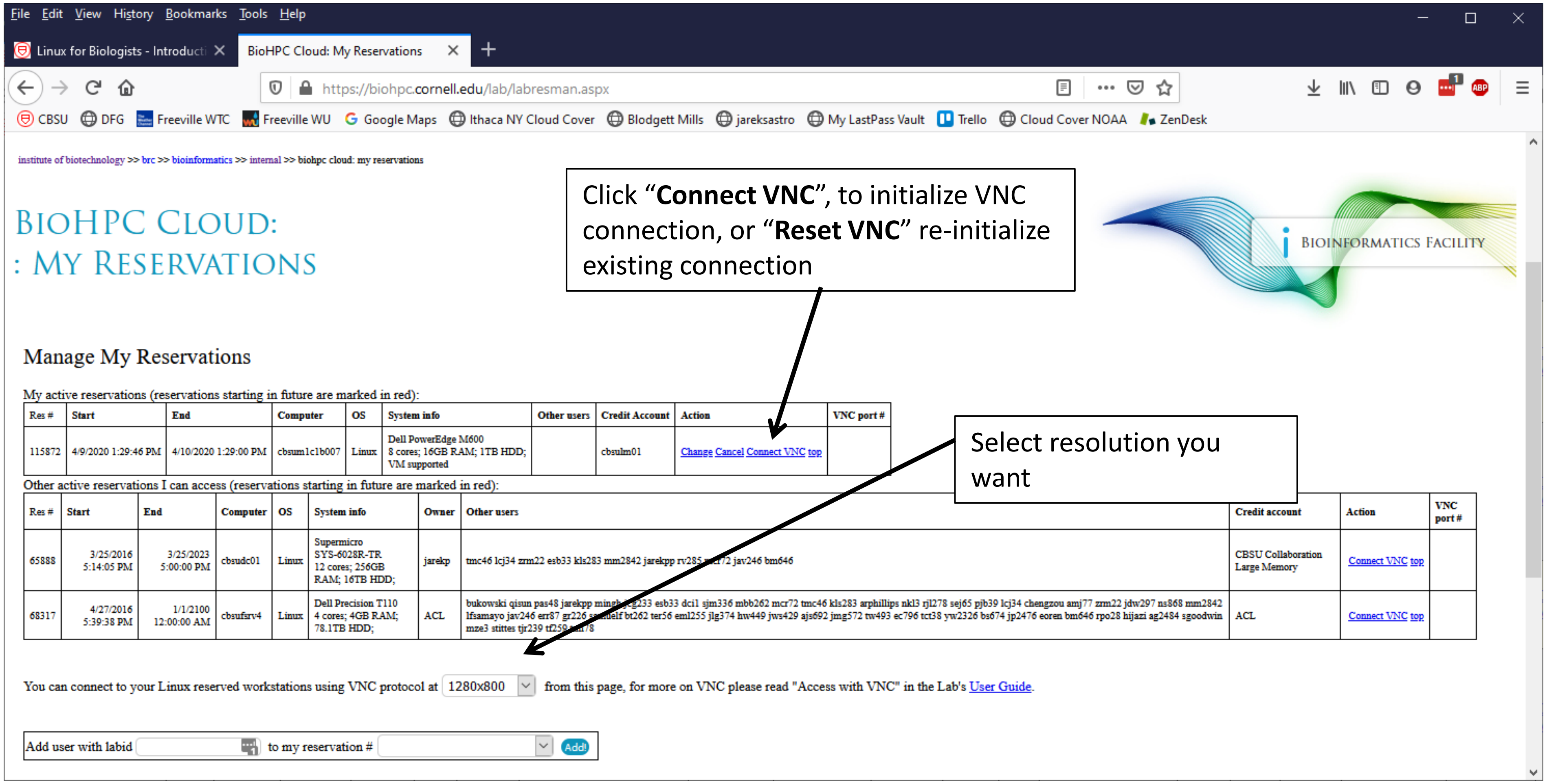
Make sure you've downloaded VNC to the computer you want to control.

Windows
Mac OS X
Linux
Raspberry Pi
iOS
Android
Chrome
Solaris
HP-UX
AIX
DOWNLOAD VNC VIEWER 5.3.2
OR
Why not try our beta?
EXE x64
Supported platforms
Important information

RealVNC uses cookies. For more information, please read our privacy policy. GOT IT

Logging in to a Linux workstation via VNC client
(GUI)

In web browser, navigate to http://biohpc.cornell.edu/, log in (if not yet logged in), click on **User:your_id**, select tab **My Reservations**


File Edit View History Bookmarks Tools Help
Linux for Biologists - Introducti
BioHPC Cloud: My Reservations
https://biohpc.cornell.edu/lab/labresman.aspx
CBSU DFG Freeville WTC Freeville WU Google Maps Ithaca NY Cloud Cover Blodgett Mills jareksastro My LastPass Vault Trello Cloud Cover NOAA ZenDesk
BIOINFORMATICS FACILITY

institute of biotechnology >> brc >> bioinformatics >> internal >> biohpc cloud: my reservations

BioHPC Cloud: : My Reservations

Click “**Connect VNC**”, to initialize VNC connection, or “**Reset VNC**” re-initialize existing connection

Manage My Reservations

My active reservations (reservations starting in future are marked in red):

Res #	Start	End	Computer	OS	System info	Other users	Credit Account	Action	VNC port #
115872	4/9/2020 1:29:46 PM	4/10/2020 1:29:00 PM	cbsum1c1b007	Linux	Dell PowerEdge M600 8 cores; 16GB RAM; 1TB HDD; VM supported		cbsulm01	Change Cancel Connect VNC top	

Select resolution you want

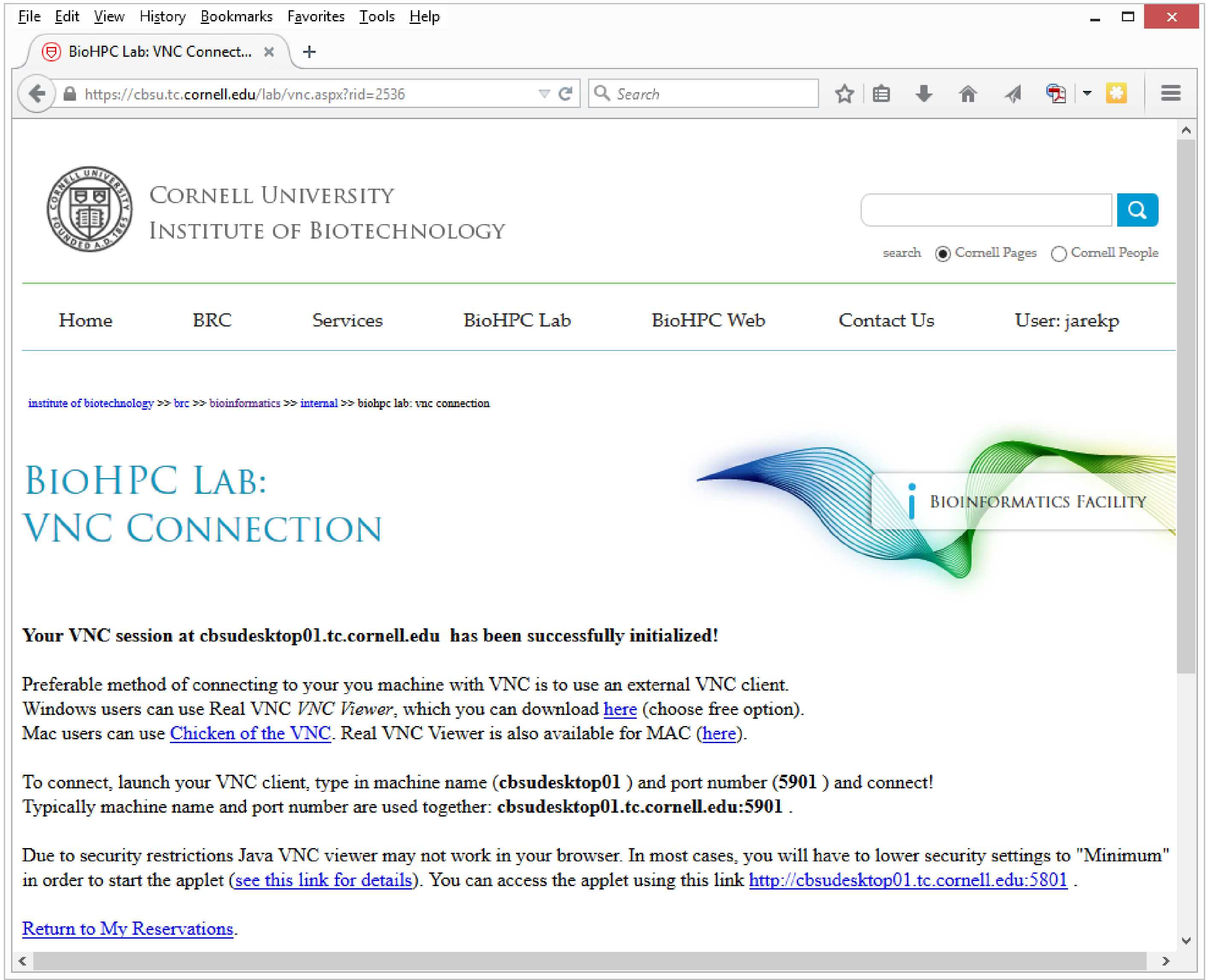
Other active reservations I can access (reservations starting in future are marked in red):

Res #	Start	End	Computer	OS	System info	Owner	Other users	Credit account	Action	VNC port #
65888	3/25/2016 5:14:05 PM	3/25/2023 5:00:00 PM	cbsudc01	Linux	Supermicro SYS-6028R-TR 12 cores; 256GB RAM; 16TB HDD;	jarekp	tmc46 lcj34 zrm22 esb33 kls283 mm2842 jarekpp rv285 mcr72 jav246 bm646	CBSU Collaboration Large Memory	Connect VNC top	
68317	4/27/2016 5:39:38 PM	1/1/2100 12:00:00 AM	cbsufsrv4	Linux	Dell Precision T110 4 cores; 4GB RAM; 78.1TB HDD;	ACL	bukowski qisun pas48 jarekpp mingh [illegible]g233 esb33 dcil sjm336 mbb262 mcr72 tmc46 kls283 arphillips nkl3 rjl278 sej65 pjb39 lcj34 chengzou amj77 zrm22 jdw297 ns868 mm2842 lfsamayo jav246 err87 gr226 [illegible]elf bt262 ter56 eml255 jlg374 hw449 jws429 ajs692 jmg572 tw493 ec796 tct38 yw2326 bs674 jp2476 eoren bm646 rpo28 hijazi ag2484 sgoodwin mze3 stittes tjr239 tf259 [illegible]78	ACL	Connect VNC top	

You can connect to your Linux reserved workstations using VNC protocol at 1280x800 from this page, for more on VNC please read "Access with VNC" in the Lab's User Guide.

Add user with labid ______ to my reservation # ______ Add!

Logging in to a Linux workstation
(GUI)


File Edit View History Bookmarks Favorites Tools Help
BioHPC Lab: VNC Connect...
https://cbsu.tc.cornell.edu/lab/vnc.aspx?rid=2536
Search
Cornell University
Institute of Biotechnology
search Cornell Pages Cornell People
Home BRC Services BioHPC Lab BioHPC Web Contact Us User: jarekp
institute of biotechnology >> brc >> bioinformatics >> internal >> biohpc lab: vnc connection
Bioinformatics Facility

BioHPC Lab:
VNC Connection

Your VNC session at cbsudesktop01.tc.cornell.edu has been successfully initialized!

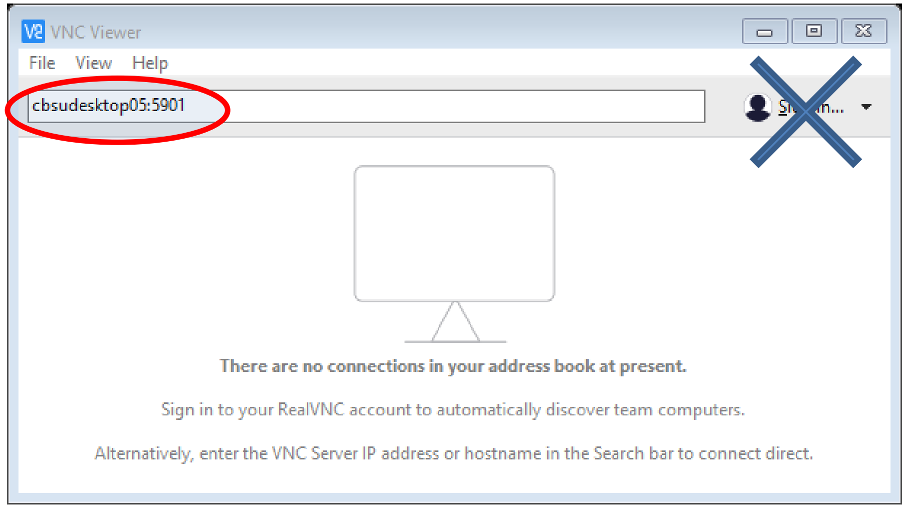
Preferable method of connecting to your you machine with VNC is to use an external VNC client.
Windows users can use Real VNC *VNC Viewer*, which you can download here (choose free option).
Mac users can use Chicken of the VNC. Real VNC Viewer is also available for MAC (here).

To connect, launch your VNC client, type in machine name (**cbsudesktop01**) and port number (**5901**) and connect!
Typically machine name and port number are used together: **cbsudesktop01.tc.cornell.edu:5901** .

Due to security restrictions Java VNC viewer may not work in your browser. In most cases, you will have to lower security settings to "Minimum" in order to start the applet (see this link for details). You can access the applet using this link http://cbsudesktop01.tc.cornell.edu:5801 .

Return to My Reservations.

VNC: starting the client and logging in


VNC Viewer

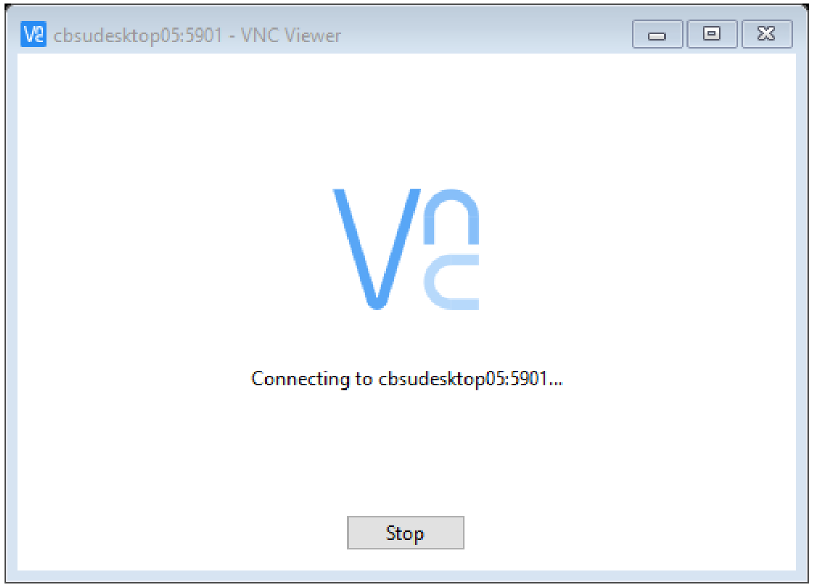
File View Help

cbsudesktop05:5901

There are no connections in your address book at present.

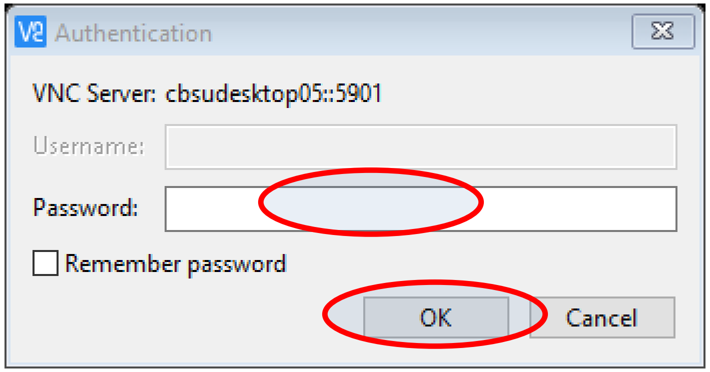
Sign in to your RealVNC account to automatically discover team computers.

Alternatively, enter the VNC Server IP address or hostname in the Search bar to connect direct.


cbsudesktop05:5901 - VNC Viewer

Connecting to cbsudesktop05:5901...

Stop


Authentication

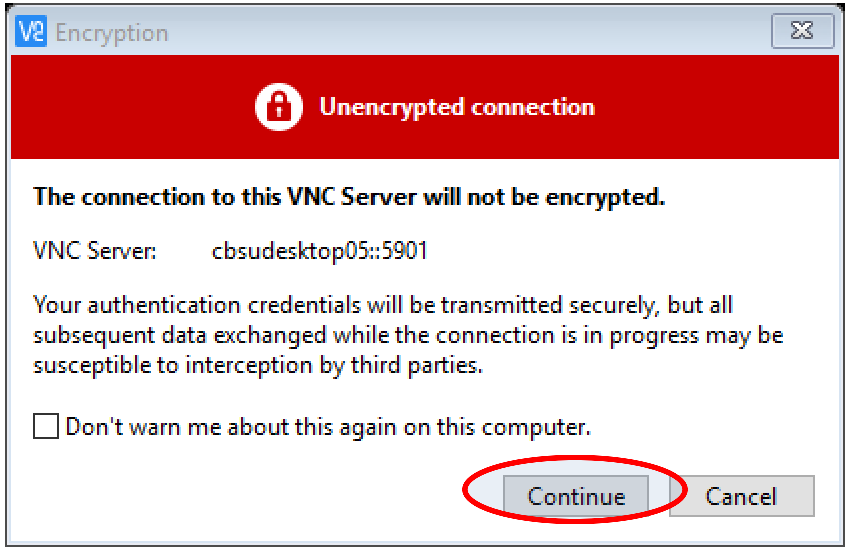
VNC Server: cbsudesktop05::5901

Username:

Password:

Remember password

OK Cancel


Encryption

Unencrypted connection

The connection to this VNC Server will not be encrypted.

VNC Server: cbsudesktop05::5901

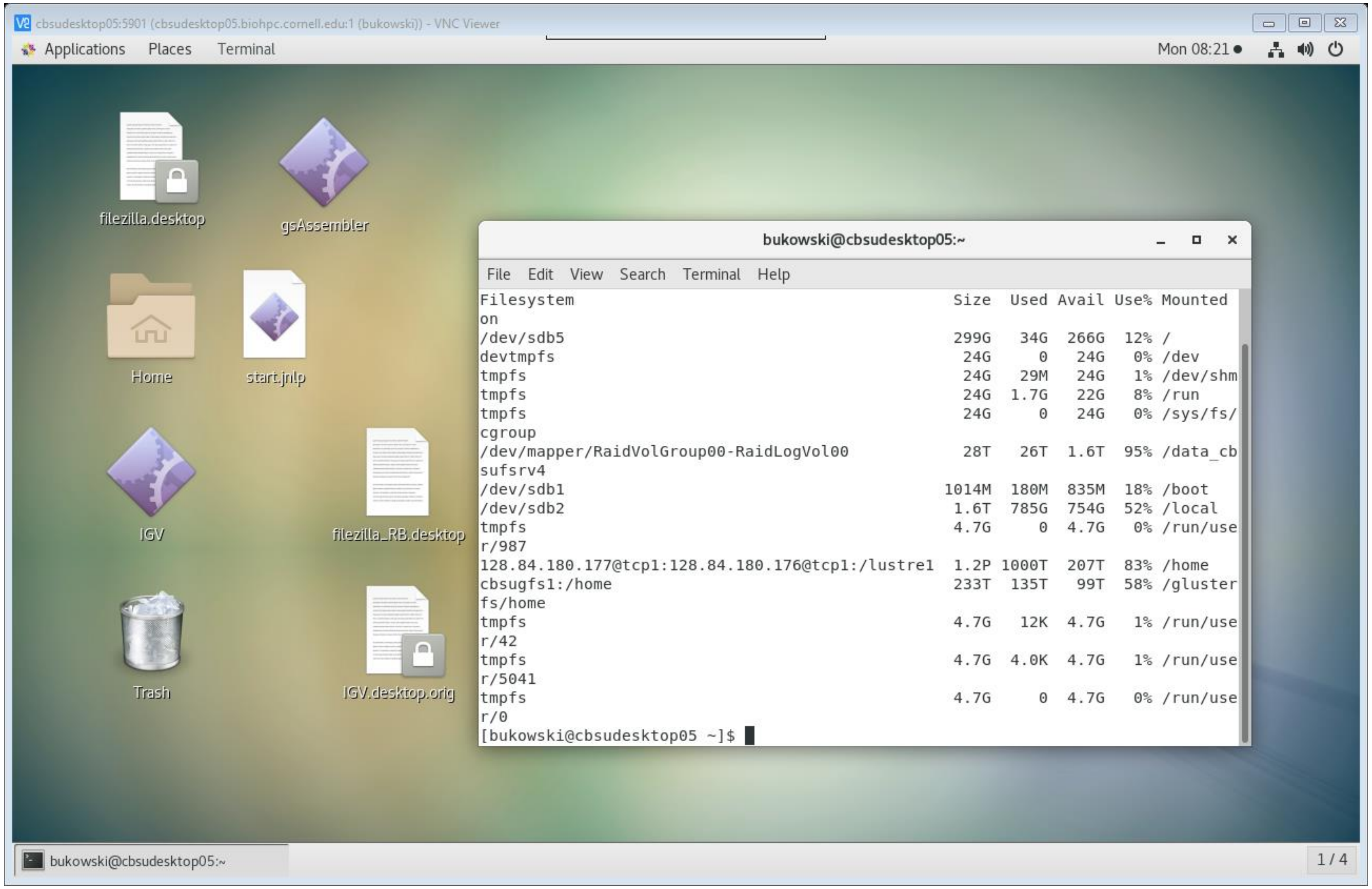
Your authentication credentials will be transmitted securely, but all subsequent data exchanged while the connection is in progress may be susceptible to interception by third parties.

Don't warn me about this again on this computer.

Continue Cancel

VNC: logged in

Right-click anywhere within blue desktop, select **Open Terminal** …. or
…. click **Applications -> Accessories -> Terminal**

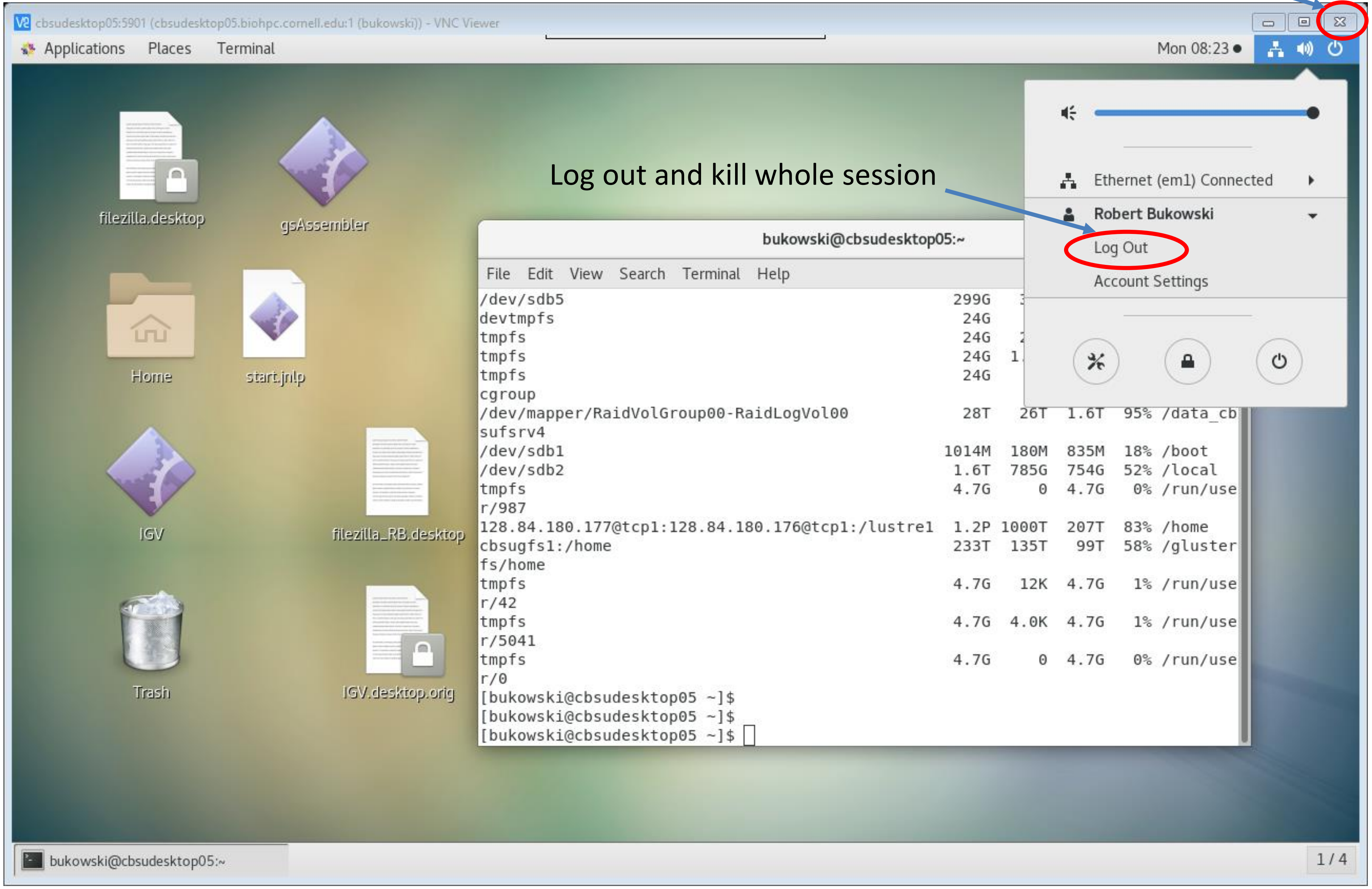


```
Filesystem                                              Size  Used Avail Use% Mounted
on
/dev/sdb5                                               299G   34G  266G  12% /
devtmpfs                                                 24G     0   24G   0% /dev
tmpfs                                                    24G   29M   24G   1% /dev/shm
tmpfs                                                    24G  1.7G   22G   8% /run
tmpfs                                                    24G     0   24G   0% /sys/fs/
cgroup
/dev/mapper/RaidVolGroup00-RaidLogVol00                  28T   26T  1.6T  95% /data_cb
sufsrv4
/dev/sdb1                                              1014M  180M  835M  18% /boot
/dev/sdb2                                               1.6T  785G  754G  52% /local
tmpfs                                                   4.7G     0  4.7G   0% /run/use
r/987
128.84.180.177@tcp1:128.84.180.176@tcp1:/lustre1        1.2P 1000T  207T  83% /home
cbsugfs1:/home                                          233T  135T   99T  58% /gluster
fs/home
tmpfs                                                   4.7G   12K  4.7G   1% /run/use
r/42
tmpfs                                                   4.7G  4.0K  4.7G   1% /run/use
r/5041
tmpfs                                                   4.7G     0  4.7G   0% /run/use
r/0
[bukowski@cbsudesktop05 ~]$
```

VNC: two ways to exit

Kill window, but session keeps running – can re-connect later

Log out and kill whole session



```
/dev/sdb5                                          299G   3
devtmpfs                                            24G
tmpfs                                               24G   2
tmpfs                                               24G   1.
tmpfs                                               24G
cgroup
/dev/mapper/RaidVolGroup00-RaidLogVol00             28T   26T  1.6T  95% /data_cb
sufsrv4
/dev/sdb1                                        1014M  180M  835M  18% /boot
/dev/sdb2                                          1.6T  785G  754G  52% /local
tmpfs                                              4.7G     0  4.7G   0% /run/use
r/987
128.84.180.177@tcp1:128.84.180.176@tcp1:/lustre1  1.2P 1000T  207T  83% /home
cbsugfs1:/home                                     233T  135T   99T  58% /gluster
fs/home
tmpfs                                              4.7G   12K  4.7G   1% /run/use
r/42
tmpfs                                              4.7G  4.0K  4.7G   1% /run/use
r/5041
tmpfs                                              4.7G     0  4.7G   0% /run/use
r/0
[bukowski@cbsudesktop05 ~]$
[bukowski@cbsudesktop05 ~]$
[bukowski@cbsudesktop05 ~]$
```

Connecting with VNC form external network without VPN Mac and Linux

- Enable your VNC connection first (see slide 82)
- Open local terminal window on your Mac or Linux computer
- Use the following command to connect to BioHPC. You can replace cbsulogin with cbsulogin2 or cbsulogin3, cbsuxxx with your server name, 5901 with your port no and biohpcid with your BioHPC userid.

```
ssh -N -L 5901:cbsuxxx:5901 biohpcid@cbsulogin.biohpc.cornell.edu
```

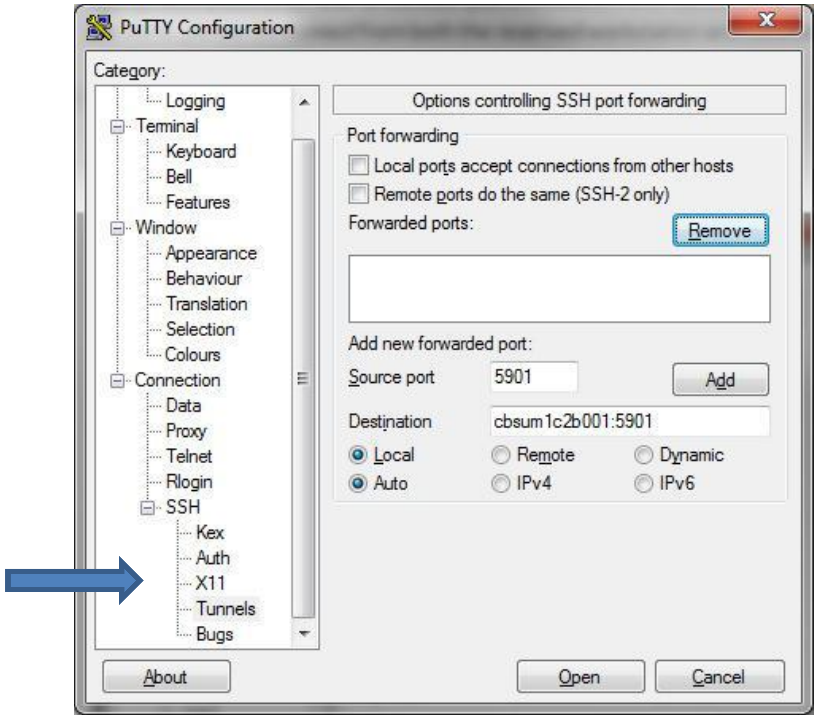
- Now you can connect to your VNC by typing localhost:5901 in your VNC Viewer software.

Connecting with VNC form external network without VPN Windows

- Enable your VNC connection first (see slide 82). Note what is your VNC port.

- Open your PuTTY and fill out cbsulogin.biohpc.cornell.edu (or cbsulogin2 or cbsulogin3) as target server.

- On the left panel scroll down to Connection -> SSH -> Tunnels



Options controlling SSH port forwarding

Port forwarding

☐ Local ports accept connections from other hosts

☐ Remote ports do the same (SSH-2 only)

Forwarded ports:

Add new forwarded port:

Source port: 5901

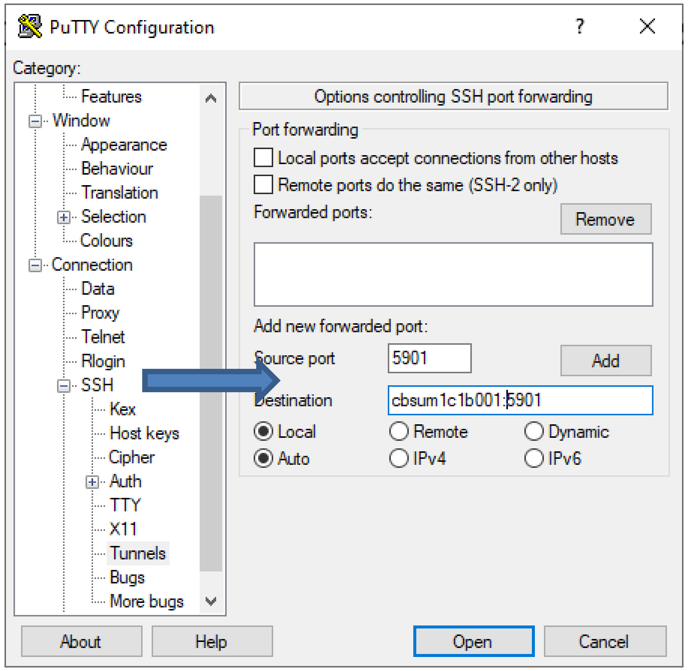
Destination: cbsum1c2b001:5901

[✓ Local] Remote Dynamic

[✓ Auto] IPv4 IPv6

Connecting with VNC form external network without VPN Windows

Enable your VNC connection first (see slide 82). Note what is your VNC port. Type the port as shown below with the destinations server name and click Add. Now you can connect to your VNC by typing localhost:5901 in your VNC Viewer software.



Options controlling SSH port forwarding

Port forwarding

- [ ] Local ports accept connections from other hosts
- [ ] Remote ports do the same (SSH-2 only)

Forwarded ports: Remove

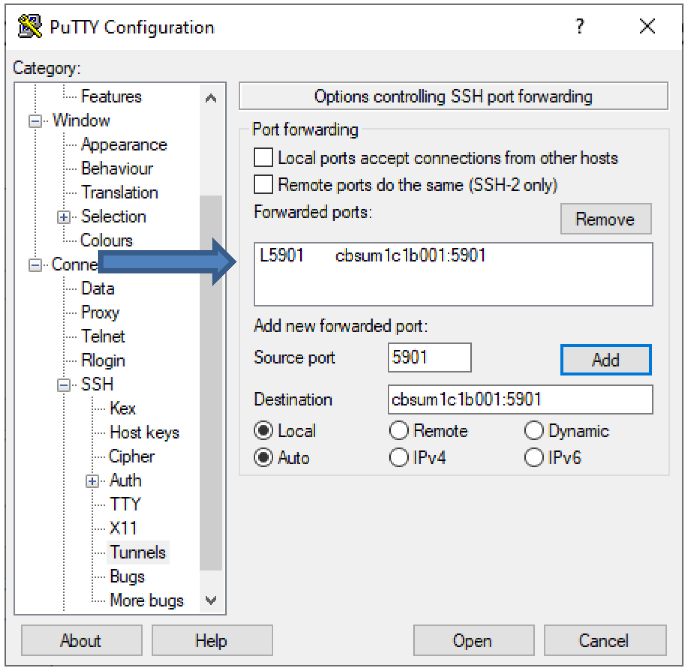
Add new forwarded port:

Source port: 5901 Add

Destination: cbsum1c1b001:5901

[✓ Local] Remote Dynamic

[✓ Auto] IPv4 IPv6



Options controlling SSH port forwarding

Port forwarding

- [ ] Local ports accept connections from other hosts
- [ ] Remote ports do the same (SSH-2 only)

Forwarded ports: Remove

```
L5901     cbsum1c1b001:5901
```

Add new forwarded port:

Source port: 5901 Add

Destination: cbsum1c1b001:5901

[✓ Local] Remote Dynamic

[✓ Auto] IPv4 IPv6

VNC: summary

VNC sessions are *persistent* (similar to *screen*)

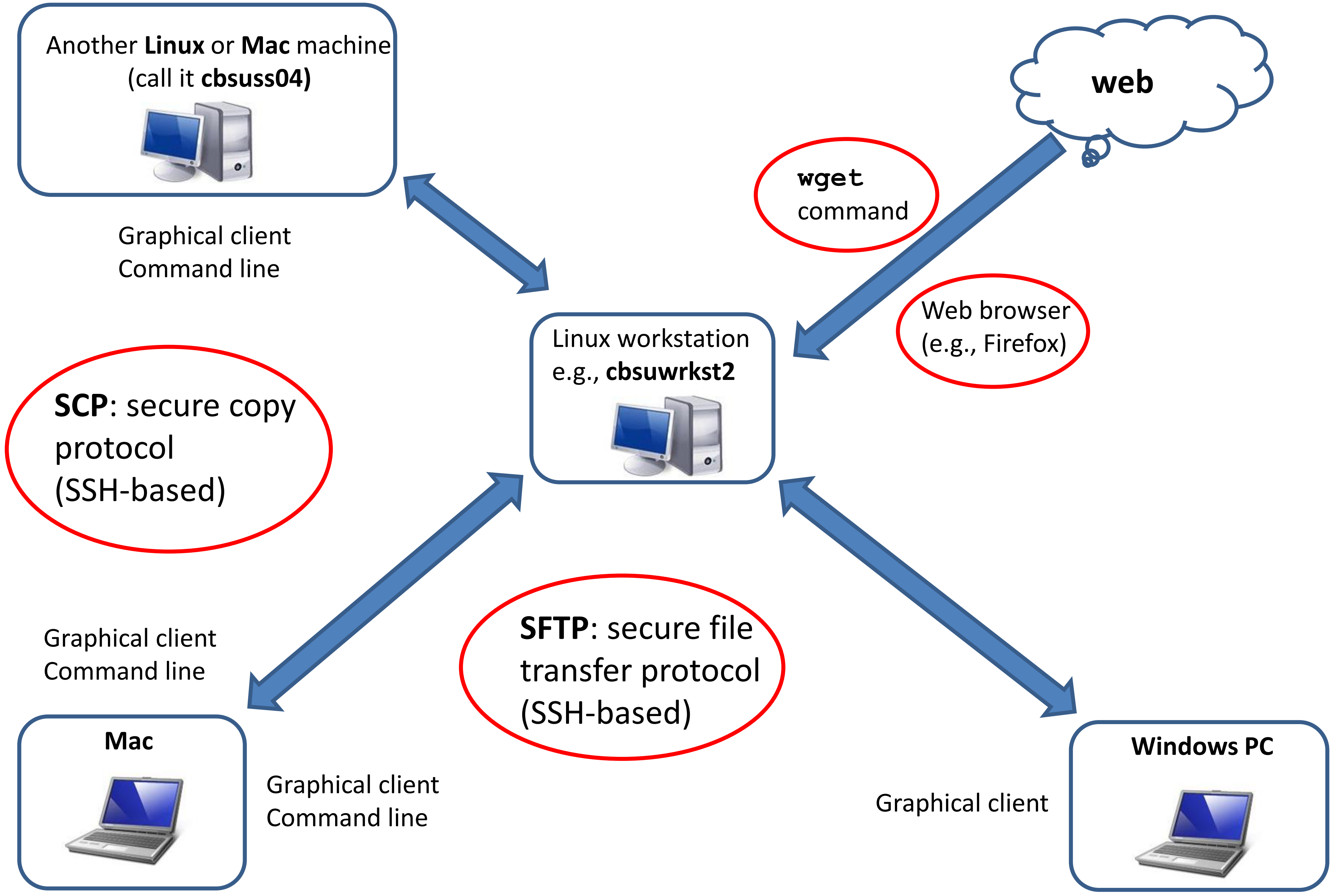
They run even when the client is disconnected

If you need to reset the session you need to use "Reset VNC" link

Equivalent to Windows Remote Desktop

File transfer

File Transfer: overview


Another **Linux** or **Mac** machine (call it **cbsuss04)**
Graphical client
Command line
web
wget command
Web browser (e.g., Firefox)
Linux workstation e.g., **cbsuwrkst2**
SCP: secure copy protocol (SSH-based)
Graphical client
Command line
SFTP: secure file transfer protocol (SSH-based)
Mac
Graphical client
Command line
Graphical client
Windows PC

File transfer: (some) graphical clients

Client	Windows	Mac	Linux
FileZilla	x	x	x
WinScp	x		
Cyberduck	x	x	
CuteFTP	x	x	
Transmit		x	
Free FTP	x		

recommended

- All clients feature
  - File explorer-like graphical interface to files on both the PC and on the Linux machine
  - Drag-and-drop functionality

- When connecting to a Linux machine from a client, use the **sftp** protocol (or **port 22**). You will be asked for your user name and password (the same you use to log in to the BioHPC workstations).

- Transfer text file in text mode, binary files in binary mode (the default "Auto" should be right, but…).

Fixing line ending problems

Files transferred **to Linux machine from a Windows or Mac machine** often have line endings incompatible with Linux (depends on transfer software used and its settings)

To fix line endings, use **`dos2unix`** command

```
dos2unix my_file
```

```
mac2unix my_file
```

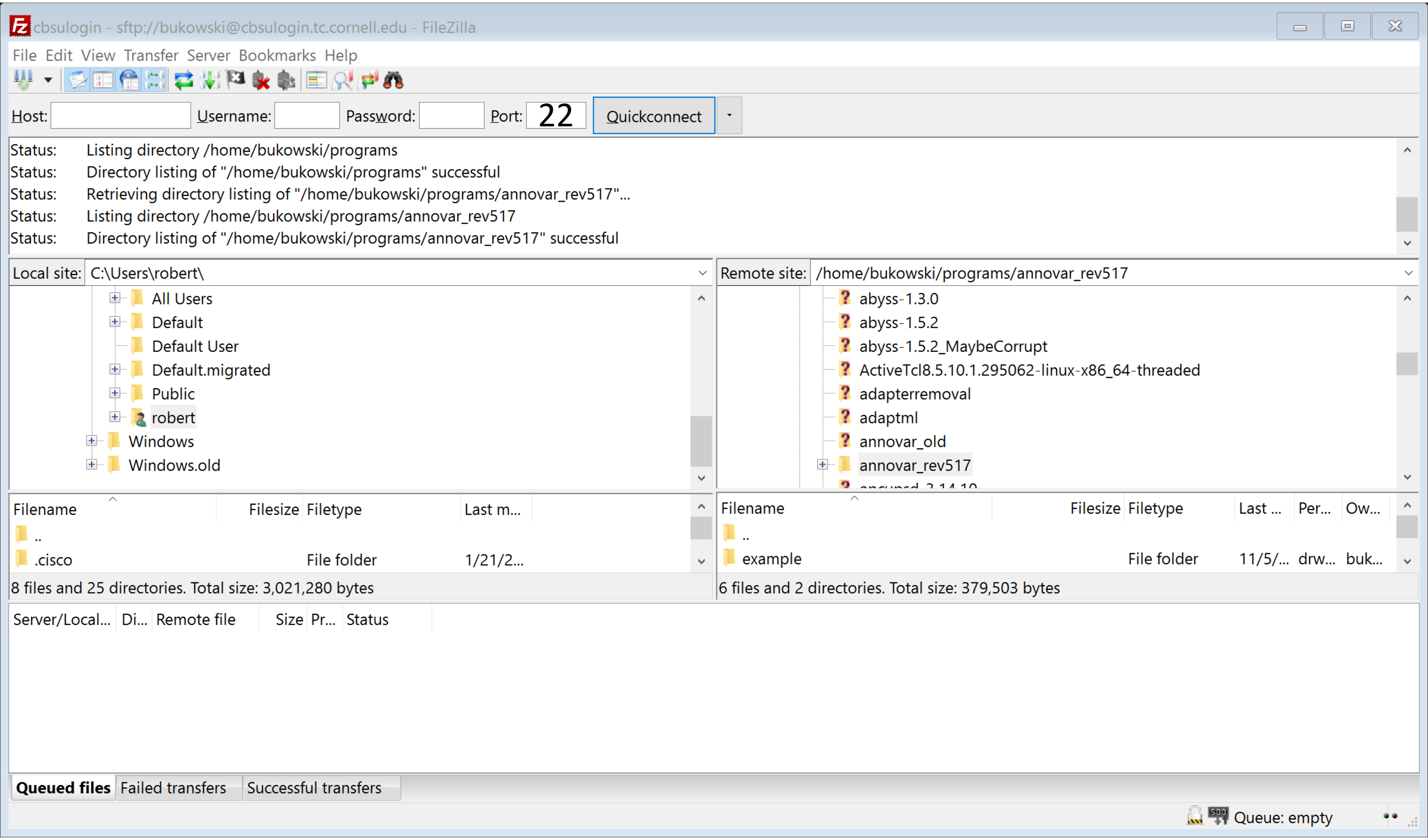
(the file **`my_file`** *will have linux line endings)*

```
dos2unix -n my_file my_file_converted
```

```
mac2unix -n my_file  my_file_converted
```

(the file **`my_file_converted`** *will have linux line endings, the original file* **`my_file`** *will be kept)*

FileZilla window



cbsulogin - sftp://bukowski@cbsulogin.tc.cornell.edu - FileZilla
File Edit View Transfer Server Bookmarks Help
Host:
Username:
Password:
Port: 22
Quickconnect

```
Status:    Listing directory /home/bukowski/programs
Status:    Directory listing of "/home/bukowski/programs" successful
Status:    Retrieving directory listing of "/home/bukowski/programs/annovar_rev517"...
Status:    Listing directory /home/bukowski/programs/annovar_rev517
Status:    Directory listing of "/home/bukowski/programs/annovar_rev517" successful
```

Local site: C:\Users\robert\

- All Users
- Default
- Default User
- Default.migrated
- Public
- robert
- Windows
- Windows.old

Filename	Filesize	Filetype	Last m...
..			
.cisco		File folder	1/21/2...

8 files and 25 directories. Total size: 3,021,280 bytes

Remote site: /home/bukowski/programs/annovar_rev517

- abyss-1.3.0
- abyss-1.5.2
- abyss-1.5.2_MaybeCorrupt
- ActiveTcl8.5.10.1.295062-linux-x86_64-threaded
- adapterremoval
- adaptml
- annovar_old
- annovar_rev517

Filename	Filesize	Filetype	Last ...	Per...	Ow...
..					
example		File folder	11/5/...	drw...	buk...

6 files and 2 directories. Total size: 379,503 bytes

Server/Local...	Di...	Remote file	Size	Pr...	Status

File transfer: command-line scp

Linux <-> Linux, Mac <-> Linux

Objective: copy a file **/data/reads/my_sequence.fa** from the local Linux or Mac machine to directory **/workdir/files** on a remote Linux machine called **cbsuwrkst2.biohpc.cornell.edu**

While logged in on the local machine, execute:

```
cd /data/reads
```

```
scp my_sequence.fa bukowski@cbsuwrkst2.biohpc.cornell.edu:/workdir/files
```

To copy in the opposite direction:

```
scp bukowski@cbsuwrkst2.biohpc.cornell.edu:/workdir/files/my_sequence.fa
.
```

NOTES:

- **scp** is a generalization of **cp**, where the source or the target file is on a remote machine
- Most **cp** options work with **scp** (**scp -r** will recursively copy whole directory)
- The remote machine must accept connection requests (depends on network config)

File transfer: from the web to Linux

Option 1: use a web browser (such as Firefox)

- Connect to Linux machine in **graphical mode (VNC)**
- Start Firefox (in terminal window, type **firefox**, or click on web browser icon)
  - **Note**: the web browser is running on Linux machine, not on your laptop!
- Navigate to desired site and download the file (will ask for directory in which to deposit the file)

Let's try to download the following file:

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

File transfer: from the web to Linux

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

- **Example 1: simple URL**

```
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)

- **Example 2: complicated URL**

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

(whole command should be on one line; note the **“”** *marks around the link and the* **–O** *option which specifies the name you want to give the downloaded file)*

File transfer: from the web to Linux

Example 3: Downloading Illumina sequencing results

Fragment of a notification e-mail from Cornell Genomics Facility:

Sample: **P_Teo_10_b**
File: **6581_7527_30809_C877GANXX_P_Teo_10_b_R1.fastq.gz**
Size **18570118164 bytes**, MD5: **118c0c974a6c4dd81895c26cdd4208e6**
Link: http://cbsuapps.biohpc.cornell.edu/Sequencing/showseqfile.aspx?mode=http&cntrl=94863491&refid=93804

Sample: **P_Teo_11_b**
File: **6582_7527_30810_C877GANXX_P_Teo_11_b_R1.fastq.gz**
Size **17854406437 bytes**, MD5: **20be4a4305b6a2f3260c461536bbf060**
Link: http://cbsuapps.biohpc.cornell.edu/Sequencing/showseqfile.aspx?mode=http&cntrl=1244420836&refid=93805

e.t.c.

How to get these files onto a Linux machine?

How to get the sequencing files onto a Linux machine?

1. Open **Firefox** (it's on a Linux machine, so need to be logged in through VNC) and navigate to each URL – very tedious if the number of files large

2. Use **`wget`** commands (provided in the notification e-mail as attachment file **`download.sh`**)

A couple of lines from the attached file **`download.sh`** (typically there is more than two wget commands – may be several hundred!):

```
wget -q -c -O 6581_7527_30809_C877GANXX_P_Teo_10_b_R1.fastq.gz
http://cbsuapps.biohpc.cornell.edu/Sequencing/showseqfile.aspx?mode=http&cntrl=
94863491&refid=93804

wget -q -c -O 6582_7527_30810_C877GANXX_P_Teo_11_b_R1.fastq.gz
http://cbsuapps.biohpc.cornell.edu/Sequencing/showseqfile.aspx?mode=http&cntrl=
1244420836&refid=93805
```

Transfer this file to your Linux machine and execute it as **shell script**:

```
sh ./download.sh
```

Transferring large numbers of small files

There is a serious time overhead when handling large number of small files

Example: 1 million files 1 Mbyte each

Network bandwidth 150 MBytes/sec → expected transfer time 1.8 hours
actual transfer time: 1-2 days !

Remedy:

Create several **tar archives**, about 100 GBytes each, each containing a different subset of original small files, then transfer those tar files, one by one, ‘untar’ at destination

a single big (1 TByte) tar archive would work as well, but more time would be wasted if transfer is interrupted for any reason and has to be restarted from the beginning

Running applications

Running applications

- Very much like running system commands

- (Very) general syntax

```
<path_to_application_executable> <options>
```

- A few quick examples:

```
blastall  -p blastx -b 1 -d ./databases/swissprot -i seq_tst.fa
```

```
samtools  flagstat alignments.bam
```

```
tophat  -p 7 -o B_L1-1 --transcriptome-index  ZmB73_5a_WGS \
--no-novel-juncs  genome/maize reads_R1.fastq.gz reads_R2.fastq.gz
```

Running applications

- Why can we call, say, **samtools** by just typing **`samtools`** rather than the full path (in this case, **`/programs/bin/samtools/samtools)`** ?
  - Because of the search path environment variable which is defined for everybody. When you type **`samtools`**, the system tries each directory on the search path one by one until it finds the corresponding executable.
  - `which samtools` *(tells us where on disk the command* bwa *is located)*
  - `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`**
  - **Note:** majority of executables **are NOT in search path** – they need to be launched using **full path**.
    - Visit https://biohpc.cornell.edu/lab/labsoftware.aspx to find out the path to your application

Running applications

- <u>How to run **Java** applications</u>?
- Java programs usually come packaged in so-called **jars**
- Java program is launched by running the **java virtual machine** with the appropriate **jar** as an argument
- Example:

Launch Java with 6GB of RAM

Run program from this jar

```
java -Xmx6g -jar GenomeAnalysisTK.jar -T UnifiedGenotyper \
-R genome.fa -i aln.bam -o variants.vcf
```

Program options

Running applications

- ❑ Need to know what program(s) are relevant for your particular problem

- ❑ Need to know what a given program does and how to use it

  - Visit our software page http://biohpc.cornell.edu/lab/labsoftware.aspx

    - Links to manuals (all options explained, examples given, test data available)

    - Specific hints on running in BioHPC environment

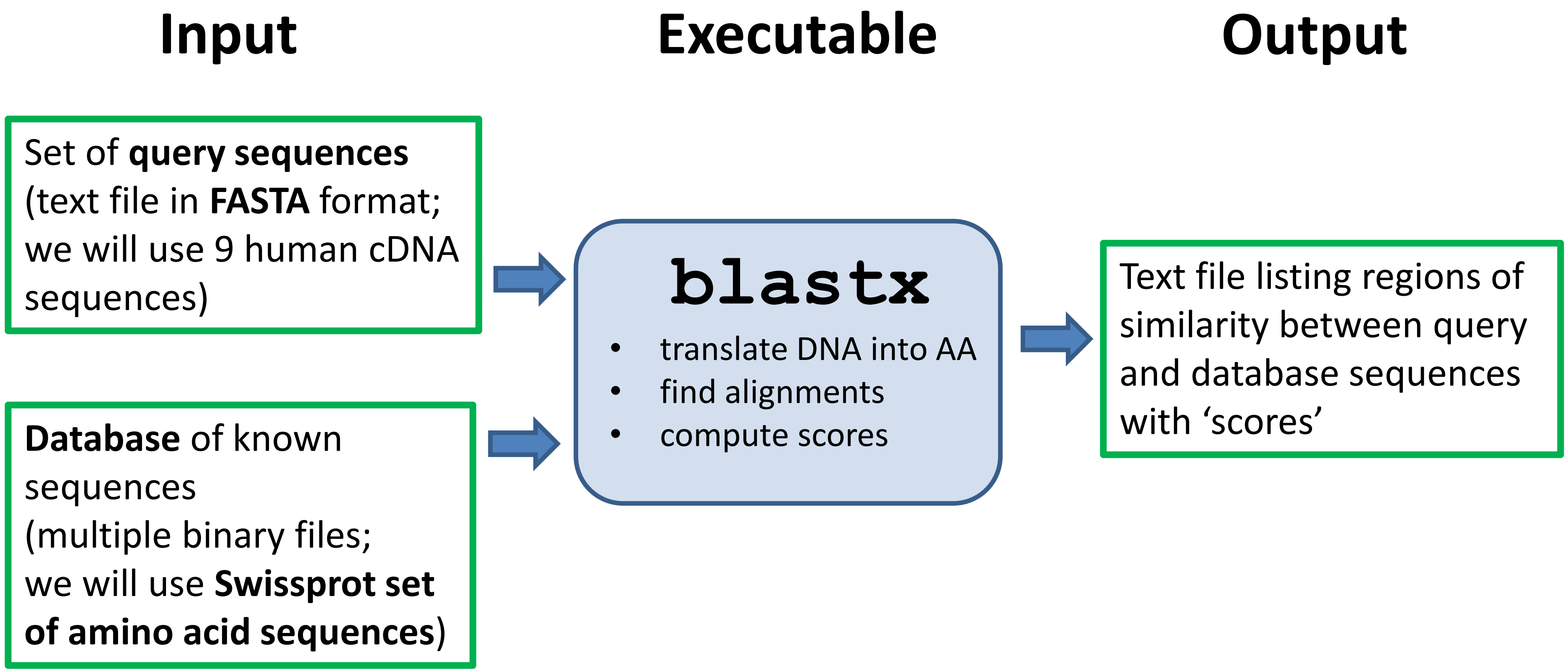
- ❑ Getting quick help – run **command without any options**, or sometimes with **`–h`** or **`--help`**

  - Should print a list of options with very short descriptions

Example: BLAST

Basic Local Alignment Search Tool

BLAST finds regions of similarity between biological sequences. The program compares nucleotide or protein sequences to **sequence databases** and calculates the statistical significance.



Input

Set of **query sequences** (text file in **FASTA** format; we will use 9 human cDNA sequences)

Database of known sequences (multiple binary files; we will use **Swissprot set of amino acid sequences**)

Executable

`blastx`

- translate DNA into AA
- find alignments
- compute scores

Output

Text file listing regions of similarity between query and database sequences with ‘scores’

Running applications example: BLAST

prepare input

- ❑ Create your local scratch directory (if not yet done) and a sub-directory **`blast_test`** where this exercise will be run

```
mkdir /workdir/bukowski
cd /workdir/bukowski
mkdir blast_test
cd blast_test
```

- ❑ Copy file with query sequences to the exercise directory:

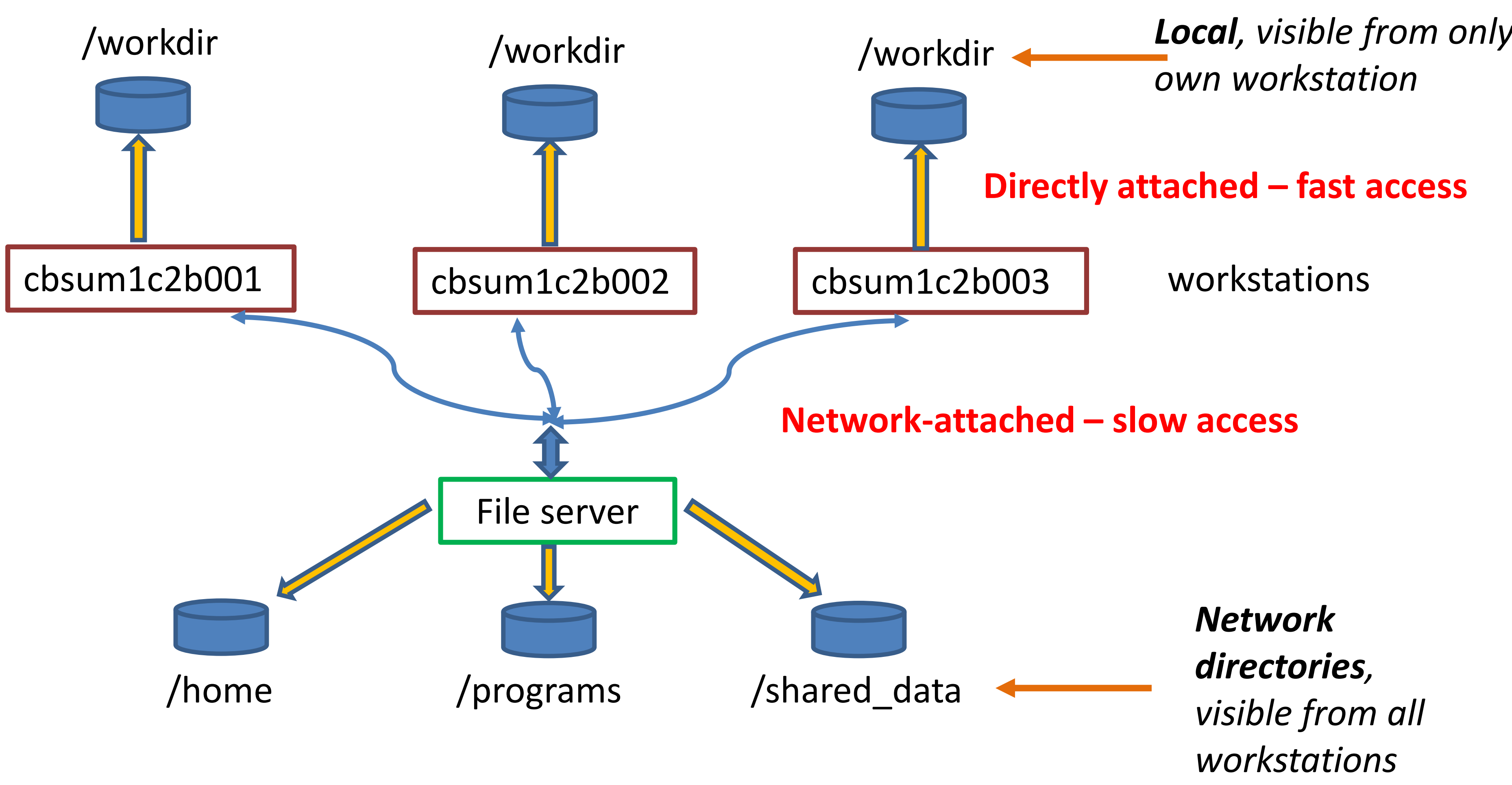
```
cp /shared_data/Linux_workshop/seq_tst.fa .
```

- ❑ Copy Swissprot BLAST database (we'll make a separate directory for it)

```
mkdir databases
cp /shared_data/Linux_workshop/databases/swissprot* ./databases
```

- ❑ Verify that the files have been copied (use **`ls`** command)

Reminder: local vs. network directories in BioHPC Cloud


/workdir
/workdir
/workdir
Local, *visible from only own workstation*
Directly attached – fast access
cbsum1c2b001
cbsum1c2b002
cbsum1c2b003
workstations
Network-attached – slow access
File server
/home
/programs
/shared_data
Network directories, *visible from all workstations*

Files frequently read and/or written (like input and output from an application being run) must be located on **local directories** (on BioHPC machines: **/workdir**)

Running applications example: BLAST

run the program

❑ **<u>Very</u> general syntax for launching applications:**

```
<path_to_application_executable>  [options] >& log
```

❑ **In our specific case** (command may be in a single line or split with "\"):

```
blastx \
-db ./databases/swissprot \
-num_alignments 1 \
-query seq_tst.fa \
-out hits.txt \
>& run.log
```

executable

path to databases files

alignments to report

query file

output file

redirect rest of STDOUT+STDERR to file on disk

❑ For full set of options, run

```
blastx -help | less
```

Running applications example: BLAST

running the program

```
blastx -db ./databases/swissprot \
-num_alignments 1 -query seq_tst.fa -out hits.txt >& run.log
```

- The program will run for about 1 minute and then write the main output to the file **`hits.txt`**, and he remaining output (STDERR stream) to **`run.log`**
  - Often output will appear in **`hits.txt`** gradually as a program is running

- For larger queries, the run will take (much) longer and produce more output…
  - 10,000 similar query sequences run using a similar command would take about 24 hours

Running a program, cnt.

- Running a program <u>in the background</u>

  - 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 prompt will return 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 (more about it later)
    - If all screen output redirected to disk, you may **log out** and leave the program running (to make sure, use **`nohup`** before the command)

```
blastx [options] >& run.log &
```

```
nohup blastx [options] >& run.log &
```

Run in the <u>background</u>

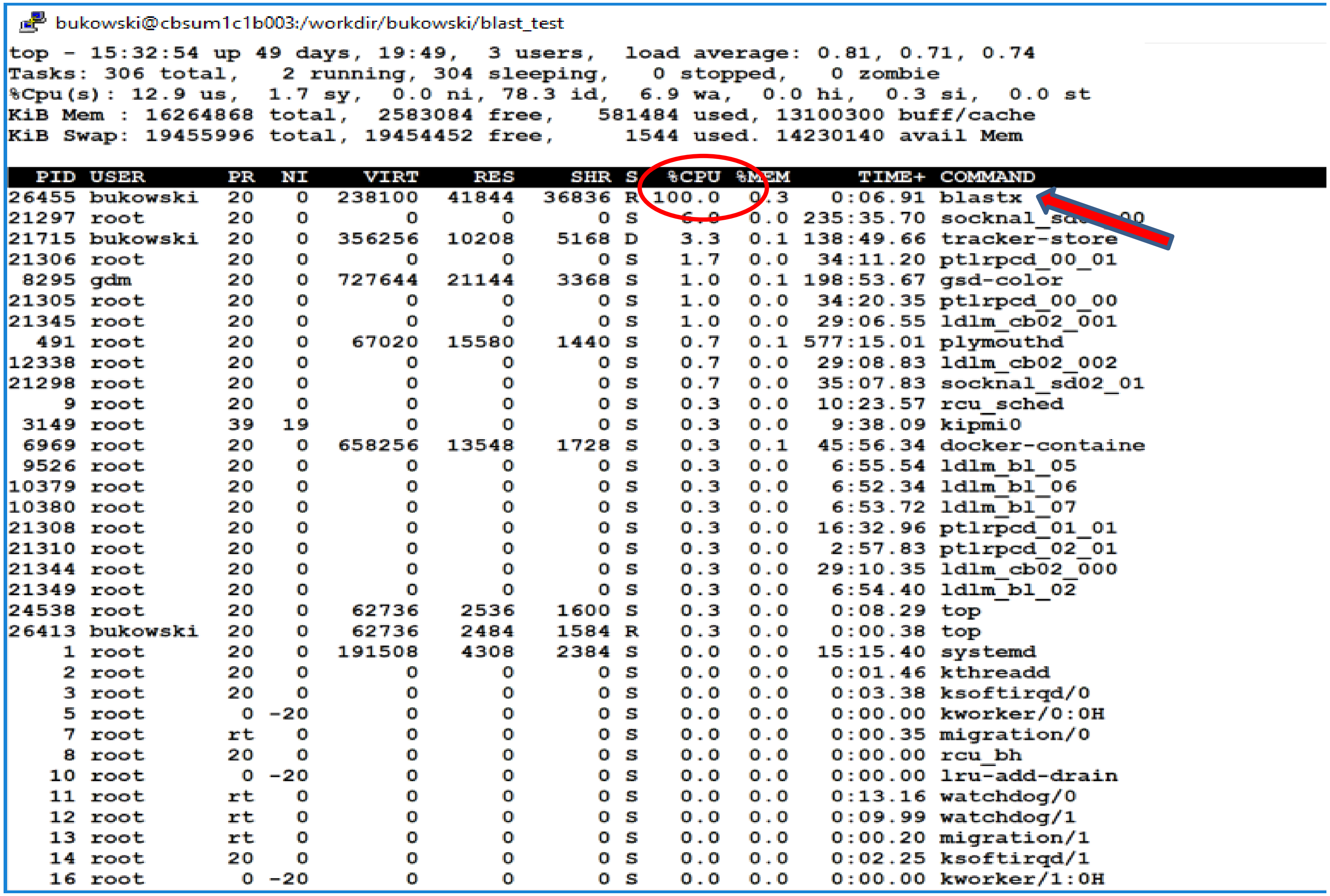
Keep running after logout

Insert options, as previously

Running applications

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

To exit – just type **q**



bukowski@cbsum1c1b003:/workdir/bukowski/blast_test

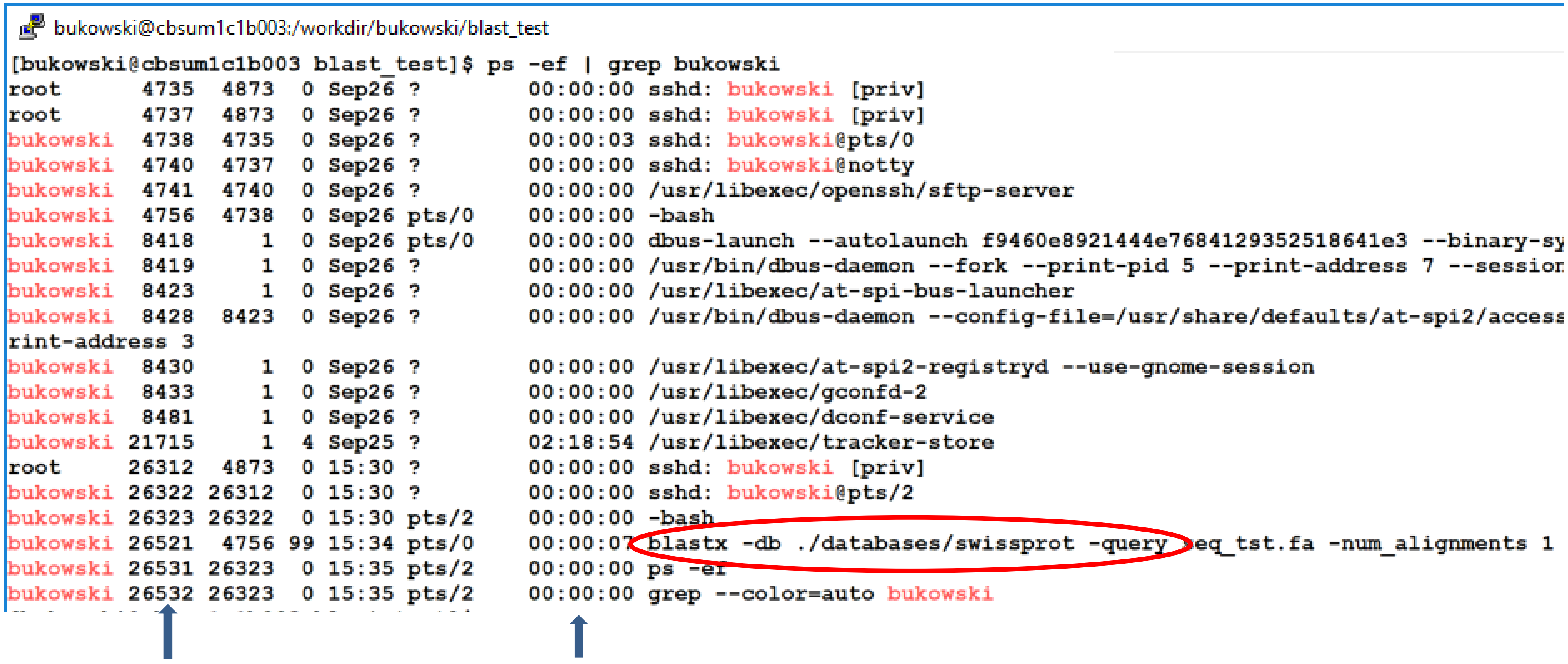
```
top - 15:32:54 up 49 days, 19:49,  3 users,  load average: 0.81, 0.71, 0.74
Tasks: 306 total,   2 running, 304 sleeping,   0 stopped,   0 zombie
%Cpu(s): 12.9 us,  1.7 sy,  0.0 ni, 78.3 id,  6.9 wa,  0.0 hi,  0.3 si,  0.0 st
KiB Mem : 16264868 total,  2583084 free,   581484 used, 13100300 buff/cache
KiB Swap: 19455996 total, 19454452 free,     1544 used. 14230140 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
26455 bukowski  20   0  238100  41844  36836 R 100.0  0.3   0:06.91 blastx
21297 root      20   0       0      0      0 S   6.0  0.0 235:35.70 socknal_sd02_00
21715 bukowski  20   0  356256  10208   5168 D   3.3  0.1 138:49.66 tracker-store
21306 root      20   0       0      0      0 S   1.7  0.0  34:11.20 ptlrpcd_00_01
 8295 gdm       20   0  727644  21144   3368 S   1.0  0.1 198:53.67 gsd-color
21305 root      20   0       0      0      0 S   1.0  0.0  34:20.35 ptlrpcd_00_00
21345 root      20   0       0      0      0 S   1.0  0.0  29:06.55 ldlm_cb02_001
  491 root      20   0   67020  15580   1440 S   0.7  0.1 577:15.01 plymouthd
12338 root      20   0       0      0      0 S   0.7  0.0  29:08.83 ldlm_cb02_002
21298 root      20   0       0      0      0 S   0.7  0.0  35:07.83 socknal_sd02_01
    9 root      20   0       0      0      0 S   0.3  0.0  10:23.57 rcu_sched
 3149 root      39  19       0      0      0 S   0.3  0.0   9:38.09 kipmi0
 6969 root      20   0  658256  13548   1728 S   0.3  0.1  45:56.34 docker-containe
 9526 root      20   0       0      0      0 S   0.3  0.0   6:55.54 ldlm_bl_05
10379 root      20   0       0      0      0 S   0.3  0.0   6:52.34 ldlm_bl_06
10380 root      20   0       0      0      0 S   0.3  0.0   6:53.72 ldlm_bl_07
21308 root      20   0       0      0      0 S   0.3  0.0  16:32.96 ptlrpcd_01_01
21310 root      20   0       0      0      0 S   0.3  0.0   2:57.83 ptlrpcd_02_01
21344 root      20   0       0      0      0 S   0.3  0.0  29:10.35 ldlm_cb02_000
21349 root      20   0       0      0      0 S   0.3  0.0   6:54.40 ldlm_bl_02
24538 root      20   0   62736   2536   1600 S   0.3  0.0   0:08.29 top
26413 bukowski  20   0   62736   2484   1584 R   0.3  0.0   0:00.38 top
    1 root      20   0  191508   4308   2384 S   0.0  0.0  15:15.40 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:01.46 kthreadd
    3 root      20   0       0      0      0 S   0.0  0.0   0:03.38 ksoftirqd/0
    5 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/0:0H
    7 root      rt   0       0      0      0 S   0.0  0.0   0:00.35 migration/0
    8 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcu_bh
   10 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 lru-add-drain
   11 root      rt   0       0      0      0 S   0.0  0.0   0:13.16 watchdog/0
   12 root      rt   0       0      0      0 S   0.0  0.0   0:09.99 watchdog/1
   13 root      rt   0       0      0      0 S   0.0  0.0   0:00.20 migration/1
   14 root      20   0       0      0      0 S   0.0  0.0   0:02.25 ksoftirqd/1
   16 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/1:0H
```

Running applications, cnt.

Checking on your application:

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

```
ps -ef | grep bukowski
```



bukowski@cbsum1c1b003:/workdir/bukowski/blast_test

```
[bukowski@cbsum1c1b003 blast_test]$ ps -ef | grep bukowski
root      4735  4873  0 Sep26 ?        00:00:00 sshd: bukowski [priv]
root      4737  4873  0 Sep26 ?        00:00:00 sshd: bukowski [priv]
bukowski  4738  4735  0 Sep26 ?        00:00:03 sshd: bukowski@pts/0
bukowski  4740  4737  0 Sep26 ?        00:00:00 sshd: bukowski@notty
bukowski  4741  4740  0 Sep26 ?        00:00:00 /usr/libexec/openssh/sftp-server
bukowski  4756  4738  0 Sep26 pts/0    00:00:00 -bash
bukowski  8418     1  0 Sep26 pts/0    00:00:00 dbus-launch --autolaunch f9460e8921444e76841293525l8641e3 --binary-sy
bukowski  8419     1  0 Sep26 ?        00:00:00 /usr/bin/dbus-daemon --fork --print-pid 5 --print-address 7 --sessio
bukowski  8423     1  0 Sep26 ?        00:00:00 /usr/libexec/at-spi-bus-launcher
bukowski  8428  8423  0 Sep26 ?        00:00:00 /usr/bin/dbus-daemon --config-file=/usr/share/defaults/at-spi2/access
rint-address 3
bukowski  8430     1  0 Sep26 ?        00:00:00 /usr/libexec/at-spi2-registryd --use-gnome-session
bukowski  8433     1  0 Sep26 ?        00:00:00 /usr/libexec/gconfd-2
bukowski  8481     1  0 Sep26 ?        00:00:00 /usr/libexec/dconf-service
bukowski 21715     1  4 Sep25 ?        02:18:54 /usr/libexec/tracker-store
root     26312  4873  0 15:30 ?        00:00:00 sshd: bukowski [priv]
bukowski 26322 26312  0 15:30 ?        00:00:00 sshd: bukowski@pts/2
bukowski 26323 26322  0 15:30 pts/2    00:00:00 -bash
bukowski 26521  4756 99 15:34 pts/0    00:00:07 blastx -db ./databases/swissprot -query seq_tst.fa -num_alignments 1
bukowski 26531 26323  0 15:35 pts/2    00:00:00 ps -ef
bukowski 26532 26323  0 15:35 pts/2    00:00:00 grep --color=auto bukowski
```

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

Running applications

- ❑ Stopping applications

- 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 **blastall** process form the previous slide, we would use

```
kill -9 18817
```

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

Keeping a program running in the background after you log out or disconnect

Option 1: Use **nohup** (as on previous slide). Of course, you can use this also with options 2 and 3.

Option 2: Start a program in a terminal within a **VNC session**

- the session keeps running after VNC connection is killed
- you can reconnect to VNC session later

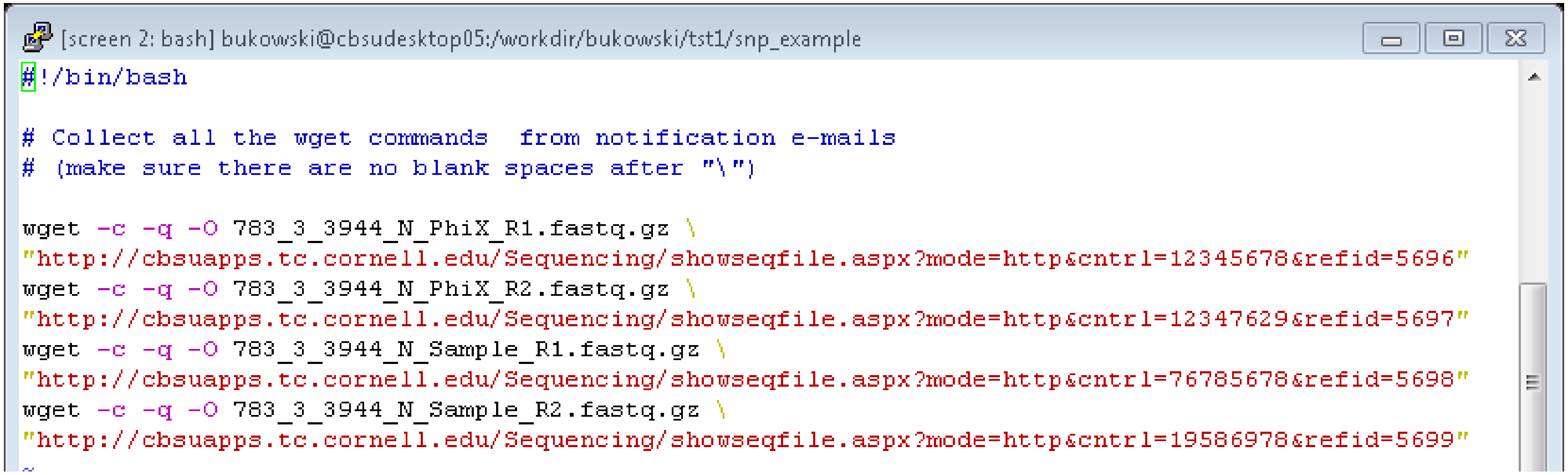
Option 3: Start a program within a **screen** window

- all such windows keep running after you disconnect using “Ctrl-a d” or by killing terminal window
- you can reconnect to screen session later

Shell scripting

Example we already talked about: Downloading Illumina sequencing results

Script **`download.sh`** is sent as attachment to notification e-mail from the sequencing facility



[screen 2: bash] bukowski@cbsudesktop05:/workdir/bukowski/tst1/snp_example

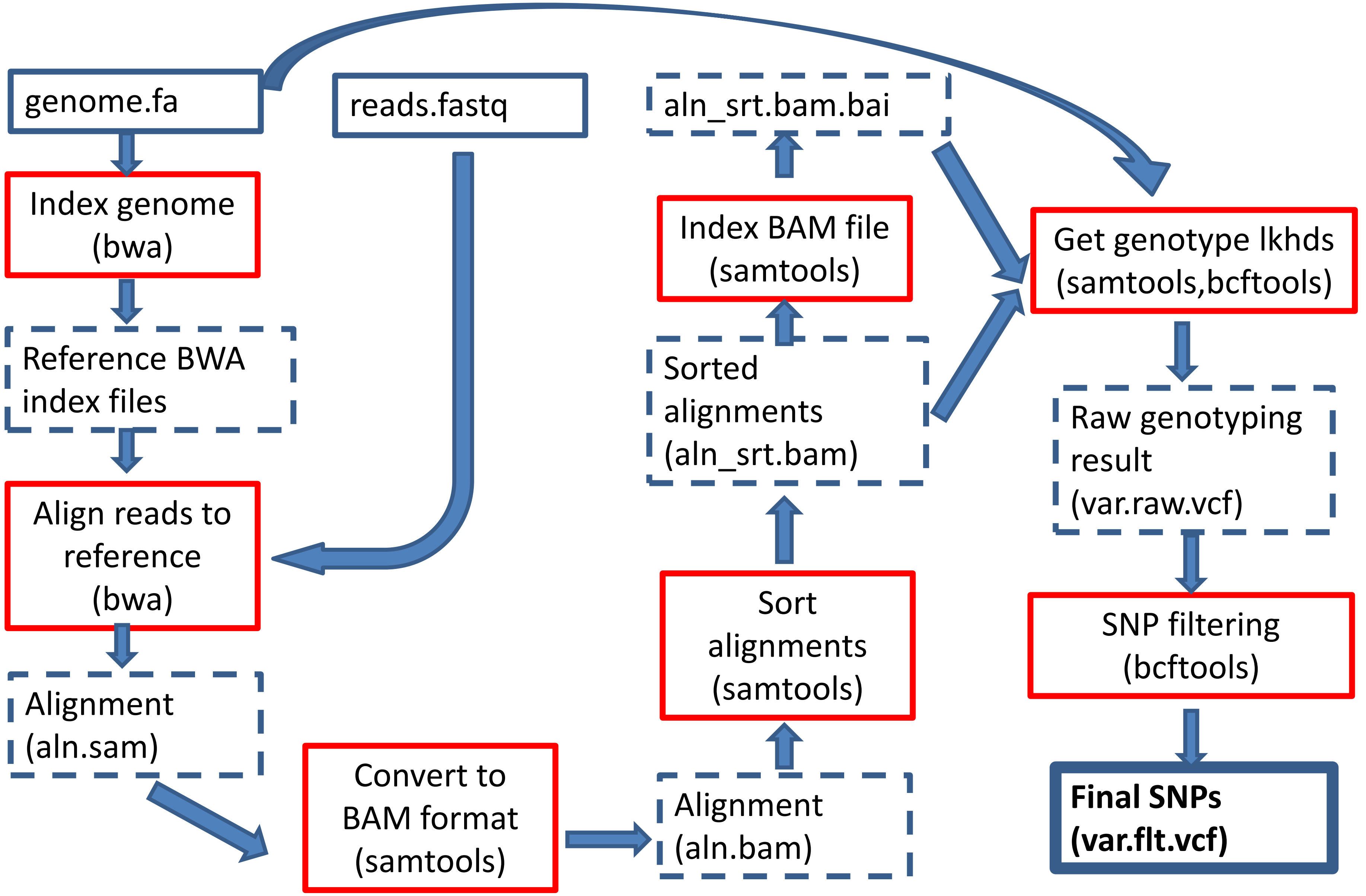
```
#!/bin/bash

# Collect all the wget commands  from notification e-mails
# (make sure there are no blank spaces after "\")

wget -c -q -O 783_3_3944_N_PhiX_R1.fastq.gz \
"http://cbsuapps.tc.cornell.edu/Sequencing/showseqfile.aspx?mode=http&cntrl=12345678&refid=5696"
wget -c -q -O 783_3_3944_N_PhiX_R2.fastq.gz \
"http://cbsuapps.tc.cornell.edu/Sequencing/showseqfile.aspx?mode=http&cntrl=12347629&refid=5697"
wget -c -q -O 783_3_3944_N_Sample_R1.fastq.gz \
"http://cbsuapps.tc.cornell.edu/Sequencing/showseqfile.aspx?mode=http&cntrl=76785678&refid=5698"
wget -c -q -O 783_3_3944_N_Sample_R2.fastq.gz \
"http://cbsuapps.tc.cornell.edu/Sequencing/showseqfile.aspx?mode=http&cntrl=19586978&refid=5699"
```

Copy **`download.sh`** to your Linux machine and run as a script

```
sh ./download.sh
```

Script for a complex task: SNP-calling

Example: given Illumina reads (in FASTQ format) and reference genome (FASTA), **call SNPs**


genome.fa
reads.fastq
aln_srt.bam.bai
Index genome (bwa)
Index BAM file (samtools)
Get genotype lkhds (samtools,bcftools)
Reference BWA index files
Sorted alignments (aln_srt.bam)
Raw genotyping result (var.raw.vcf)
Align reads to reference (bwa)
Sort alignments (samtools)
SNP filtering (bcftools)
Alignment (aln.sam)
Convert to BAM format (samtools)
Alignment (aln.bam)
Final SNPs (var.flt.vcf)

Scripts: tools for executing complex tasks

- Sequence of steps on previous slide is an example of a **pipeline**

  - Each step corresponds to (typically) one instance of a program or command

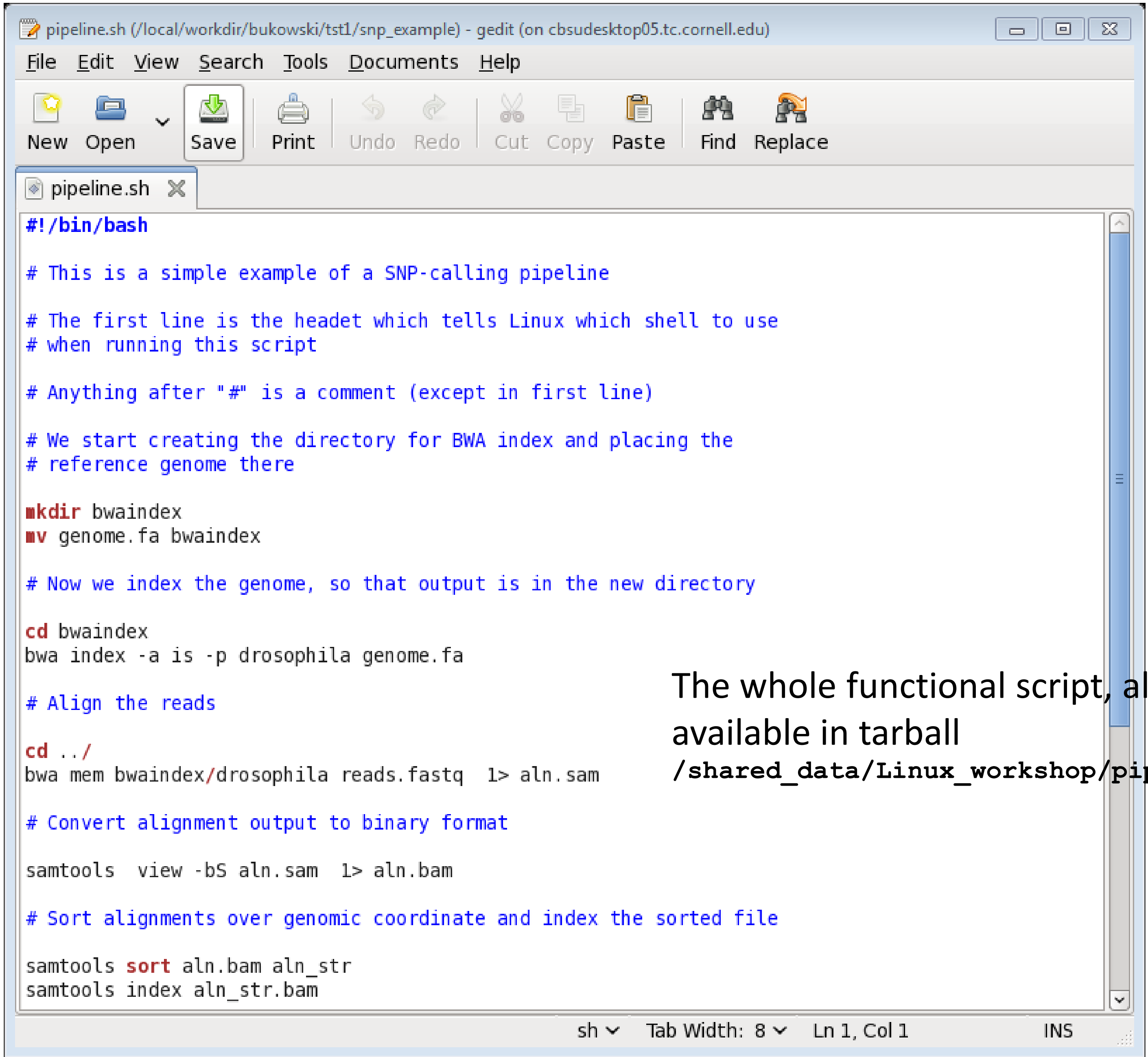
  - Input files used in a step are (typically) generated in preceding steps

  - Some steps may run quite long (depends on amount of input data and size of reference)

  - Executing each step in a terminal as a command is possible, but tedious and hard to repeat (for example, with a new input data)

  - Remedy: write a **shell script** – <u>a text file with commands</u>

Shell script: a set of commands (and comments) in a text file



pipeline.sh (/local/workdir/bukowski/tst1/snp_example) - gedit (on cbsudesktop05.tc.cornell.edu)
File Edit View Search Tools Documents Help
New Open Save Print Undo Redo Cut Copy Paste Find Replace
pipeline.sh
sh Tab Width: 8 Ln 1, Col 1 INS

```
#!/bin/bash

# This is a simple example of a SNP-calling pipeline

# The first line is the headet which tells Linux which shell to use
# when running this script

# Anything after "#" is a comment (except in first line)

# We start creating the directory for BWA index and placing the
# reference genome there

mkdir bwaindex
mv genome.fa bwaindex

# Now we index the genome, so that output is in the new directory

cd bwaindex
bwa index -a is -p drosophila genome.fa

# Align the reads

cd ../
bwa mem bwaindex/drosophila reads.fastq  1> aln.sam

# Convert alignment output to binary format

samtools  view -bS aln.sam  1> aln.bam

# Sort alignments over genomic coordinate and index the sorted file

samtools sort aln.bam aln_str
samtools index aln_str.bam
```

This is a fragment of an actual script implementing the SNP-calling pipeline.

The whole functional script, along with input files is available in tarball
`/shared_data/Linux_workshop/pipeline_example.tgz`

Shell scripts

- First line should be **`#!/bin/bash`** (indicates the shell used to interpret the script)
  - If absent, default shell will be used (bash)

- Everything in a line following "**#**" is a **comment**

- May include system commands (like **`cp, mv, mkdir`**, …) and commands launching programs (**`blastall, bwa, samtools`**, …)

- Commands will be executed "in the order of appearance"

- Long lines can be broken with "\" character
  - The "\" character must be the last one in a line (no blank spaces after it)

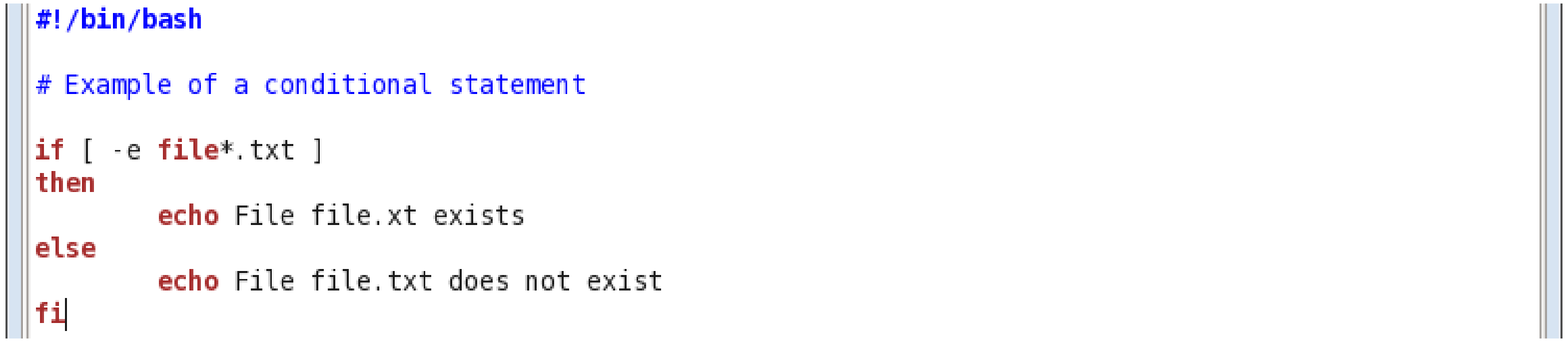
- Script (e.g., **`my_script.sh`**, in the current directory) can be run as in the following:

```
bash ./my_script.sh >& my_script.log &
./my_script.sh >& my_script.log  &
```

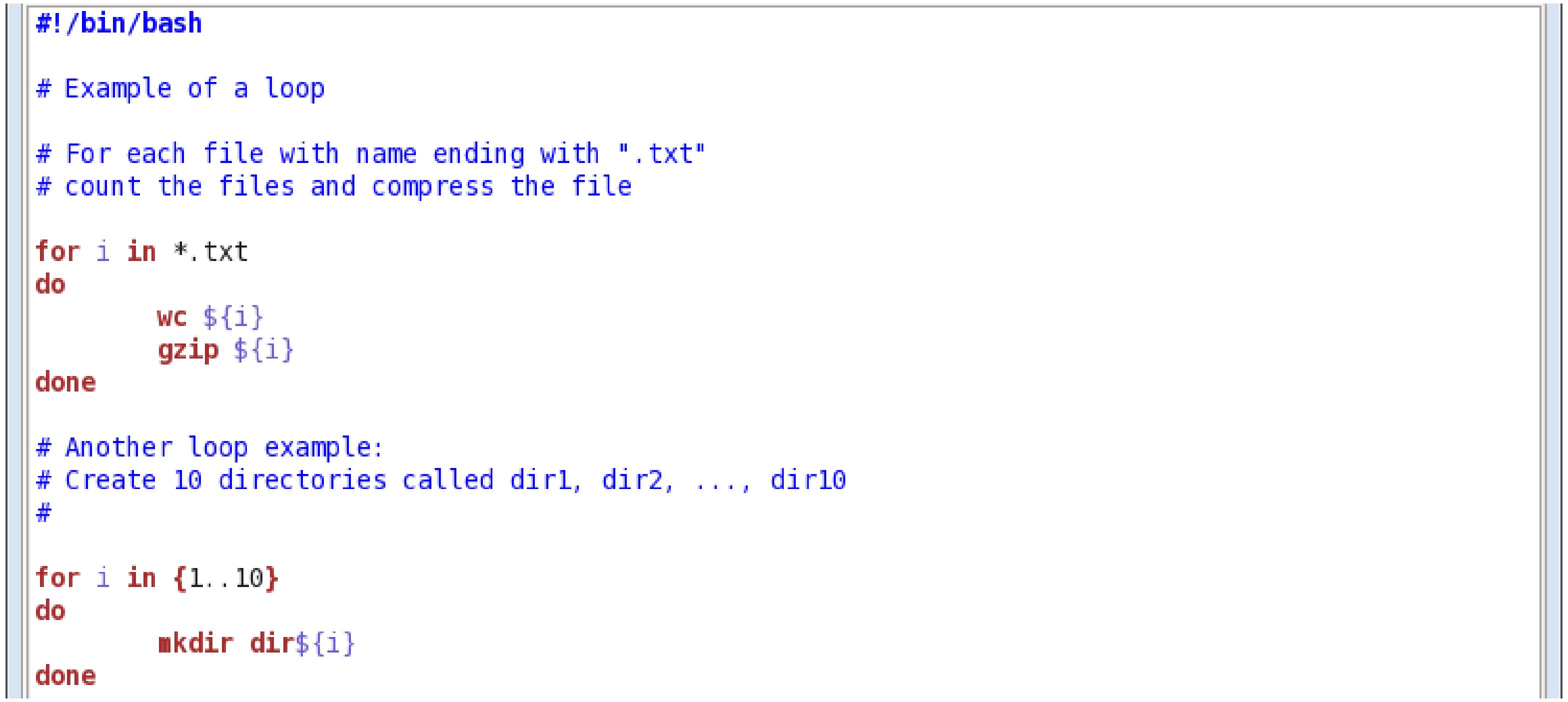
- The second command will work if the file **`my_script.sh`** is <u>made executable </u>with the command

```
chmod u+x my_script.sh
```

Shell scripts: conditionals and loops



```bash
#!/bin/bash

# Example of a conditional statement

if [ -e file*.txt ]
then
        echo File file.xt exists
else
        echo File file.txt does not exist
fi
```



```bash
#!/bin/bash

# Example of a loop

# For each file with name ending with ".txt"
# count the files and compress the file

for i in *.txt
do
        wc ${i}
        gzip ${i}
done

# Another loop example:
# Create 10 directories called dir1, dir2, ..., dir10
#

for i in {1..10}
do
        mkdir dir${i}
done
```

More about scripting

Multiple scripting tools available

- **shell** (bash, tcsh – good for stitching together shell commands)
- **perl** (very 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)

Using multiple processors

Recommended reading:

Efficient use of CPUs/cores on BioHPC Cloud machines

http://biohpc.cornell.edu/lab/doc/using_BioHPC_CPUs.pdf

Multiple processors

Using **BLAST** to search **swissprot** database for matches of 10,000 randomly chosen human cDNA sequences (swissprot is a good example of a small memory footprint).

machine	**CPU availa ble**	**cores available**	**cores used**	**time (hrs)**	**speedup (in machine)**
cbsulm10	4	64	**64**	0.931	27.506
cbsulm10	4	64	**16**	1.962	13.056
cbsulm10	4	64	**1**	25.619	1.000
cbsumm15	2	24	**24**	2.058	12.117
cbsumm15	2	24	**12**	2.593	9.616
cbsumm15	2	24	**1**	24.930	1.000
cbsum1c2b008	2	8	**8**	4.193	6.717
cbsum1c2b008	2	8	**1**	28.161	1.000

Using **BLAST** to search **nr** database for matches of 2,000 randomly chosen human cDNA sequences (nr is a good example of a large memory footprint).

machine	**CPU available**	**cores available**	**cores used**	**time (hrs)**	**speedup (in machine)**
cbsulm10	4	64	**64**	10.97	2.222
cbsulm10	4	64	**16**	24.37	1.000
cbsumm15	2	24	**24**	26.10	2.140
cbsumm15	2	24	**12**	55.85	1.000

Multiple processors

- It is VERY important to use multiple cores. BLAST on 64 cores takes only 0.931 hours (2K cDNA vs swissprot), the same run on a single core takes over 25 hours!

- **Speedup** is not directly proportional to the number of cores. Most often it is less than expected, but still sufficiently large to justify the effort. 64 cores compared to 1 core in swissprot example give 27.5 speedup rate, much less than 64-fold, but still large!

- Speedup depends on the machine (hardware), program (algorithm), and parameters (e.g., nr vs swissport). When using **nr** database on cbsumm15 the speedup between 12 and 24 cores is 2.14. For **swissprot** on the same machine it is only 1.26.

  - It is often a good idea to run a short example first (if possible) on a subset of data to figure out the optimal number of cores.

Multiple processors

Three ways to utilize multiple CPU cores on a machine:

- Using a given program’s built-in parallelization
- Simultaneously executing several programs in the background
- Using a “driver” program to execute multiple tasks in parallel

Multiple processors

- Take advantage of a program's built-in parallelism invoked with an option
  - read documentation to find out if your program has this feature
  - Look for keywords like "multithreading", "parallel execution", "multiple processors", etc.

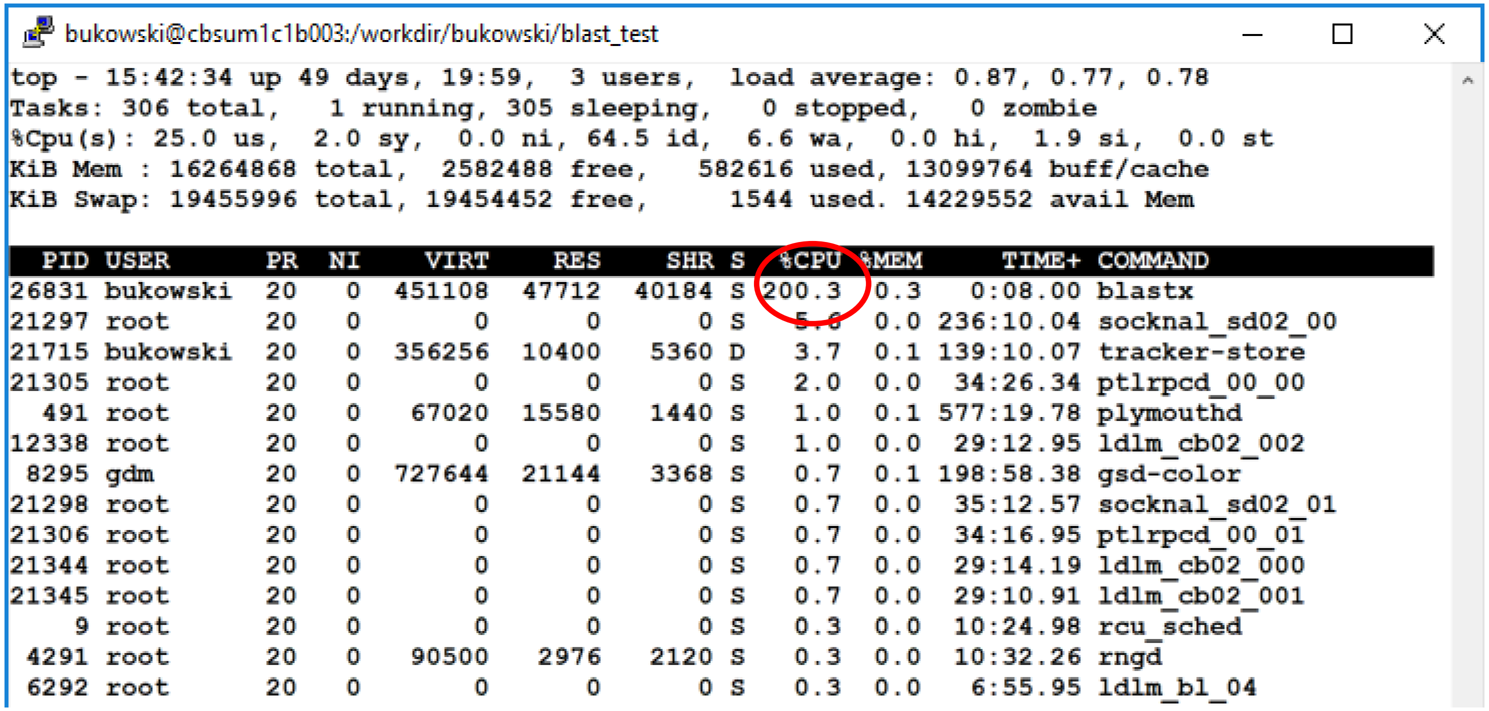
A few examples:

```
blastall -a 8 [other options]

blastx -num_threads 8 [other options]

tophat -p 8 [other options]

cuffdiff -p 8 [other options]

bwa -t 8 [other options]

bowtie -p 8 [other options]
```

Remember speedup is not perfect, so optimal number of threads needs to be optimized by trial and error using subset of input data

Multiple processors

```
blastx -num_threads 2 -db ./databases/swissprot -query seq_tst.fa
```



bukowski@cbsum1c1b003:/workdir/bukowski/blast_test

```
top - 15:42:34 up 49 days, 19:59,  3 users,  load average: 0.87, 0.77, 0.78
Tasks: 306 total,   1 running, 305 sleeping,   0 stopped,   0 zombie
%Cpu(s): 25.0 us,  2.0 sy,  0.0 ni, 64.5 id,  6.6 wa,  0.0 hi,  1.9 si,  0.0 st
KiB Mem : 16264868 total,  2582488 free,   582616 used, 13099764 buff/cache
KiB Swap: 19455996 total, 19454452 free,     1544 used. 14229552 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
26831 bukowski  20   0  451108  47712  40184 S 200.3  0.3   0:08.00 blastx
21297 root      20   0       0      0      0 S   5.6  0.0 236:10.04 socknal_sd02_00
21715 bukowski  20   0  356256  10400   5360 D   3.7  0.1 139:10.07 tracker-store
21305 root      20   0       0      0      0 S   2.0  0.0  34:26.34 ptlrpcd_00_00
  491 root      20   0   67020  15580   1440 S   1.0  0.1 577:19.78 plymouthd
12338 root      20   0       0      0      0 S   1.0  0.0  29:12.95 ldlm_cb02_002
 8295 gdm       20   0  727644  21144   3368 S   0.7  0.1 198:58.38 gsd-color
21298 root      20   0       0      0      0 S   0.7  0.0  35:12.57 socknal_sd02_01
21306 root      20   0       0      0      0 S   0.7  0.0  34:16.95 ptlrpcd_00_01
21344 root      20   0       0      0      0 S   0.7  0.0  29:14.19 ldlm_cb02_000
21345 root      20   0       0      0      0 S   0.7  0.0  29:10.91 ldlm_cb02_001
    9 root      20   0       0      0      0 S   0.3  0.0  10:24.98 rcu_sched
 4291 root      20   0   90500   2976   2120 S   0.3  0.0  10:32.26 rngd
 6292 root      20   0       0      0      0 S   0.3  0.0   6:55.95 ldlm_bl_04
```

- >100% CPU indicates the program is **multithreaded**
  - Multiple threads within a single process rather than multiple processes

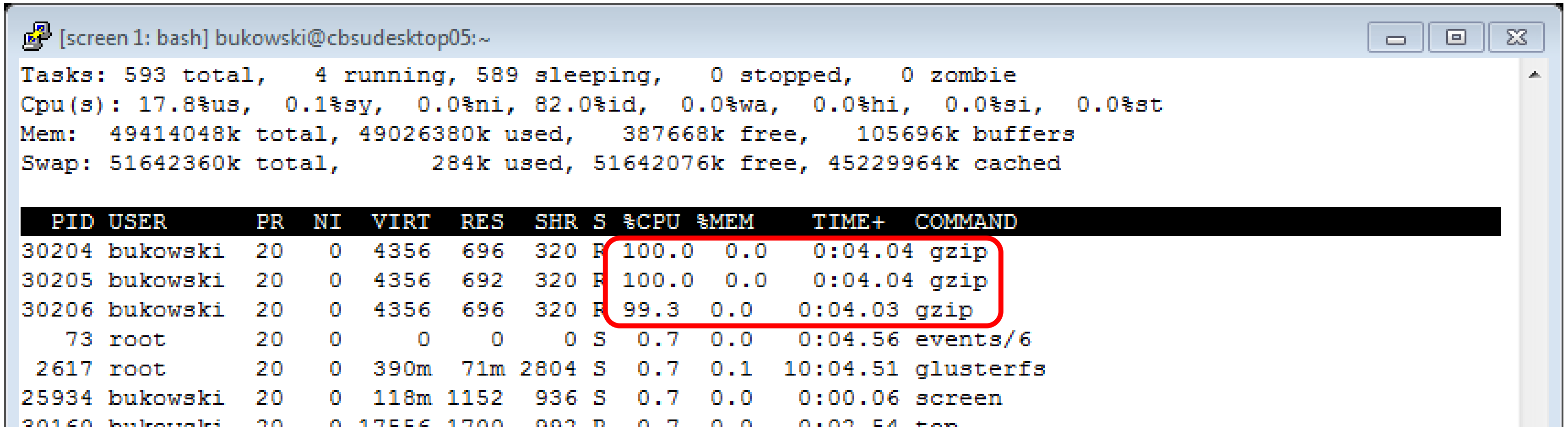
Multiple processors

- ❑ Simultaneously executing several programs in the background

Example: suppose we have to compress (gzip) several files. We can simply launch multiple **gzip** commands in the background, without waiting for previous ones to finish:

```
gzip file1 &
gzip file2 &
gzip file3 &
```

Multiple processes
(1 thread in each)



```
Tasks: 593 total,   4 running, 589 sleeping,   0 stopped,   0 zombie
Cpu(s): 17.8%us,  0.1%sy,  0.0%ni, 82.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  49414048k total, 49026380k used,   387668k free,   105696k buffers
Swap: 51642360k total,      284k used, 51642076k free, 45229964k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
30204 bukowski  20   0  4356  696  320 R 100.0  0.0   0:04.04 gzip
30205 bukowski  20   0  4356  692  320 R 100.0  0.0   0:04.04 gzip
30206 bukowski  20   0  4356  696  320 R 99.3  0.0   0:04.03 gzip
   73 root      20   0     0    0    0 S  0.7  0.0   0:04.56 events/6
 2617 root      20   0  390m  71m 2804 S  0.7  0.1  10:04.51 glusterfs
25934 bukowski  20   0  118m 1152  936 S  0.7  0.0   0:00.06 screen
```

Multiple processors

What if in the previous example, we had, say, **3000** files instead of just 3, but **still only a few processors**?

Submitting all 3000 commands simultaneously in the background (in principle, it could be done painlessly using a script) would not work too well, because:

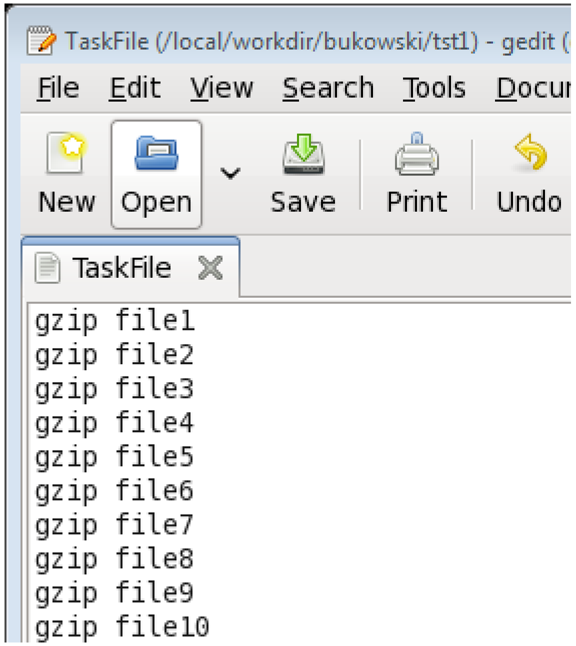
- ❑ Each processor would have to switch between many processes – possible, but inefficient
- ❑ With large number (and/or size) of files being processed, access to disk would become a bottleneck (i.e., processes would spend most of their time competing for access to disk)
    - ❑ Disk access (referred to as I/O – input/output) is always an issue for programs which do a lot of reading/writing (like **`gzip`**)
- ❑ As a result, we would get no speedup, or (more likely) **processing of all files in parallel would take longer than processing them one by one**

In situations like this (many short tasks and a few processors), we need a special "driver" tool to efficiently distribute the tasks.

Multiple processors

❑ Using a “driver” program to execute multiple tasks in parallel

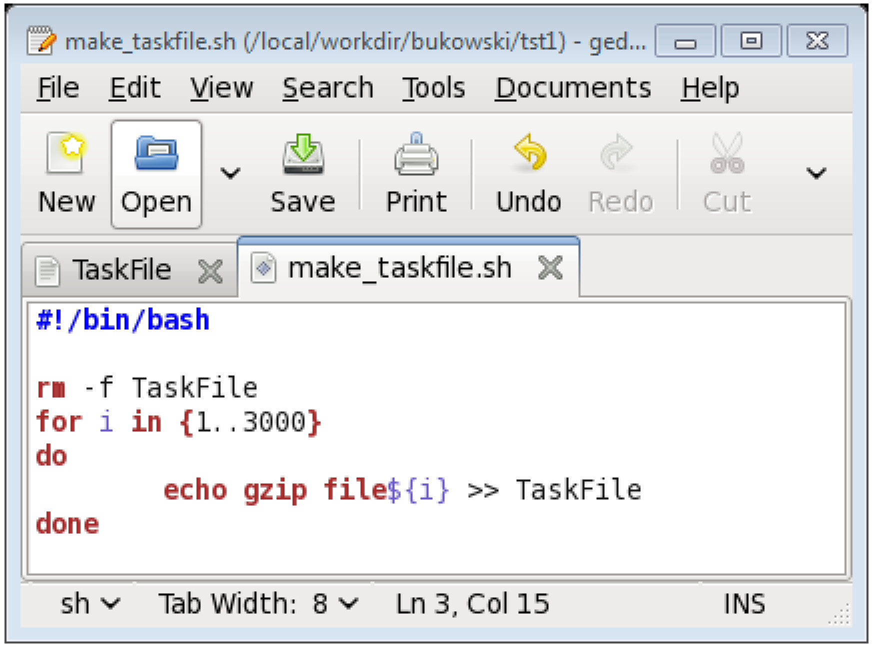
Example: create a file called (for example) **`TaskFile`**
(This is **NOT** a script, although it could be executed as such…)


TaskFile (/local/workdir/bukowski/tst1) - gedit (
File Edit View Search Tools Docur
New Open Save Print Undo
TaskFile

```
gzip file1
gzip file2
gzip file3
gzip file4
gzip file5
gzip file6
gzip file7
gzip file8
gzip file9
gzip file10
```

….. (up to **`file3000`**)

This long file can be created, for example, using the following shell script:


make_taskfile.sh (/local/workdir/bukowski/tst1) - ged...
File Edit View Search Tools Documents Help
New Open Save Print Undo Redo Cut
TaskFile make_taskfile.sh
sh Tab Width: 8 Ln 3, Col 15 INS

```
#!/bin/bash

rm -f TaskFile
for i in {1..3000}
do
        echo gzip file${i} >> TaskFile
done
```

Multiple processors

Then run the command (assuming the **`TaskFile`** and all **`file*`** files are in the current dir)

```
/programs/bin/perlscripts/perl_fork_univ.pl TaskFile NP >& log &
```

where **`NP`** is the number of processors to use (e.g., 10)

- **`perl_fork_univ.pl`** is an CBSU in-house “driver” script (written in perl)
- It will execute tasks listed in **`TaskFile`** using up to **`NP`** processors
  - The first **`NP`** tasks will be launched simultaneously
  - The **`(NP+1)`** th task will be launched right after one of the initial ones completes and a “slot” becomes available
  - The **`(NP+2)`** nd task will be launched right after another slot becomes available
  - …… etc., until all tasks are distributed
- Only up to **`NP`** tasks are running at a time (less at the end)
- All **`NP`** processors always kept busy (except near the end of task list) – **Load Balancing**

Mixed parallelization: running several simultaneous multi-threaded tasks (each processing different data) on a large machine (here: 64-core)

```
tophat -p 7 -o B_L1-1 --transcriptome-index genome/transcriptome/ZmB73_5a_WGS \
   --no-novel-juncs genome/maize \
   fastq/2284_6063_7073_C3AR7ACXX_B_L1-1_ATCACG_R1.fastq.gz \
   fastq/2284_6063_7073_C3AR7ACXX_B_L1-1_ATCACG_R2.fastq.gz >& B_L1-1.log &
tophat -p 7 -o B_L1-2 --transcriptome-index genome/transcriptome/ZmB73_5a_WGS \
   --no-novel-juncs genome/maize \
   fastq/2284_6063_7076_C3AR7ACXX_B_L1-2_TGACCA_R1.fastq.gz  \
   fastq/2284_6063_7076_C3AR7ACXX_B_L1-2_TGACCA_R2.fastq.gz >& B_L1-2.log &
tophat -p 7 -o B_L1-3 --transcriptome-index genome/transcriptome/ZmB73_5a_WGS \
   --no-novel-juncs genome/maize \
   fastq/2284_6063_7079_C3AR7ACXX_B_L1-3_CAGATC_R1.fastq.gz  \
   fastq/2284_6063_7079_C3AR7ACXX_B_L1-3_CAGATC_R2.fastq.gz >& B_L1-3.log &
tophat -p 7 -o L_L1-1 --transcriptome-index genome/transcriptome/ZmB73_5a_WGS \
   --no-novel-juncs genome/maize \
   fastq/2284_6063_7074_C3AR7ACXX_L_L1-1_CGATGT_R1.fastq.gz \
   fastq/2284_6063_7074_C3AR7ACXX_L_L1-1_CGATGT_R2.fastq.gz >& L_L1-1.log &
tophat -p 7 -o L_L1-2 --transcriptome-index genome/transcriptome/ZmB73_5a_WGS \
   --no-novel-juncs genome/maize \
   fastq/2284_6063_7077_C3AR7ACXX_L_L1-2_ACAGTG_R1.fastq.gz  \
   fastq/2284_6063_7077_C3AR7ACXX_L_L1-2_ACAGTG_R2.fastq.gz >& L_L1-2.log &
tophat -p 7 -o L_L1-3 --transcriptome-index genome/transcriptome/ZmB73_5a_WGS \
   --no-novel-juncs genome/maize \
   fastq/2284_6063_7080_C3AR7ACXX_L_L1-3_ACTTGA_R1.fastq.gz \
   fastq/2284_6063_7080_C3AR7ACXX_L_L1-3_ACTTGA_R2.fastq.gz >& L_L1-3.log &
tophat -p 7 -o S_L1-1 --transcriptome-index genome/transcriptome/ZmB73_5a_WGS \
   --no-novel-juncs genome/maize \
   fastq/2284_6063_7075_C3AR7ACXX_S_L1-1_TTAGGC_R1.fastq.gz \
   fastq/2284_6063_7075_C3AR7ACXX_S_L1-1_TTAGGC_R2.fastq.gz >& S_L1-1.log &
tophat -p 7 -o S_L1-2 --transcriptome-index genome/transcriptome/ZmB73_5a_WGS \
   --no-novel-juncs genome/maize \
   fastq/2284_6063_7078_C3AR7ACXX_S_L1-2_GCCAAT_R1.fastq.gz \
   fastq/2284_6063_7078_C3AR7ACXX_S_L1-2_GCCAAT_R2.fastq.gz >& S_L1-2.log &
tophat -p 7 -o S_L1-3 --transcriptome-index genome/transcriptome/ZmB73_5a_WGS \
   --no-novel-juncs genome/maize \
   fastq/2284_6063_7081_C3AR7ACXX_S_L1-3_GATCAG_R1.fastq.gz \
   fastq/2284_6063_7081_C3AR7ACXX_S_L1-3_GATCAG_R2.fastq.gz >& S_L1-3.log &
```

Multiple processors

General guidelines

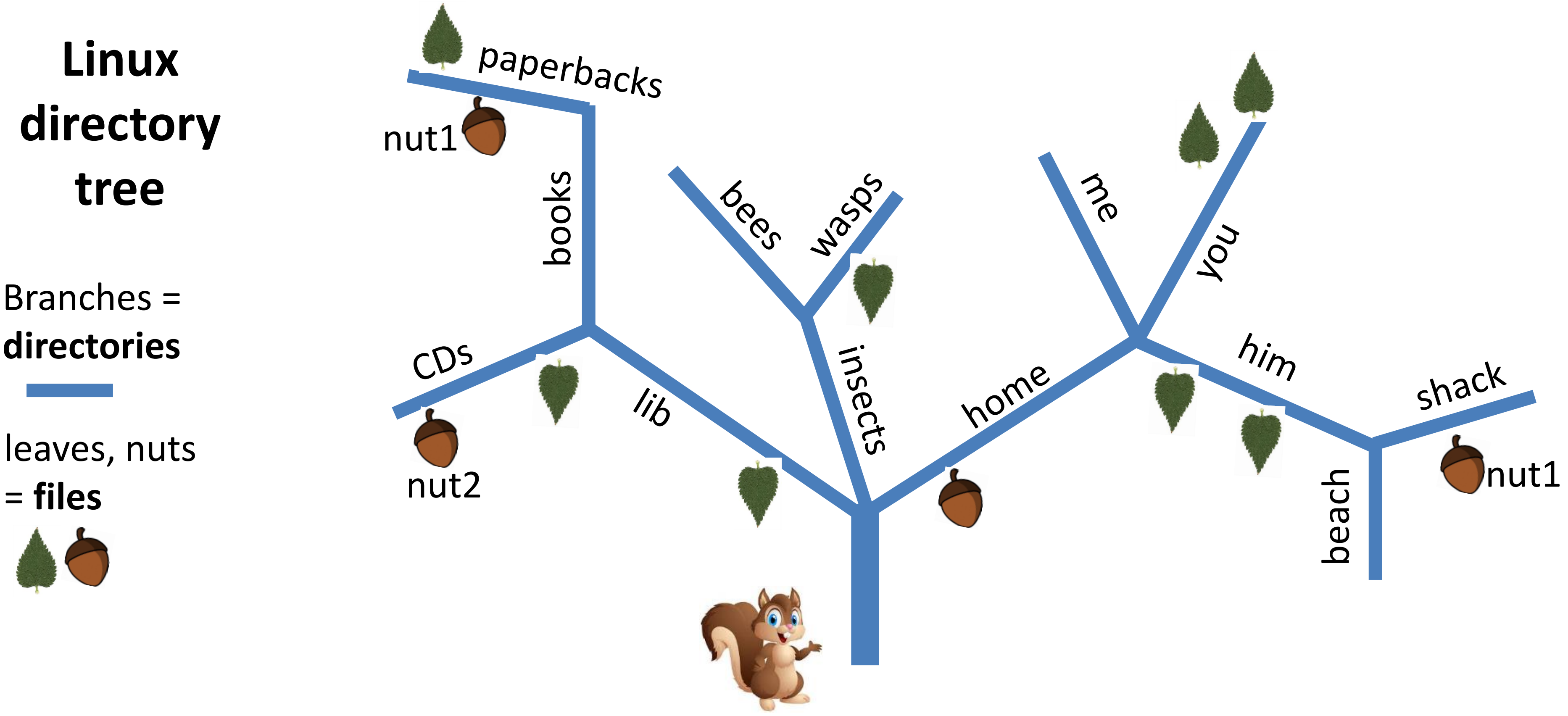
- ❑ Do not run more processes/threads than CPU cores available on the machine
  - For large number of tasks, use script **`perl_fork_univ.pl`**
- ❑ Run only as many simultaneous processes as will **fit in memory** (RAM)
  - when in doubt, run a single process first and check its memory requirement (for example, using **`top`**)
- ❑ Programs heavy on I/O will compete for disk access if run in parallel – running too many simultaneously is not a good idea
- ❑ If available, use program’s own multithreading options
- ❑ Using subset of input data, try to determine number of CPU cores which (for a given machine, input, and program options) gives the optimal speedup.

Old/Extras

Linux directory tree

Branches = **directories**

leaves, nuts = **files**



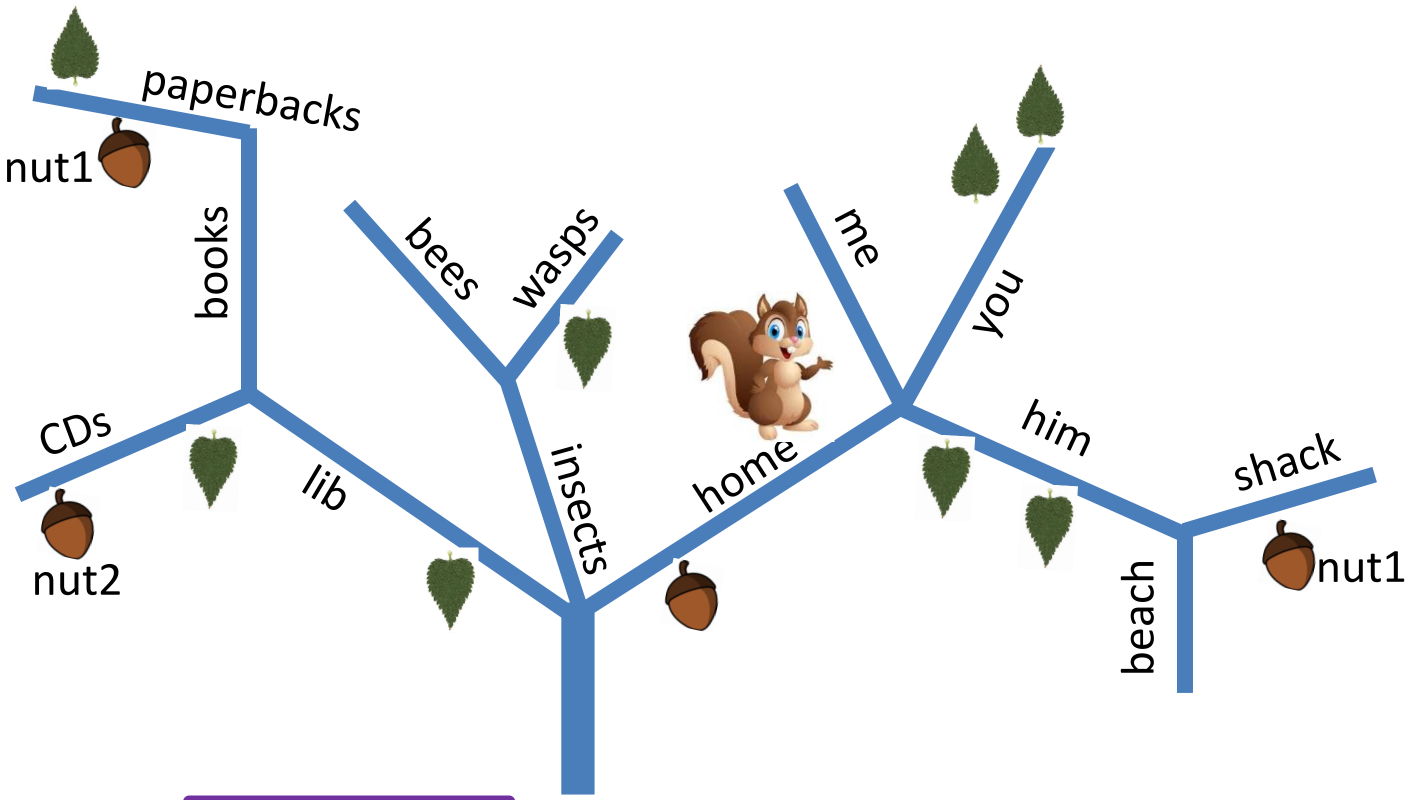
Direct squirrel to **nut1** (on the right) using commands:

`/`	get on the main trunk (referred to as **root**)
`some_name/`	from where you are, turn into branch "**`some_name`**"
`../`	return to the previous branch (closer to root)
`./`	stay where you are

Using these, direction from the ground to **`nut1`** will be:

```
/home/him/shack/nut1
```

This is called **absolute path** (starting from the trunk)

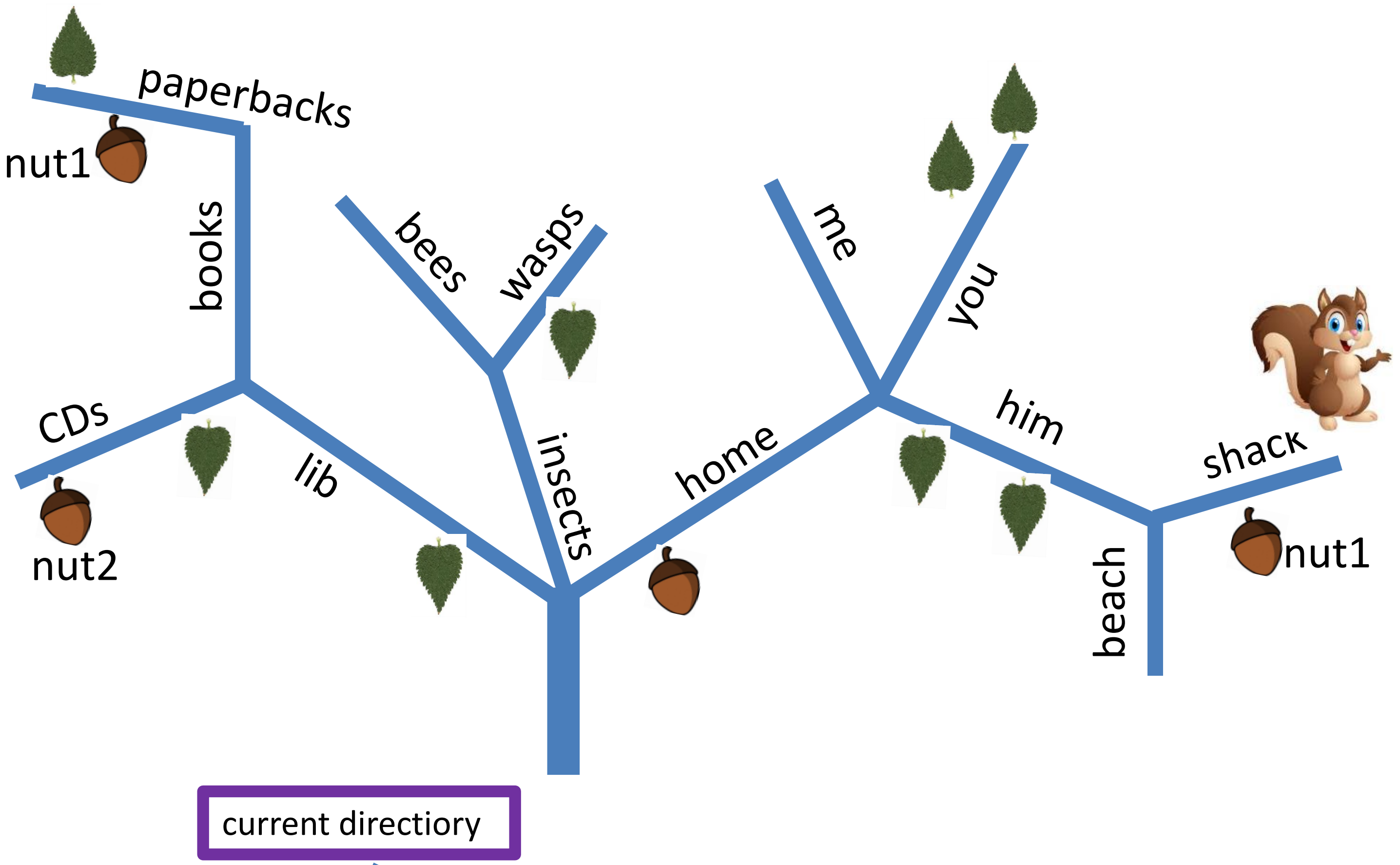


current directiory

Assume squirrel sitting on **`home`** rather than on the ground. We could make him jump to the ground and use the absolute path. Instead, we can simplify:

```
him/shack/nut1
```

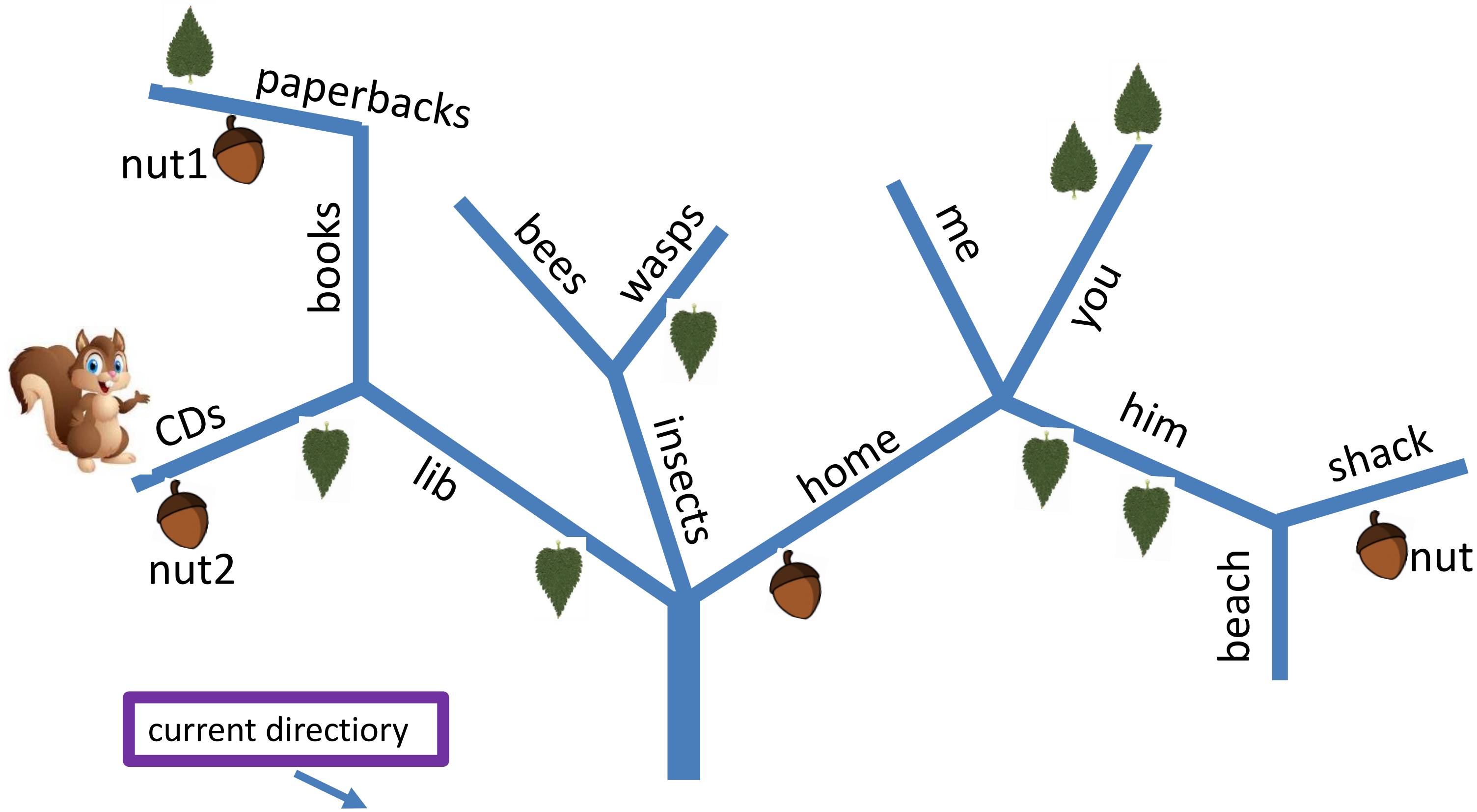
This is called **relative path** (starting from where “we are”)



Assume squirrel sitting on **`shack`**. We could make him jump to the ground and use the absolute path. Instead, we can simplify:

`nut1` or `./nut1`

This is called **relative path** (starting from where "we are")



Assume squirrel sitting on **CDs**. We could make him jump to the ground and use the absolute path. Instead, we can simplify:

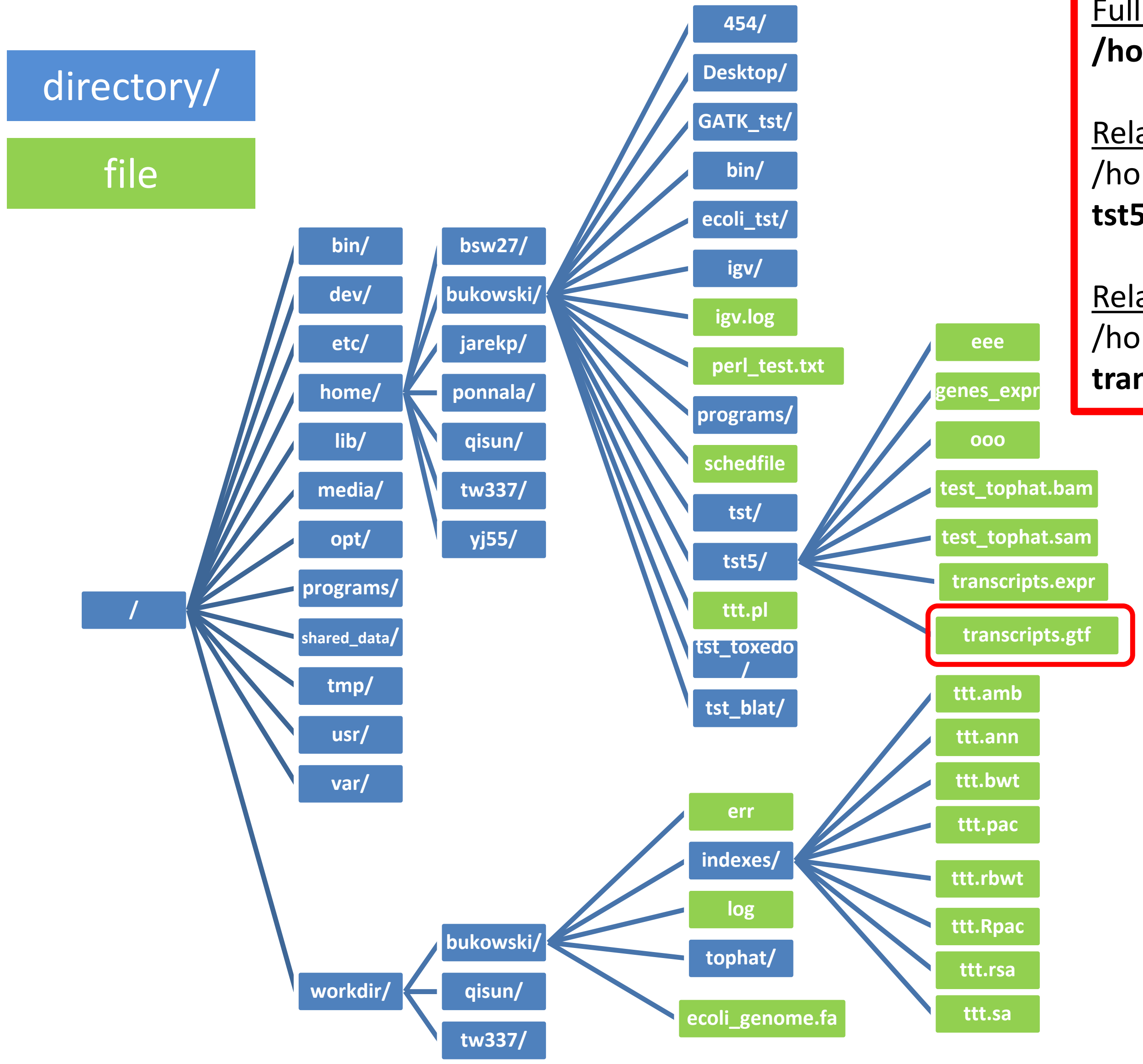
```
../../home/him/shack/nut1
```

Another example of **relative path**. Could also use, for example,

```
../../insects/bees/../wasps/../../home/me/../him/shack/nut1
```

Sounds unnecessarily long, but sometimes useful

Example of directory tree (more real)



Referring to files:

Full path:
/home/bukowski/tst5/transcripts.gtf

Relative path (i.e., relative to /home/bukowski)
tst5/transcripts.gtf

Relative path (i.e., relative to /home/bukowski/tst5)
transcripts.gtf

Cornell-Ithaca NetID simplifies work – get it if you can!

https://it.cornell.edu/cuweblogin-netids-policy/who-eligible-netid

Excerpt:

Weill Cornell Medical College faculty and staff can be issued a NetID if they need access to online services offered on the Ithaca campus. A NetID may be requested by contacting the IT Service Desk.

Students at Weill Cornell Medical College are not eligible for Cornell NetIDs.

Logging in via ssh from Windows PC

- Install remote access software (**PuTTy**). For details, consult http://biohpc.cornell.edu/lab/doc/Remote_access.pdf

- Use **PuTTy** to open a terminal window on the reserved workstation using **ssh** protocol
    - When connecting for the first time, a window will pop out about “caching server hostkey” – answer “Yes”. The window will not appear next time around
    - while you are typing your password, the terminal will appear frozen – this is on purpose!
    - Adjust colors, if desired (before or after connecting)
    - configure **X11 forwarding** (if you intend to run graphical software)
    - Save the configuration under an informative name

- You may open several terminal windows, if needed (in PuTTy – can use “Duplicate Session” function).

Working with text files

<u>Viewing text files</u>

```
less README.txt
```

(display the content of the file README.txt in the current directory dividing the file into pages; press SPACE bar to go to the next page or use up/down arrows)

```
head -100 my_reads.fastq
```

(display first 100 lines of the file my_reads.fastq in the current directory)

```
tail -100 my_reads.fastq
```

(display last 100 lines of the file my_reads.fastq in the current directory)

pipe
Output from first command is "piped" as input to the second

```
tail -1000 my_reads.fastq | less
```

(extract the last 1000 lines of the file my_reads.fastq and display them page by page)

```
head -1000 my_reads.fastq | tail -100
```

(display lines 901 through 1000 of the file my_reads.fastq). Note the **"|"** character: it <u>pipes</u> the output from one command as input to another

```
cat my reads.fastq
```

(print the file on screen)

```
cat my reads.fastq >> reads all
```

(append a file to the end of another)

```
wc  my_reads.fastq
```

(display the number of lines, words, and characters in a file)

Working with text files

<u>Looking for a string in a text file:</u>

```
grep “Error: lane” calc.log
```

(display all lines of the file **`calc.log`** *in the current directory which contain the string “Error: lane”)*

<u>Looking for a string in a group of text files:</u>

```
grep “Error: lane”  *.out
```

(display all files **`*.out`** *in the current directory which contain the string “Error: lane”; also display the lines containing that string)*

<u>Looking for lines which do **not** contain a string (ignore case)</u>

```
grep –i –v “some STring” my_file
```

Look for lines containing “AAA” surrounded by TABs

```
grep –P  “\tAAA\t”  my_file
```

`cut/paste` examples

File1

```
a b c
g h i
d e f
j k l
```

File2

```
 1  2  3
 7  8  9
 4  5  6
10 11 12
```

TAB-delimited files

```
cut -f 1,3 File1
```

```
a c
g i
d f
j l
```

```
cut -f 1 --complement File1
```

```
b c
h i
e f
k l
```

```
paste File1 File2
```

```
a b c  1  2  3
g h i  7  8  9
d e f  4  5  6
j k l 10 11 12
```

"–" means that the second file is to be read from **STDIN** (passed on through **pipe** "**|**")

```
cut -f 1,3 File1 | paste File2 -
```

```
 1  2  3 a c
 7  8  9 g i
 4  5  6 d f
10 11 12 j l
```

`sort` command

Let **`File`** contain a TAB- or space-delimited table

```
sort File
```

(sort **`File`** *alphabetically over whole rows)*

```
sort -k 2,2 -k 3,3n  -k 5,5nr  File  > new_File
```

(sort **`File`** *alphabetically over column 2, then numerically from small to large over column 3, and then numerically from large to small over column 5; write result to file* **`new_File`***)*

```
sort -u File
```

(sort **`File`** *keeping only unique rows)*

See `man sort` for lot's more information

Working with text files

Files transferred **to Linux machine from a Windows or Mac machine** often have line endings incompatible with Linux (depends on transfer software used and its settings)

To fix line endings, use **`dos2unix`** command

```
dos2unix my_file
```

```
mac2unix my_file
```

(the file **`my_file`** *will have linux line endings)*

```
dos2unix -n my_file my_file_converted
```

```
mac2unix -n my_file  my_file_converted
```

(the file **`my_file_converted`** *will have linux line endings, the original file* **`my_file`** *will be kept)*

VNC: starting VNC server

Please do NOT do it this way on BioHPC workstations! See next slide for server starting procedure on BioHPC Lab!

Log in to the machine via ssh client (e.g., PuTTy), then in the terminal window type:

```
vncserver
```

You will be asked to set up a password for your VNC session (it is separate from your password on the machine). Once this is done, the VNC server will start running. It will print out the port number (a small integer, typically 1, 2, …) to use while connecting from the client.

On BioHPC Lab machines, the VNC server is started through our website.

Running applications example: BLAST

- ❑ Input:
  - **FASTA file** with query sequences
    - We will use 9 random human cDNA sequences
  - **Database** of known sequences with which the query is to be compared
    - We will use **Swissprot** set of amino acid sequences
      - Need to translate each cDNA query in 6 frames and align to Swissprot templates
- ❑ Output
  - Text file describing hits
- ❑ Program to run: **`blastx`**

  Part of the **`blast+`** suite of programs