

# Practical Linux Examples

- Processing large text files
- Working with lots of files

Qi Sun

Bioinformatics Facility

Cornell University

# File types used in this lecture:

.fasta files are used for sequence records.

.FASTA

.FASTQ

.GFF3

>seq1

```
GAAACGCCGTAGCGCCGATGGTAGTGTGGGGTCTCCCCATGCGAGAGTAGGGAAGTCCAGGTATCAAAT  
TAAGCAGTAAGCCGGTGAATTCGGTGGAGCGGTAGTTCAGTCGGTTAG  
AATACCTGCCTGTCACCCCGTCCGTTCCGCCACTTACTAAGAAGCCTC  
GAGTTAACGCTCGAGGTTATTGCCAAAATCGCAAAAATCCTCTGCGTT  
TTACGCCATTTTTCCGACCTCAGTTAACGAAAATAGCATTAAAAGAGG  
CATATTATGGCTATCCCGTCTGAAAGACGACGTTGTTATTTTCATCTG  
TGAAAACGGCGCTTGAAGTTGATTATCGCGCAATTGATACCGCACAAATCTATGATAACGAAGCCGCAGT  
AGGTCAGGCGATTGCAGAAAGTGGCGTGCCACGTCATGAACTCCACATCACCCTAAAATCTGGATTGAA  
AATCTCAGCAAAGACAAATTGATCCCGAGTCTGAAAGAGAGCCTGCAAAAATTGCGTACCGATTATGTTG  
ATCTGACGCTAATCCACTGGCCGTCACCAAACGATGAAGTCTCTGTTGAAGAGTTTATGCAGGCGCTGCT  
GGAAGCCAAAAACAAGGGCTGACGCGTGAGATCGGTATTTCCAACCTTACGATCCCATTGATGGAAAAA  
GCGATTGCTGCTGTTGGCGCTGAAAACATCGCTACTAACCAGATTGAACTCTCTCCTTATCTGCAAAACC  
GTAAAGTGGTCGCTTGGGCTAAACAGCACAGTATCCATATACTTCTTATATGACGCTGGCGTATGGCAA  
GGCACTGAAAGATGAGGTTATTGCTCGTATTGCAGCTAA
```

The character ">"  
specify the title of the  
sequence record

>seq2

```
CCTGGTTAGCCCGGAAGGTCTGGCTCCTGAATGGGATTAAGCCTCTCTGACAGCTCCTCCGGGAGCTGTT  
TTTACATGCTCGCTAAGGAAATCGATAAAAGCCCGGATGCGCGTACTTACCGCACGGTCGCTGTAATAGA  
CTGCACTGAATGGCATTCTACTGGCAACACTTTATCTGCTATTAACCTCCACCAAT
```



# File types used in this lecture:

.FASTA

.FASTQ

.GFF3

.gff3 file is used for genome feature annotation

```
##gff-version 3
```

```
chr1 . gene          1300  9000  .  +  .  ID=g0001;Name=sonichedgehog
chr1 . mRNA          1300  9000  .  +  .  ID=mrna0001;Parent=g0001
chr1 . exon          1300  1500  .  +  .  ID=exon00001;Parent=mrna0001
chr1 . exon          1050  1500  .  +  .  ID=exon00002;Parent=mrna0001
chr2 . gene          3000  3902  .  -  .  ID=g59003;Name=abc
chr2 . mRNA          3000  3902  .  -  .  ID=mrna6704;Parent=g59003
chr2 . exon          3700  3902  .  -  .  ID=exon00005;Parent=mrna6704
```

chromosome

Feature  
type

start

end

strand

ID & annotation









```
cat DeD7RACXX.fastq | head -n 40000 | grep AGATCGGAAGAGC
```

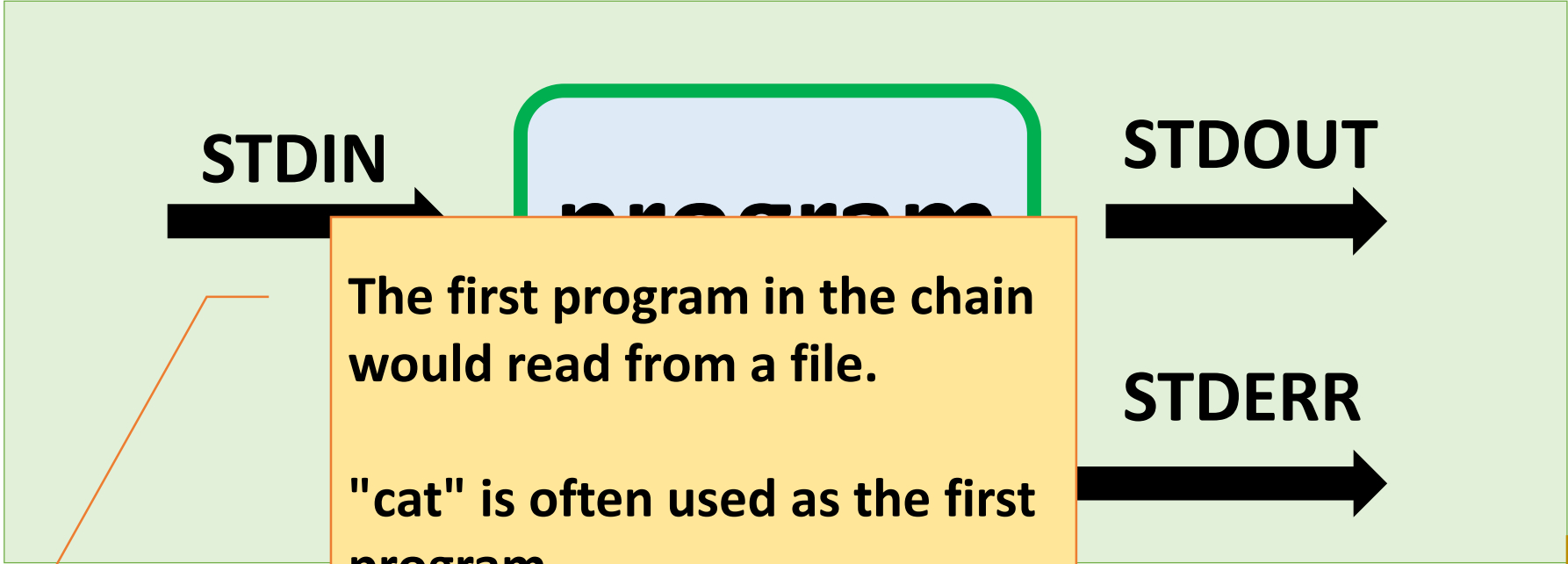
VS

```
cat DeD7RACXX.fastq | grep AGATCGGAAGAGC | head -n 40000
```





# Three streams for a standard Linux program



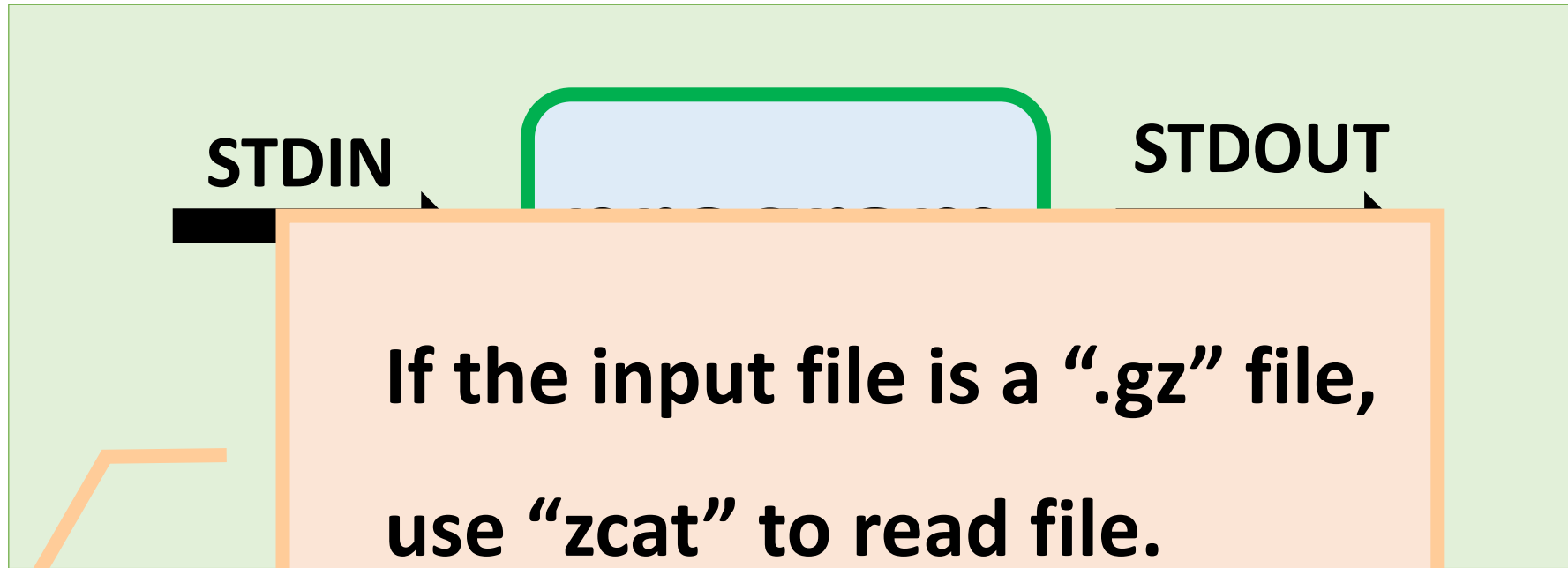
The first program in the chain would read from a file.  
"cat" is often used as the first program.

Write to file

One liner



# Three streams for a standard Linux program



One line



# grep

Search for a pattern and output  
matched lines

```
$ cat mydata.txt
```

```
AAGATCAAAAAAGA  
ATTACGAAAAAGA  
ACCTGTTGGATCAAAGTT  
AACTTTCGACGATCT  
ATTTTTTTAGAAAGG
```

```
$ cat mydata.txt | grep '[AC]GATC'
```

```
AAGATCAAAAAAGA  
AACTTTCGACGATCT
```

**These two commands are same**

```
cat mydata.txt | grep 'GATC'
```

**=**

```
grep 'GATC' mydata.txt
```

\* Functions like sort, wc, gunzip, head, tail, uniq can process either a file or data from STDIN.

# grep -v

Filter out certain lines

```
$ cat mydata.txt
```

```
#seq1  
AAGATCAAAAAAGA  
  
#seq2  
  
ACCTGTTGGATCCAAAGTT
```

```
$ grep -v '^#'
```

```
AAGATCAAAAAAGA  
  
ACCTGTTGGATCCAAAGTT
```

- “-v” to select non-matching lines
- “^” beginning of a line; “\$” end of line



# **wc -l** Count the number of lines

```
$ cat mydata.txt
```

```
AAGATCAAAAAAGA  
ATTACGAAAAAGA  
ACCTGTTGGATCAAAGTT  
AACTTTCGACGATCT  
ATTTTTTTAGAAAGG
```

```
$ cat mydata.txt | grep '[AC]GATC' | wc -l
```

```
2
```

# sort

Sort the text in a file

```
$ sort myChr.txt
```

```
Chr1  
Chr10  
Chr2  
Chr3  
Chr4  
Chr5
```

```
$ sort -V myChr.txt
```

```
Chr1  
Chr2  
Chr3  
Chr4  
Chr5  
Chr10
```

```
$ sort -n myPos.txt
```

```
1  
2  
3  
4  
5  
10
```

# sort

Sort the text by multiple columns

```
$ sort -k1,1V -k2,2n myChrPos.txt
```

```
Chr1      12  
Chr1     100  
Chr1     200  
Chr2      34  
Chr2     121  
Chr2     300
```

# Locale and sorting

## Computer English

LC\_ALL=C

Alphabetic order

A  
B  
C  
...  
X  
Y  
Z  
a  
b  
c  
...  
x  
y  
z

## US English \*

LC\_ALL=US\_en

Alphabetic order

a  
A  
b  
B  
c  
C  
...  
x  
X  
y  
Y  
z  
Z

Use this Linux command to find out the locale setting on your server:

**locale**

\* On Linux, US English locale sorting also ignore the non-alphanumeric characters. This could cause problems.

# Locale setting on BioHPC

Use the “locale” command to check locale setting:

```
[qisun@cbsum1c2b010 ~]$ locale
LANG=en_US.UTF-8
LC_CTYPE="C"
LC_NUMERIC="C"
LC_TIME="C"
LC_COLLATE="C"
LC_MONETARY="C"
LC_MESSAGES="C"
LC_PAPER="C"
LC_NAME="C"
LC_ADDRESS="C"
LC_TELEPHONE="C"
LC_MEASUREMENT="C"
LC_IDENTIFICATION="C"
LC_ALL=C
```

If you see errors like this when running software:

RuntimeError:

Python 3 was configured to **use ASCII as encoding** for the environment. Consult <https://click.palletsprojects.com/en/7.x/python3/> for mitigation steps.

You need to change the locale:

```
export LC_ALL=en_US.utf-8
export LANG=en_US.utf-8
```

Some extra parameter to set for the “sort” command

**LC\_ALL=C sort -S 4G -k1,1 myChr.txt**

**LC\_ALL=C** vs **LC\_ALL=US\_en**

|     |     |
|-----|-----|
| AA  | aa  |
| BB  | a.a |
| a.a | AA  |
| aa  | bb  |
| bb  | BB  |

\* BioHPC default locale is C. So you can skip this parameter

**-S 4G**

Use RAM as buffer 4G



# uniq -c

Count the occurrence of unique tags

```
$ cat mydata.txt
```

```
ItemB  
ItemA  
ItemB  
ItemC  
ItemB  
ItemC  
ItemB  
ItemC
```

```
$ cat mydata.txt | sort | uniq -c
```

```
1  ItemA  
4  ItemB  
3  ItemC
```

Mark sure to run  
"sort" before "uniq"

## Merging files:

**cat** f1 f2 VS **paste** f1 f2 VS **join** f1 f2

### File 1:

Item1

Item2

### File2:

Item3

Item4

```
cat File1 File2 > mergedfile1
```

Item1

Item2

Item3

Item4

```
paste File1 File2 > mergedfile2
```

Item1

Item3

Item2

Item4

\* Make sure that that two files has same number of rows and sorted the same way. Otherwise, use "join"

# join

## Merging two files that share a common field

### File 1:

|       |             |
|-------|-------------|
| Gene1 | DNA-binding |
| Gene2 | kinase      |
| Gene3 | membrane    |

### File2:

|       |     |
|-------|-----|
| Gene2 | 764 |
| Gene3 | 23  |
| Gene4 | 34  |

```
join -1 1 -2 1 File1 File2 > mergedfile
```

```
Gene2      Kinase      764  
Gene3      membrane    23
```

```
join -1 1 -2 1 -a1 File1 File2 > mergedfile
```

```
Gene1      DNA-binding  
Gene2      Kinase      764  
Gene3      membrane    23
```

# cut

## Output selected columns in a table

```
$ cat mydata.txt
```

```
Chr1 1000 2250 Gene1  
Chr1 3010 5340 Gene2  
Chr1 7500 8460 Gene3  
Chr2 8933 9500 Gene4
```

```
$ cat mydata.txt | cut -f 1,4
```

```
Chr1 Gene1  
Chr1 Gene2  
Chr1 Gene3  
Chr2 Gene4
```

# sed

## Modify text in a file

```
$ cat mydata.txt
```

```
Chr1 1000 2250 Gene1  
Chr1 3010 5340 Gene2  
Chr1 7500 8460 Gene3  
Chr2 8933 9500 Gene4
```

```
$ cat mydata.txt | sed "s/^Chr//"
```

```
1 1000 2250 Gene1  
1 3010 5340 Gene2  
1 7500 8460 Gene3  
2 8933 9500 Gene4
```

# awk

Probably the most versatile function  
in Linux

```
$ cat mydata.txt
```

```
Chr1 1000 2250 Gene1  
Chr1 3010 5340 Gene2  
Chr1 7500 8460 Gene3  
Chr2 8933 9500 Gene4
```

```
$ cat mydata.txt |\  
awk '{if ($1=="Chr1") print $4}'
```

```
Gene1  
Gene2  
Gene3
```



# awk vs cut

cut -f2,1 mydata.txt

Input data file: mydata.txt

|      |      |      |       |
|------|------|------|-------|
| Chr1 | 1000 | 2250 | Gene1 |
| Chr1 | 3010 | 5340 | Gene2 |
| Chr1 | 7500 | 8460 | Gene3 |
| Chr2 | 8933 | 9500 | Gene4 |

|      |      |
|------|------|
| Chr1 | 1000 |
| Chr1 | 3010 |
| Chr1 | 7500 |
| Chr2 | 8933 |

Same order as  
the input file

awk 'BEGIN{FS="\t"; OFS="\t"} {print \$2,\$1;}' \  
mydata.txt

|      |      |
|------|------|
| 1000 | Chr1 |
| 3010 | Chr1 |
| 7500 | Chr1 |
| 8933 | Chr2 |

Customized order

A Good Practice:

Create a shell script file for the one liner


```
cat D7RACXX.fastq | \  
head -n 40000 | \  
grep AGATCGGAAGAGC | \  
wc -l
```

Run the shell script

```
sh checkadapter.sh
```

# Debug a one-liner

```
zcat human.gff3.gz | \
```

```
 awk 'BEGIN {OFS = "\t"}; {if ($3=="gene") print $1,$4-1,$5}' | \
```

```
bedtools coverage -a win1mb.bed -b stdin -counts | \
```

```
LC_ALL=C sort -k1,1V -k2,2n > gene.cover.bed
```

```
zcat human.gff3.gz | head -n 1000 > tmpfile
```

```
cat tmpfile | \
```

```
awk 'BEGIN {OFS = "\t"}; {if ($3=="gene") print $1,$4-1,$5}' | head -n 100
```

# Many bioinformatics software support STDIN instead of input file

Run “BWA” without pipe:

```
bwa mem ref.fa reads.fq > tmp.sam  
samtools view -b tmp.sam > out.bam
```

Create a temporary SAM file

With pipe:

```
bwa mem ref.fa reads.fq | samtools view -bS - > out.bam
```

Use "-" to specify input from STDIN instead of a file

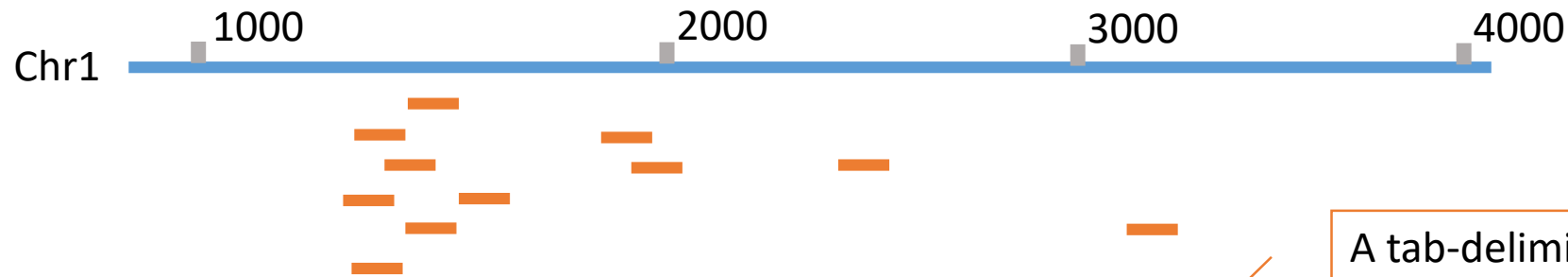
## Using pipe with bed tools:

The bedtools takes in two input files, you need to specify which file from stdin

```
..... | bedtools coverage -a FirstFile -b stdin
```

# Using BEDtools to process genomics data files

An example: Count the number of reads in each 1kb sliding window of the genome



A tab-delimited text file:

|      |          |
|------|----------|
| Chr1 | 21234023 |
| Chr2 | 19282343 |
| Chr3 | 15845499 |

```
bedtools makewindows -g genome.txt -w 1000 -s 1000 > win1000.bed
```

```
bedtools coverage -abam Sample.bam -b win1000.bed -counts> coverage.bed
```



**When working with text files from Windows (“/r/n”):**

```
dos2unix myfile.sh
```

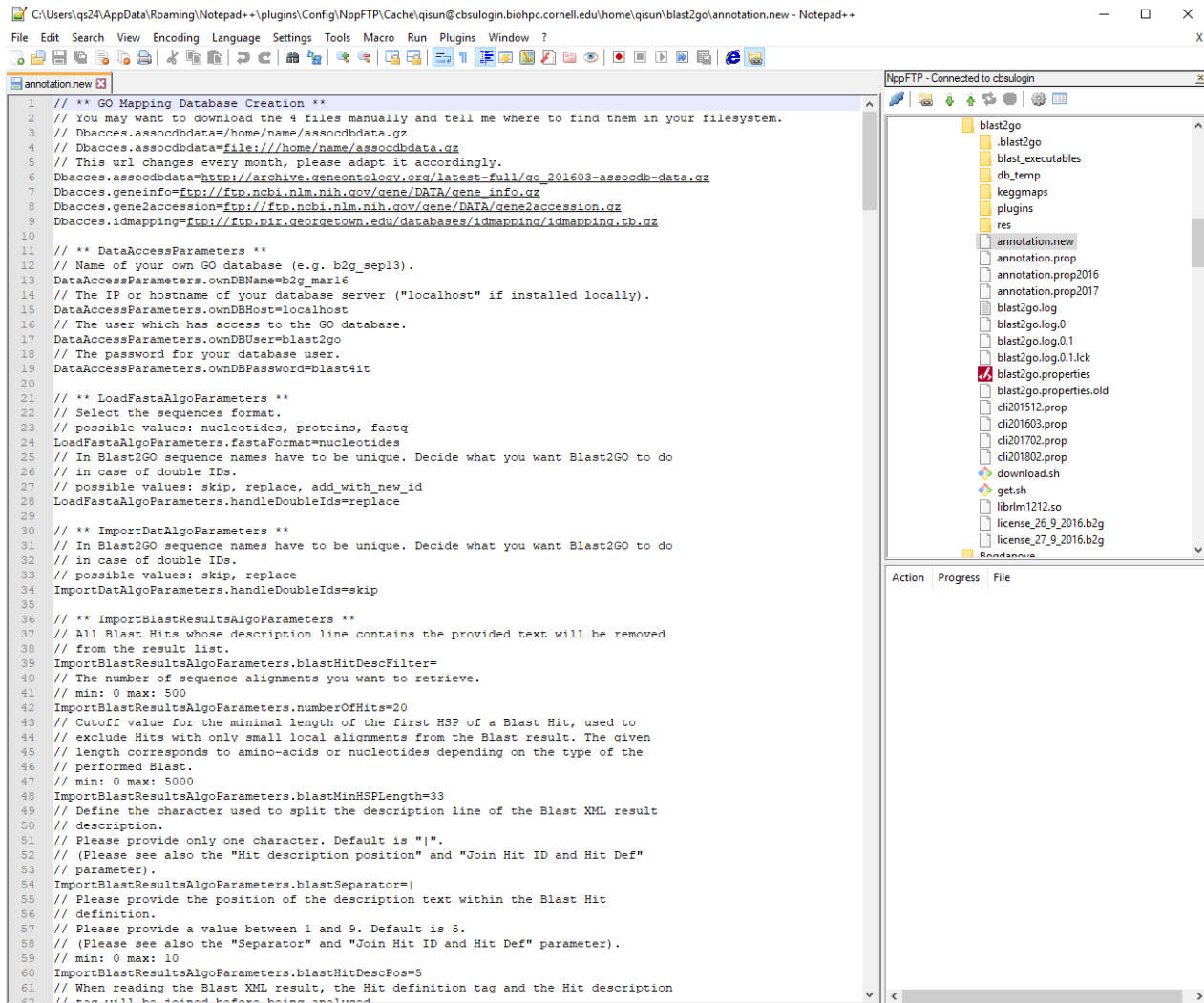
**When working with text files from Mac:**

- Mac files in general are same as Linux files;
- Only exception is text file saved from Excel (“/r”):

```
mac2unix myfile.txt
```

# Edit text file without using “vi”

Software to directly open/edit/save files on a remote Linux server



The screenshot shows a Notepad++ window titled 'annotation.new' with the following content:

```
1 // ** GO Mapping Database Creation **
2 // You may want to download the 4 files manually and tell me where to find them in your filesystem.
3 // Dbaces.assocdbdata=/home/name/assocdbdata.gz
4 // Dbaces.assocdbdata=file:///home/name/assocdbdata.gz
5 // This url changes every month, please adapt it accordingly.
6 Dbaces.assocdbdata=http://archive.geneontology.org/latest-full/go_201603-assocdb-data.gz
7 Dbaces.geneinfo=http://ftp.ncbi.nlm.nih.gov/gene/DATA/gene_info.gz
8 Dbaces.gene2accession=http://ftp.ncbi.nlm.nih.gov/gene/DATA/gene2accession.gz
9 Dbaces.idmapping=http://ftp.nir.georgetown.edu/databases/idmapping/idmapping.tb.gz
10
11 // ** DataAccessParameters **
12 // Name of your own GO database (e.g. b2g_sep13).
13 DataAccessParameters.ownDBName=b2g_mar16
14 // The IP or hostname of your database server ("localhost" if installed locally).
15 DataAccessParameters.ownDBHost=localhost
16 // The user which has access to the GO database.
17 DataAccessParameters.ownDBUser=blast2go
18 // The password for your database user.
19 DataAccessParameters.ownDBPassword=blast4it
20
21 // ** LoadFastaAlgoParameters **
22 // Select the sequences format.
23 // possible values: nucleotides, proteins, fastq
24 LoadFastaAlgoParameters.fastaFormat=nucleotides
25 // In Blast2GO sequence names have to be unique. Decide what you want Blast2GO to do
26 // in case of double IDs.
27 // possible values: skip, replace, add_with_new_id
28 LoadFastaAlgoParameters.handleDoubleIds=replace
29
30 // ** ImportDatAlgoParameters **
31 // In Blast2GO sequence names have to be unique. Decide what you want Blast2GO to do
32 // in case of double IDs.
33 // possible values: skip, replace
34 ImportDatAlgoParameters.handleDoubleIds=skip
35
36 // ** ImportBlastResultsAlgoParameters **
37 // All Blast Hits whose description line contains the provided text will be removed
38 // from the result list.
39 ImportBlastResultsAlgoParameters.blastHitDescFilter=
40 // The number of sequence alignments you want to retrieve.
41 // min: 0 max: 500
42 ImportBlastResultsAlgoParameters.numberOfHits=20
43 // Cutoff value for the minimal length of the first HSP of a Blast Hit, used to
44 // exclude Hits with only small local alignments from the Blast result. The given
45 // length corresponds to amino-acids or nucleotides depending on the type of the
46 // performed Blast.
47 // min: 0 max: 5000
48 ImportBlastResultsAlgoParameters.blastMinHSPLength=33
49 // Define the character used to split the description line of the Blast XML result
50 // description.
51 // Please provide only one character. Default is "|".
52 // (Please see also the "Hit description position" and "Join Hit ID and Hit Def"
53 // parameter).
54 ImportBlastResultsAlgoParameters.blastSeparator=|
55 // Please provide the position of the description text within the Blast Hit
56 // definition.
57 // Please provide a value between 1 and 9. Default is 5.
58 // (Please see also the "Separator" and "Join Hit ID and Hit Def" parameter).
59 // min: 0 max: 10
60 ImportBlastResultsAlgoParameters.blastHitDescPos=5
61 // When reading the Blast XML result, the Hit definition tag and the Hit description
62 // tag will be closed before being analyzed
```

## Windows: Notepad++

(<https://notepad-plus-plus.org/>)

- Install Notepad++
- Click Plugins->Plugin Admins and install NPPFTP
- In the NPPFTP, click “Setting” -> “Profile Setting”, set new profile with connection type “SFTP”.

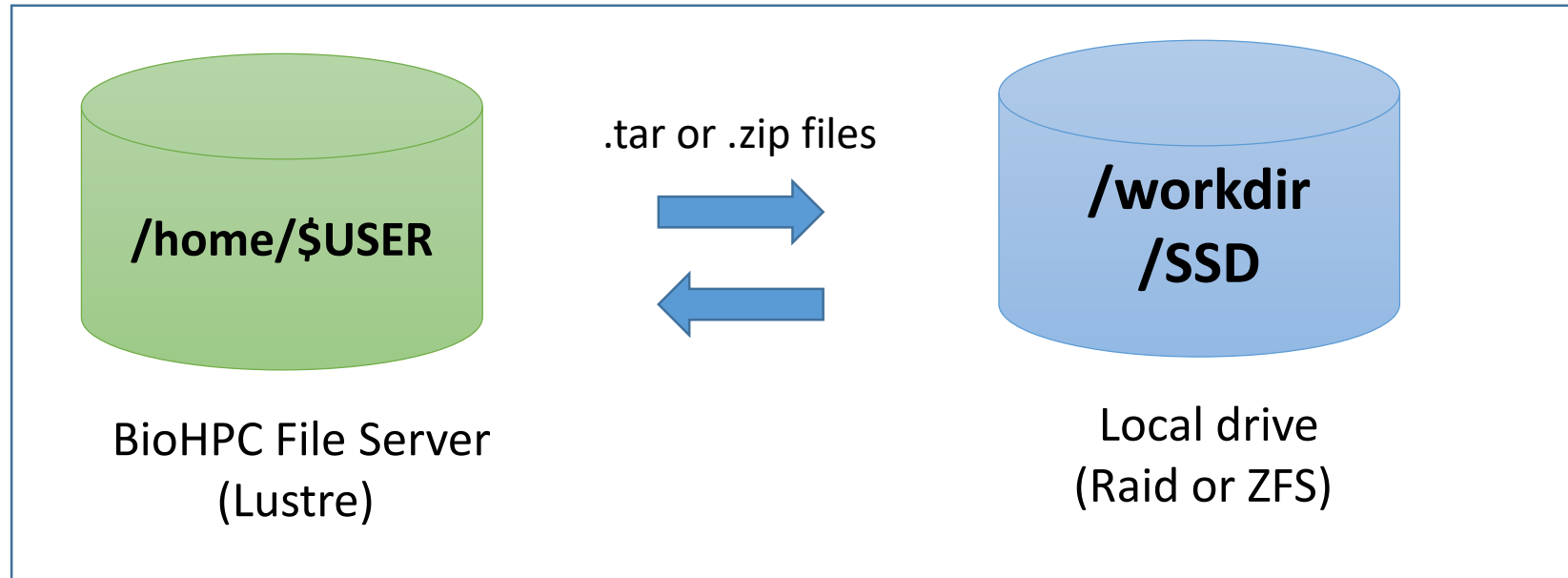
## Mac: BEdit (free mode)

(<https://www.barebones.com/products/bbedit/>)

- Open remote file from SFTP Server from the File menu

# Dealing with large number of files

## Data storage/transfer



### Avoid large number of file:

- File system is good for large size, bad for large file numbers;
- Very slow data transfer ;

# Solution: Archive into one single file

## tar

- Commonly used in Linux;
- Slow to check content or access single file;
- Compression rely on other technology;

```
# archive
tar cf myDataDir.tar myDataDir

#unarchive
tar xf myDataDir.tar
```

- Use “-z” to compress. E.g.  
tar cfz myDataDir.tar.gz myDataDir

## zip

- Not commonly used in Linux;
- Very fast to check content or access single file;
- Built-in compression;

```
# archive
zip -0 -r myDataDir.zip myDataDir

#unarchive
unzip myDataDir.zip
```

- “-0” suppresses compression, good to use if the files were already compressed.

# Accessing files in archive

tar: very slow

```
# list content
```

```
tar --list --file=myDataDir.tar
```

```
#extract a file
```

```
tar --extract --file=myDataDir.tar  
myFile
```

\* Use --to-stdout to stream to stdout

zip: very fast

```
# list content
```

```
less myDataDir.zip
```

```
zipinfo myDataDir.zip dataset1/*
```

```
# extract a file
```

```
#to a new file
```

```
unzip -j myDataDir.zip <path/>myFile
```

```
#to stdout
```

```
unzip -p myDataDir.zip <path/>myFile
```

## **A few suggestions:**

- Organize files into subdirectories. Too many files (>~1000) in a single directory would be difficult to “ls”.
- Too big tar file (>100G) could be difficult to transfer through network.
- Compression saves storage, but slow to process. Do not compress compressed files, or over-compression (set compression level).

# Run software on large number of files

## Create a task list

```
ls *fastq.gz | while read file; \  
do echo "STAR --runThreadN 4 --genomeDir REFERENCE --readFilesIn ${file}; \  
done;
```

A text file will be created

```
STAR --runThreadN 4 --genomeDir REFERENCE --readFilesIn sample1.fastq.gz  
STAR --runThreadN 4 --genomeDir REFERENCE --readFilesIn sample2.fastq.gz  
STAR --runThreadN 4 --genomeDir REFERENCE --readFilesIn sample3.fastq.gz
```

## Process the tasks in parallel

```
export PATH=/programs/parallel/bin:$PATH  
parallel --jobs 24 --timeout 1h < tasks
```