UNIX III lab: Data processing with pipes

This is the second half of UNIX III, as a lab / reading to work through on your own or in a lab session. It uses the class data files and assumes you have done UNIX II (pipes, redirection, `grep` basics) and the first part of UNIX III (variables, loops and scripts).

Setup

If you already did the setup in UNIX III, just go to that folder and set the locale:

```
cd ~/bigdata/gen220/unix3/data
export LC_ALL=C
```

Otherwise, do the UNIX III setup first: it makes the folder, explains `export LC_ALL=C` (all outputs here were made with it, so `sort` and `join` match), and downloads these files:

File	What it is
`codon_table.txt`	the genetic code: codon, one-letter amino acid, amino acid name (tab-separated)
`numbers.txt`	100 numbers between 0 and 99, one per line
`rice_random_exons.bed`	1000 rice exons in BED format: chromosome, start, end
`yeast_orfs-to-chr1.FASTA.tab`	yeast genes searched against chromosome I, BLAST-style tabular output
`S_cerevisiae.gff3.gz`	yeast genome annotation (GFF3) from SGD
`S_cerevisiae.ORFs.fasta.gz`	DNA sequences of all yeast ORFs
`threatened-species.csv.gz`	IUCN Red List species and their threat category (CSV)

The tools

Most bioinformatics files are text tables: one record per line, columns separated by tabs (or commas). A small set of tools, chained together with pipes, can answer a surprising number of questions about them:

Tool	Does
`head`, `tail`	first or last lines; `tail -n +2` skips a header line
`wc -l`	count lines
`grep`	keep lines matching a pattern
`cut`	keep some columns
`sort`	sort lines (by text, number, one or more columns)
`uniq`	collapse repeated lines, count them with `-c`
`tr`	replace or delete single characters
`paste`	put files or lines side by side
`join`	combine two tables on a shared column
`sed`	find and replace, delete or print lines
`awk`	a small language for column data: filter, calculate, summarize

Build pipelines one step at a time. Run the first command with `| head`, check the output, then add the next step. Everything in this lab is run from the `data` folder:

```
cd ~/bigdata/gen220/unix3/data
```

Looking at a table

Our "BLAST" table has 12 tab-separated columns. It was made with `fasta36`, which writes the same format as `blastn -outfmt 6`:

Column	Name	Meaning
1	qseqid	query sequence name
2	sseqid	subject (database sequence) name
3	pident	percent identity
4	length	alignment length
5	mismatch	number of mismatches
6	gapopen	number of gap openings
7, 8	qstart, qend	where the alignment starts and ends in the query
9, 10	sstart, send	where the alignment starts and ends in the subject
11	evalue	E-value
12	bitscore	bit score

```
wc -l yeast_orfs-to-chr1.FASTA.tab
head -n 3 yeast_orfs-to-chr1.FASTA.tab
```

```
252 yeast_orfs-to-chr1.FASTA.tab
YAL027W Chr_I   100.00  786 0   0   1   786 94688   95473   3.3e-196    676.8
tL(CAA)A    Chr_I   100.00  44  0   0   39  82  181205  181248  6e-10   54.8
tL(CAA)A    Chr_I   100.00  38  0   0   1   38  181135  181172  3.6e-08 48.9
```

Tabs don't line up the columns nicely. `column -t` pads them with spaces, and `less -S` lets you scroll sideways through wide tables instead of wrapping lines (use the arrow keys, `q` to quit):

```
head -n 3 yeast_orfs-to-chr1.FASTA.tab | column -t
column -t yeast_orfs-to-chr1.FASTA.tab | less -S
```

```
YAL027W   Chr_I  100.00  786  0  0  1   786  94688   95473   3.3e-196  676.8
tL(CAA)A  Chr_I  100.00  44   0  0  39  82   181205  181248  6e-10     54.8
tL(CAA)A  Chr_I  100.00  38   0  0  1   38   181135  181172  3.6e-08   48.9
```

To see whether a file uses tabs or spaces, `cat -A` shows tabs as `^I` and line ends as `$` (Windows files show `^M$`):

```
head -n 2 codon_table.txt | cat -A
```

```
ATT^II^IIsoleucine$
ATC^II^IIsoleucine$
```

grep: keep lines that match

You met `grep`, `-c` and `-i` in UNIX II. A few more options:

Option	Meaning
`-c`	count matching lines instead of printing them
`-v`	invert: keep lines that do **not** match
`-w`	match whole words only
`-i`	ignore upper/lower case
`-E`	extended regular expressions, e.g. `A|B` means A or B
`-n`	show line numbers
`-m N`	stop after N matches

`-w` often matters. `leucine` also matches Iso**leucine**, and `Chr1` also matches `Chr10`, `Chr11` and `Chr12`:

```
grep -c -i leucine codon_table.txt
grep -c -i -w leucine codon_table.txt
grep -c Chr1 rice_random_exons.bed
grep -c -w Chr1 rice_random_exons.bed
```

```
grep -c -v -w Chr1 rice_random_exons.bed
grep -E 'Asparagine|Aspartic' codon_table.txt
```

```
9
6
328
146
854
AAT N   Asparagine
AAC N   Asparagine
GAT D   Aspartic
GAC D   Aspartic
```

^ means "at the start of the line", so `grep -v '^#'` removes comment lines, and `grep -c '^>'` counts FASTA sequences.

cut: keep some columns

`cut -f` keeps columns (fields) by number; list several with commas or ranges like `1-4`. The default separator is a tab; use `-d` to choose another, such as a comma:

```
cut -f 1,3 codon_table.txt | tail -n 3
zcat threatened-species.csv.gz | head -n 3 | cut -d , -f 2,8,13
```

```
TAA Stop
TAG Stop
TGA Stop
kingdom_name,scientific_name,category
PLANTAE,Eugenia oreophila,LR/lc
PLANTAE,Eugenia orites,LR/cd
```

`cut` can't reorder columns (`cut -f 3,1` prints them in the order 1, 3); use `awk` for that.

A CSV warning. Real CSV files can have commas *inside* a field, protected by double quotes. This file has plenty, like `"(Ridl.) Orel, Peter G.Wilson, Curry &amp; Luu"` in the taxonomic authority column. `cut -d ,` doesn't know about quotes, so every such row gets shifted and the counts come out wrong:

```
zcat threatened-species.csv.gz | cut -d , -f 13 | sort | uniq -c | sort -nr |
    head -n 3
```

```
  79231
  28632 LC
  10102 EN
```

The correct answer, from a real CSV reader, is 75247 species in category `LC` (Least Concern). The early columns, before any quoted field, are fine: `cut -d , -f 2` counts the kingdoms correctly. For CSV files with quotes use Python (`csv` module or pandas) or R, which we cover later. (gawk can cope using `FPAT`, see the gawk manual.) Tab-separated files avoid the problem, which is one reason bioinformatics uses them.

sort and uniq

`sort` sorts lines. By default it compares text, character by character:

Option	Meaning
`-n`	numeric sort
`-r`	reverse (largest first)
`-k 2,2`	sort on column 2 only (from column 2 to column 2)
`-k 2,2n`	sort on column 2, numerically
`-t ,`	columns are separated by `,` (default: blanks and tabs)

Option	Meaning
`-u`	keep only one copy of identical lines
`-V`	"version" sort: Chr2 before Chr10

Sorting numbers as text puts `13` before `2`:

```
head -n 5 numbers.txt | tr '\n' ' '; echo
sort numbers.txt | head -n 5 | tr '\n' ' '; echo
sort -n numbers.txt | head -n 5 | tr '\n' ' '; echo
sort -nr numbers.txt | head -n 5 | tr '\n' ' '; echo
```

```
36 57 51 15 87
0 1 13 13 13
0 1 2 3 4
96 95 95 94 92
```

(`tr '\n' ' '` turns the lines into one line, to save space. `echo` adds the final newline back.)

`uniq` removes **adjacent** repeated lines, so always `sort` first. `uniq -c` counts how many times each line occurs. How many different numbers are there, and which come up most often?

```
sort -n numbers.txt | uniq | wc -l
sort -n numbers.txt | uniq -c | head -n 4
```

```
62
      1 0
      1 1
      1 2
      1 3
```

`sort | uniq -c | sort -nr | head` is one of the most useful idioms in UNIX: "what are the most common values, and how many of each?"

```
sort numbers.txt | uniq -c | sort -nr | head -n 5
```

```
      4 91
      4 54
      4 32
      3 57
      3 22
```

(`sort -u` gives the same list as `sort | uniq`. `uniq -d` prints only the lines that are repeated.)

Sorting on columns

`-k` chooses which column(s) to sort on. Always give a start and an end (`-k1,1`); `-k1` alone means "from column 1 to the end of the line". To sort genomic features by chromosome, then by start position numerically, use two keys:

```
head -n 3 rice_random_exons.bed
sort -k1,1 -k2,2n rice_random_exons.bed | head -n 3
```

```
Chr7    21408673    21408826
Chr9    16031526    16031938
Chr11   4762531 4762595
```

```
Chr1    12152   12435
Chr1    98088   98558
Chr1    216884  217664
```

This is how most tools (bedtools, tabix, IGV) want BED files sorted. Sorted as text, `Chr10` comes before `Chr2`. `-V` sorts the numbers inside names properly:

```
cut -f 1 rice_random_exons.bed | sort -u | tr '\n' ' '; echo
cut -f 1 rice_random_exons.bed | sort -u -V | tr '\n' ' '; echo
```

```
Chr1 Chr10 Chr11 Chr12 Chr2 Chr3 Chr4 Chr5 Chr6 Chr7 Chr8 Chr9 ChrSy
Chr1 Chr2 Chr3 Chr4 Chr5 Chr6 Chr7 Chr8 Chr9 Chr10 Chr11 Chr12 ChrSy
```

Sort the BLAST hits by percent identity (column 3, numeric, highest first) and show the query and identity:

```
sort -k3,3nr yeast_orfs-to-chr1.FASTA.tab | cut -f 1,3 | head -n 3
```

```
HRA1    100.00
YAL001C 100.00
YAL002W 100.00
```

Lines that tie on the key are put in order using the whole line, which is why these came out alphabetically.

tr: translate characters

`tr` replaces characters one for one, or deletes them with `-d`. It only reads from STDIN (a pipe or `<`), never from a file name:

```
echo "atgcgtacNNNgtt" | tr 'acgt' 'ACGT'
echo "atgcgtacNNNgtt" | tr '[:lower:]' '[:upper:]'
echo "ATGCGTACNNNGTT" | tr -d 'N'
echo "ATGCGTAC" | tr 'ACGT' 'TGCA' | rev      # reverse complement
head -n 3 codon_table.txt | tr '\t' ','       # tabs to commas
```

```
ATGCGTACNNNGTT
ATGCGTACNNNGTT
ATGCGTACGTT
GTACGCAT
ATT,I,Isoleucine
ATC,I,Isoleucine
ATA,I,Isoleucine
```

How many bases are in all the yeast ORFs? Remove the header lines, delete the newlines, and count characters:

```
zcat S_cerevisiae.ORFs.fasta.gz | grep -v '^>' | tr -d '\n' | wc -c
```

```
9078756
```

paste: side by side

`paste` joins files line by line, putting a tab between them (`-d` chooses another separator). `paste -s` joins all the lines of one file into a single line:

```
cut -f 1 codon_table.txt | head -n 3 > codons.txt
cut -f 3 codon_table.txt | head -n 3 > aminoacids.txt
paste codons.txt aminoacids.txt
paste -s -d , codons.txt
rm codons.txt aminoacids.txt
```

```
ATT Isoleucine
ATC Isoleucine
ATA Isoleucine
ATT,ATC,ATA
```

A handy trick: `paste - - - -` reads four lines at a time from STDIN and puts them on one line, turning each 4-line FASTQ record into one row of a table:

```
zcat ../pipeline/reads/wt_1_R1.fastq.gz | paste - - - - | cut -f 1,2
```

```
@wt_1_1/1    ACGTTGCAAGGT
@wt_1_2/1    ACGTTGCAAGGT
@wt_1_3/1    ACGTTGCAAGGT
```

join: combine two tables on a shared column

`paste` just puts lines side by side. `join` matches rows that have the same value in a key column, like a database or a spreadsheet lookup. **Both files must be sorted on the key column** (with the same `LC_ALL` setting). `-t $'\t'` says the columns are tab-separated.

Our BLAST table only has systematic gene names (YAL001C). The standard names (TFC3) are in the FASTA headers of the ORF file:

```
zcat S_cerevisiae.ORFs.fasta.gz | grep '^>' | head -n 1 | cut -c 1-70
```

```
>YAL001C TFC3 SGDID:S000000001, Chr I from 151006-147594,151166-151097
```

Build a table of systematic name and gene name: keep the header lines, keep the first two space-separated words, remove the `>`, turn the space into a tab, and sort:

```
zcat S_cerevisiae.ORFs.fasta.gz | grep '^>' | cut -d ' ' -f 1,2 | tr -d '>' |
    tr ' ' '\t' | sort -k1,1 > gene_names.tsv
wc -l gene_names.tsv
```

```
6713 gene_names.tsv
```

(A line ending in `|` continues on the next line.) Next take the best hit for each query (explained in the awk section below), keep a few columns, and sort on the query name:

```
sort -k1,1 -k12,12nr yeast_orfs-to-chr1.FASTA.tab | awk '!seen[$1]++' |
    cut -f 1,3,9,10 | sort -k1,1 > best_hits.tsv
join -t $'\t' best_hits.tsv gene_names.tsv | head -n 4
```

```
YAL001C 100.00  147596  149990  TFC3
YAL002W 100.00  143709  147533  VPS8
YAL003W 100.00  142621  143162  EFB1
YAL004W 100.00  140762  141409  YAL004W
```

`join` only prints rows found in both files. `-v 1` prints the rows of file 1 that had **no** match - here, the queries that are tRNAs and other RNA genes, not ORFs:

```
join -t $'\t' -v 1 best_hits.tsv gene_names.tsv | cut -f 1 | tr '\n' ' '; echo
```

```
HRA1 YAR062W snR18 tA(UGC)A tL(CAA)A tP(UGG)A tS(AGA)A
```

If `join` complains that a file is "not in sorted order", re-sort both files on the key with the same `LC_ALL` setting.

sed: stream editor

`sed` edits text as it streams past. The most common use is find-and-replace, `s/old/new/`, which replaces the first match on each line; add `g` ("global") to replace every match:

```
echo "Chr1 Chr1 Chr1" | sed 's/Chr/chr/'
echo "Chr1 Chr1 Chr1" | sed 's/Chr/chr/g'
head -n 2 yeast_orfs-to-chr1.FASTA.tab | sed 's/Chr_I/chrI/'
zcat S_cerevisiae.ORFs.fasta.gz | head -n 1 | sed 's/ .*//'    # keep only the ID
```

```
chr1 Chr1 Chr1
chr1 chr1 chr1
YAL027W chrI    100.00  786 0   0   1   786 94688   95473   3.3e-196    676.8
tL(CAA)A    chrI    100.00  44  0   0   39  82  181205  181248  6e-10   54.8
>YAL001C
```

The last one uses a *regular expression*: `.` means any character and `*` means "repeated any number of times", so `.*` is "a space and everything after it".

`sed` can also print or delete lines by number or by pattern. `-n` means "don't print lines unless I say so" and `p` means print:

```
sed -n '2,4p' codon_table.txt          # print lines 2 to 4
sed -n '/Stop/p' codon_table.txt       # print lines matching Stop (like grep)
sed '1d' ../pipeline/samples.tsv       # delete line 1 (the header)
```

```
ATC I   Isoleucine
ATA I   Isoleucine
CTT L   Leucine
TAA *   Stop
TAG *   Stop
TGA *   Stop
wt_1    wildtype    reads/wt_1_R1.fastq.gz  reads/wt_1_R2.fastq.gz
wt_2    wildtype    reads/wt_2_R1.fastq.gz  reads/wt_2_R2.fastq.gz
mut_1   mutant  reads/mut_1_R1.fastq.gz reads/mut_1_R2.fastq.gz
```

`sed '/^#/d'` deletes comment lines, the same as `grep -v '^#'`.

`sed -i` edits a file in place: it overwrites the original, with no undo. Try it on a copy first, or give a suffix so a backup is kept (`sed -i.bak` saves the original as `FILE.bak`):

```
cp codon_table.txt codon_copy.txt
sed -i.bak 's/Stop/STOP/' codon_copy.txt
tail -n 1 codon_copy.txt codon_copy.txt.bak
rm codon_copy.txt codon_copy.txt.bak
```

```
==> codon_copy.txt <==
TGA *   STOP

==> codon_copy.txt.bak <==
TGA *   Stop
```

(On a Mac, the built-in `sed` needs `sed -i ''` instead of `sed -i`. The cluster uses GNU sed.)

awk: a small language for columns

`awk` reads a file line by line, splits each line into fields `$1`, `$2`, ... (`$0` is the whole line), and runs your program on each one. The program goes in **single quotes** so the shell doesn't touch the `$`s. On the cluster, `awk` is GNU awk (`gawk`).

A program is a list of `pattern { action }`. The action runs on lines where the pattern is true. With no pattern it runs on every line; with no action it prints the line.

Printing columns

```
awk '{print $1, $3}' yeast_orfs-to-chr1.FASTA.tab | head -n 2
awk '{print $1, $NF}' yeast_orfs-to-chr1.FASTA.tab | head -n 2    # $NF: the last field
awk '{print $3, $1}' codon_table.txt | head -n 2                    # any order
```

```
YAL027W 100.00
tL(CAA)A 100.00
YAL027W 676.8
tL(CAA)A 54.8
Isoleucine ATT
Isoleucine ATC
```

By default `awk` splits on any run of spaces or tabs, and the comma in `print $1, $3` puts a space in the output. For real tab-separated files it is safer to split on tabs only (`-F'\t'`, in case a column contains spaces) and to print tabs between columns (set the **O**utput **F**ield **S**eparator, `OFS`). `-F ,` splits on commas:

```
awk -F'\t' -v OFS='\t' '{print $1, $3}' yeast_orfs-to-chr1.FASTA.tab | head -n 2
zcat threatened-species.csv.gz | awk -F , 'NR > 1 {print $2, $8}' | head -n 2
```

```
YAL027W 100.00
tL(CAA)A    100.00
PLANTAE Eugenia oreophila
PLANTAE Eugenia orites
```

`NR` is the line ("record") number, so `NR > 1` skips the header.

Filtering rows

A condition on its own prints the matching lines. Compare numbers with `<`, `<=`, `==`, `!=`, `>=`, `>`, text with `==`, and match regular expressions with `~`. Combine conditions with `&&` (and), `||` (or):

```
awk '$3 < 90' yeast_orfs-to-chr1.FASTA.tab | wc -l
awk '$3 < 90 && $4 >= 1000' yeast_orfs-to-chr1.FASTA.tab | cut -f 1-4
awk '$1 == "YAL001C"' yeast_orfs-to-chr1.FASTA.tab | cut -f 1-4
awk '$1 ~ /^t/' yeast_orfs-to-chr1.FASTA.tab | cut -f 1 | sort -u    # names starting t
```

```
102
YAL060W Chr_I   58.36   1160
YAL061W Chr_I   58.36   1160
YAL063C Chr_I   72.51   1357
YAL063C Chr_I   75.38   1044
YAR050W Chr_I   75.17   1047
YAL001C Chr_I   100.00  2395
YAL001C Chr_I   98.35   1212
YAL001C Chr_I   97.30   74
tA(UGC)A
tL(CAA)A
tP(UGG)A
tS(AGA)A
```

Unlike `grep`, `awk '$1 == "YAL001C"'` only matches in column 1, and it compares the whole column, so it won't also match `YAL001C-A`.

Calculating new columns

A BED file stores start and end positions, so the length of each feature is `$3 - $2`:

```
awk -v OFS='\t' '{print $1, $2, $3, $3 - $2}' rice_random_exons.bed | head -n 3
```

```
Chr7    21408673    21408826    153
Chr9    16031526    16031938    412
Chr11   4762531 4762595 64
```

(BED coordinates start counting at 0 and don't include the end, so `end - start` is the length. GFF and BLAST coordinates start at 1 and include both ends, so there the length is `end - start + 1`.)

In the BLAST table, the part of the query that aligned runs from column 7 (qstart) to column 8 (qend), and the part of the chromosome from column 9 (sstart) to 10 (send):

```
awk '{print $1, $8 - $7 + 1}' yeast_orfs-to-chr1.FASTA.tab | sort -k2,2n | head -n 3
```

```
YAL029C -4414
YAL024C -4306
YAL026C -4066
```

Negative lengths! **Check your data.** This file was made with `fasta36`, which marks hits on the reverse strand by writing the query coordinates backwards (qstart > qend); NCBI BLAST does it the other way, with sstart > send. Either way, take the absolute value:

```
awk '$7 > $8' yeast_orfs-to-chr1.FASTA.tab | wc -l
awk '{len = $8 - $7; if (len < 0) len = -len; print $1, len + 1}' \
    yeast_orfs-to-chr1.FASTA.tab | sort -k2,2nr | head -n 3
```

```
110
YAR050W 4614
YAL029C 4416
YAL024C 4308
```

(Programs can span several lines, and `;` separates statements.)

Summing, averages, min and max

Variables in `awk` don't need to be created first; they start as 0 (or empty). `BEGIN { }` runs before the first line and `END { }` runs after the last one - the place to print totals. `printf` prints formatted output: `%d` is a whole number, `%.1f` a number with 1 decimal place, `%s` text, and you add `\n` yourself:

```
awk '{total += $3 - $2} END {print "exons:", NR, "total bp:", total}' \
    rice_random_exons.bed
awk '{total += $3 - $2} END {printf "mean exon length: %.1f bp\n", total / NR}' \
    rice_random_exons.bed
```

```
exons: 1000 total bp: 369855
mean exon length: 369.9 bp
```

Counting and summing per group

An **associative array** is indexed by text instead of numbers, so `count[$1]++` keeps a separate count for every different value of column 1. In `END`, `for (key in array)` loops over them, in no particular order, so pipe the result to `sort`:

```
awk '{n[$1]++; len[$1] += $3 - $2}
     END {for (chr in n) printf "%s\t%d\t%.1f\n", chr, n[chr], len[chr] / n[chr]}' \
    rice_random_exons.bed | sort -k1,1V | head -n 4
```

```
Chr1    146 349.5
Chr2    123 436.2
Chr3    122 308.7
Chr4    92  423.3
```

(The `\` at the end of a line means "the command continues on the next line".)

The best hit for each query: `!seen[$1]++`

This short program prints only the **first** line for each value of column 1. `seen[$1]++` is 0 (false) the first time a name is seen and then counts up; `!` flips it, so the line is printed only the first time. Sort so the best hit comes first within each query - by query name, then bit score (column 12) highest first - and keep the first line of each:

```
sort -k1,1 -k12,12nr yeast_orfs-to-chr1.FASTA.tab | awk '!seen[$1]++' | head -n 3
sort -k1,1 -k12,12nr yeast_orfs-to-chr1.FASTA.tab | awk '!seen[$1]++' | wc -l
```

```
HRA1    Chr_I   100.00  564 0   0   1   564 99306   99869   3.9e-118    417.0
YAL001C Chr_I   100.00  2395    0   0   3483    1089    147596  149990  0   1850.3
YAL002W Chr_I   100.00  3825    0   0   1   3825    143709  147533  0   2826.4
127
```

Passing shell variables into awk

Inside single quotes, `$MIN` would be awk's field number `MIN`, not your shell variable. Pass values in with `-v`:

```
MIN=90
awk -v min="$MIN" '$3 >= min' yeast_orfs-to-chr1.FASTA.tab | wc -l
```

```
150
```

Questions answered with a pipeline

A GFF3 file has 9 tab-separated columns: sequence (chromosome), source, feature **type**, start, end, score, strand, phase, and attributes (`ID=...;Name=...`). Lines starting with `#` are comments.

```
zcat S_cerevisiae.gff3.gz | grep -v '^#' | cut -f 1-8 | sed -n '5,7p'
```

```
chrI    SGD telomeric_repeat    1   62  .   -   .
chrI    SGD gene    335 649 .   +   .
chrI    SGD CDS 335 649 .   +   0
```

How many of each type of feature are there?

```
zcat S_cerevisiae.gff3.gz | grep -v '^#' | cut -f 3 | sort | uniq -c |
    sort -nr | head -n 8
```

```
   7058 CDS
   6600 mRNA
   6600 gene
    484 noncoding_exon
    383 long_terminal_repeat
    377 intron
    352 ARS
    299 tRNA_gene
```

Which chromosomes have the most genes? Use `awk` to keep only the `gene` lines, because `grep gene` would also match `tRNA_gene`, and the word gene in the attribute column:

```
zcat S_cerevisiae.gff3.gz | awk -F'\t' '$3 == "gene"' | cut -f 1 | sort | uniq -c |
    sort -nr | head -n 3
```

```
    836 chrIV
    597 chrXV
    583 chrVII
```

Which chromosome has the most tRNA genes?

```
zcat S_cerevisiae.gff3.gz | awk -F'\t' '$3 == "tRNA_gene"' | cut -f 1 | sort | uniq -c |
    sort -nr | head -n 3
```

```
     36 chrVII
     28 chrIV
     24 chrmt
```

What are the mean, shortest and longest gene lengths? Keep our own counter `n`, because `NR` counts *all* lines in the file, not just the gene lines:

```
zcat S_cerevisiae.gff3.gz | awk -F'\t' '
    $3 == "gene" {
        n++
        len = $5 - $4 + 1
        total += len
        if (n == 1 || len < min) min = len
        if (len > max) max = len
```

```
    }
    END {printf "genes: %d  mean: %.1f  min: %d  max: %d\n", n, total / n, min, max}'
```

```
genes: 6600  mean: 1345.9  min: 51  max: 14733
```

Which gene is the longest? Print the length first, sort on it, and look at the top line:

```
zcat S_cerevisiae.gff3.gz |
    awk -F'\t' -v OFS='\t' '$3 == "gene" {print $5 - $4 + 1, $1, $9}' |
    sort -k1,1nr | head -n 1 | cut -c 1-60
```

```
14733   chrXII  ID=YLR106C;Name=YLR106C;gene=MDN1;Alias=MDN1,AA
```

MDN1, midasin, a huge protein needed for ribosome assembly.

BLAST: how many queries have more than one hit? Count hits per query, then count the queries with a count above 1:

```
cut -f 1 yeast_orfs-to-chr1.FASTA.tab | sort | uniq -c | sort -nr | head -n 3
cut -f 1 yeast_orfs-to-chr1.FASTA.tab | sort | uniq -c | awk '$1 > 1' | wc -l
```

```
     34 YAR062W
     14 YAR050W
     12 YAL063C
30
```

BLAST: how many different queries hit something else at less than 90% identity (probably a related gene, a paralog, rather than the gene itself)?

```
awk '$3 < 90' yeast_orfs-to-chr1.FASTA.tab | cut -f 1 | sort -u | wc -l
```

```
22
```

Exercises

1. **Top N.** From `codon_table.txt`: how many codons does each amino acid have? List them from most to fewest. Which amino acids have only one codon?
2. **awk filters.** From `rice_random_exons.bed`: how many exons are longer than 1000 bp? What is the total length of the exons on Chr3? What are the coordinates of the longest exon?
3. **GFF.** How many genes are on the minus strand (column 7) of chromosome V? What is the mean length of a tRNA gene? Make a table of the number of genes on each chromosome (hint: `sort -k1,1V` won't help with Roman numerals - why not?).
4. **BLAST.** For each query in the BLAST table print the query name, its number of hits and its best percent identity, as a tab-separated table, sorted by number of hits.

Further reading

- Vince Buffalo, *Bioinformatics Data Skills*, chapter 7 "Unix Data Tools" (O'Reilly Learning; check the UCR Library for access)
- The GNU Awk User's Guide
- explainshell.com - paste in a command and it explains each part
- Software Carpentry, The Unix Shell: Pipes and Filters

Next

The same questions answered in Python are in Python II and Python III; with SQL in DuckDB; and with pandas in Python IV.