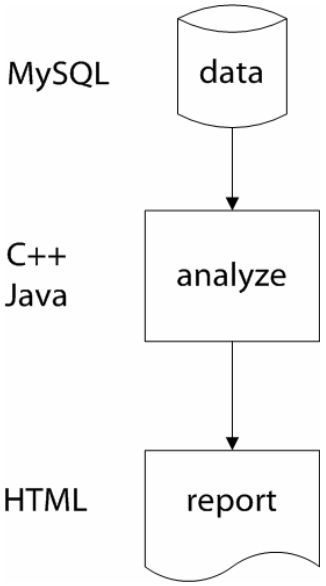
2.1.2.4.1

Command-Line Data Analysis and Reporting – Session 1

- you don't need to write scripts to carry out data mining and analysis – even fairly complex cases
- UNIX provides a ready toolbox of **text processing** tools that make this possible
  - when data is represented in plain text, command-line binaries that **search**, **extract**, **replace** text can be used
- each tool is designed to perform a specific task, and output of one can be **piped** to another

Build Separable and Reusable Analysis Components

- leverage strengths of languages and formats
- adopt workflow that incorporates data analysis and mining at all levels
  - simple tools for simple questions
    - Q: what is the mean of the third column? = SIMPLE
    - Q: what does this data mean? = HARD
- use flat-file output as much as possible
  - keep number of fields in each line constant
  - separate words within a field by a different delimiter
    - e.g. “1 2 apple_banana 5” vs “1 2 apple banana 5”
- translate to a more complex format if you specifically require
  - avoid visual formatting for large data sets

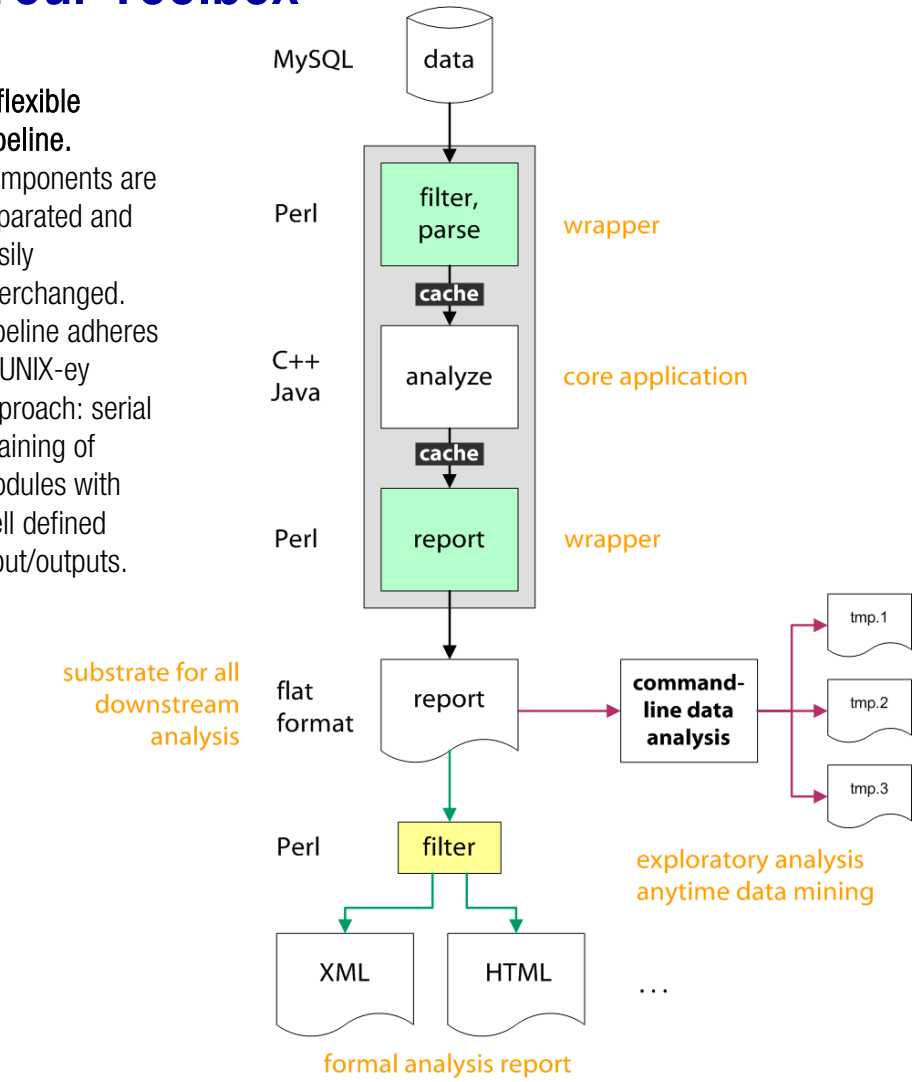


An inflexible pipeline.
A request for a different report format is likely to generate a lot of work.

Make the Command-Line Part of Your Toolbox

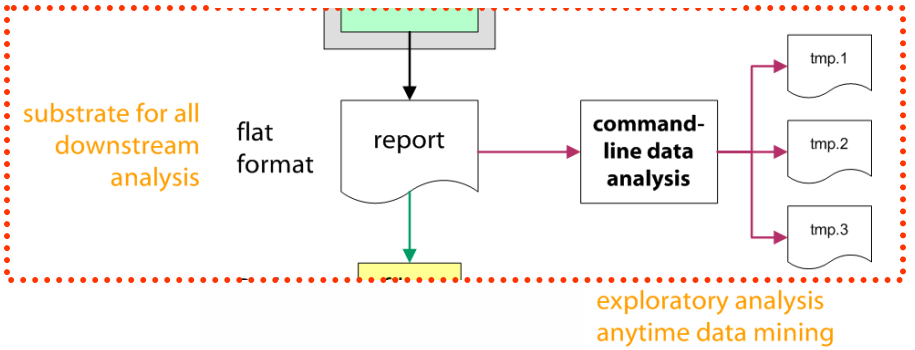
- you *will* need to perform exploratory analysis on your data
  - rapid, throw-away analysis forms the basis of prototype building
- eliminate one-off scripts by combining command-line tools and flexible I/O **"prompt tool"** scripts
- apply light weight tools to answer quick "research" questions
- apply formal process design for lengthier analysis and production pipelines

A flexible pipeline. Components are separated and easily interchanged. Pipeline adheres to UNIX-ey approach: serial chaining of modules with well defined input/outputs.



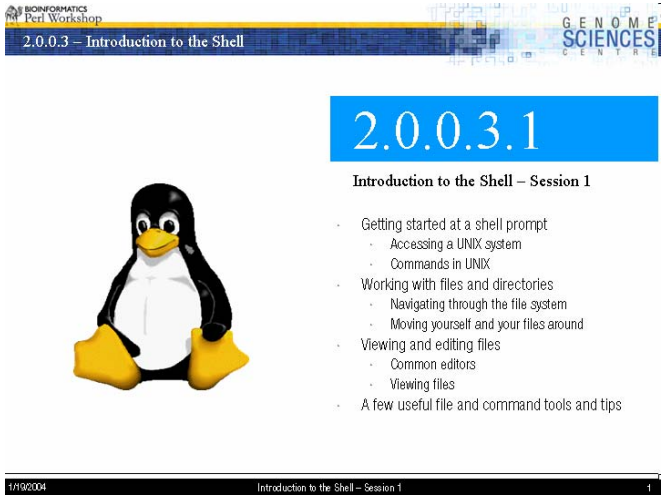
Things We'll Cover

- recipes for creating useful data reports
  - maximize utility,
  - limit complexity and effort
- ways to manipulate your text reports
  - command-line methods
  - specialized prompt tools
    - statistics
    - column management (a la **cut**)
    - line filters (a la **grep**)
    - histogramming (a la **uniq**)
- analysis idioms with common tools
  - /bin, /usr/bin, and bash
  - command-line Perl
- rejuvenate/discover your passion for the prompt



What You Will Need

- basic knowledge of UNIX
  - file management
  - notion of a pipe and redirect
- willingness to explore the GUI*less* land of the command line
  - you can't break anything by experimenting…
  - … except delete all your files
  - don't experiment with **rm**
- refresh your basic UNIX knowledge with Erin's 2.0.0.3 course



Workshop 2.0.0.3. Review the course slides to brush up on UNIX fundamentals. Erin covers file management and command line tools like grep and sort.

What You Will Learn

- command-line voodoo
  - increase productivity
  - ask more questions
  - interrogate data in complex ways
  - relieve yourself from the **dependence** of other people's black-box parsers and scripts for simple tasks
  - **eliminate** need for **formal DB** layer in pilot/prototype projects
- best practices for generating text reports
  - how to make a flat-file report and be proud of it
- how to deal with other people's BAD and NASTY file formats

Motivational Example


```
#
MCF7_1-100G11
               BES:           targetedESPplate5B02TF
                              Mapping success: 1
                              Reason for failure: 0
                                          BES with 270 bp of unique sequence is loca
starting at 24 in BES at 99.7772828507795% identity)
                                                          orientation: Plus
               BES:           targetedESPplate5B02TR
                              Mapping success: 1
                              Reason for failure: 0
                                          BES with 184 bp of unique sequence is located on chr1 starting at 111122200 (427 bp
starting at 3 in BES at 99.0632318501171% identity)
                                                          orientation: Minus

                              PAIRED!!!This clone has apparent length of 123561 bp
. . .
clone: MCF7_1-124I17
               BES: targetedESPplate5F05TF of > 672 bp: chr17 @ 59314680 (Minus)
                              15 BES within +/- 50000, of which 4 from translocations, 0 from clones with wrong end orientation, 0
from clones with wrong apparent size
               BES: targetedESPplate5F05TR of > 405 bp: chr4 @ 129284290 (Plus)
                              0 BES within +/- 50000, of which 0 from translocations, 0 from clones with wrong end orientation, 0
from clones with wrong apparent size
clone: MCF7_1-124I19
               BES: targetedESPplate5G05TF of > 519 bp: chr20 @ 53161812 (Plus)
                              11 BES within +/- 50000, of which 2 from translocations, 0 from clones with wrong end orientation, 0
from clones with wrong apparent size
               BES: targetedESPplate5G05TR of > 462 bp: chr3 @ 63951454 (Plus)
                              24 BES within +/- 50000, of which 9 from translocations, 0 from clones with wrong end orientation, 0
from clones with wrong apparent size
. . .
Screwed up clone: MCF7_1-69H4 - targetedESPplate3A10TR : chr9 @ 38944527 etc - multiple HSPs!!!
               But $multiple = 1 and $longest = 0
               0 666
               $q = 1
```

For man or machine? Decide!

This isn't meant for human eyes. But it's not designed well for automated parsing. What is the target audience?

Unfortunately, it was me.

Motivational Example


```
#
MCF7_1-100G11
               BES:           targetedESPplate5B02TF
                              Mapping success: 1
                              Reason for failure: 0
                                          BES with 270 bp of unique sequence is loca
starting at 24 in BES at 99.7772828507795% identity)
                                                          orientation: Plus
               BES:           targetedESPplate5B02TR
                              Mapping success: 1
                              Reason for failure: 0
                                          BES with 184 bp of unique sequence is located on chr1 starting at 111122200 (427 bp
starting at 3 in BES at 99.0632318501171% identity)
                                                          orientation: Minus

                              PAIRED!!!This clone has apparent length of 123561 bp
. . .
clone: MCF7_1-124I17
               BES: targetedESPplate5F05TF of > 672 bp: chr17 @ 59314680 (Minus)
                              15 BES within +/- 50000, of which 4 from translocations, 0 from clones with wrong end orientation, 0
from clones with wrong apparent size
               BES: targetedESPplate5F05TR of > 405 bp: chr4 @ 129284290 (Plus)
                              0 BES within +/- 50000, of which 0 from translocations, 0 from clones with wrong end orientation, 0
from clones with wrong apparent size
clone: MCF7_1-124I19
               BES: targetedESPplate5G05TF of > 519 bp: chr20 @ 53161812 (Plus)
                              11 BES within +/- 50000, of which 2 from translocations, 0 from clones with wrong end orientation, 0
from clones with wrong apparent size
               BES: targetedESPplate5G05TR of > 462 bp: chr3 @ 63951454 (Plus)
                              24 BES within +/- 50000, of which 9 from translocations, 0 from clones with wrong end orientation, 0
from clones with wrong apparent size
. . .
Screwed up clone: MCF7_1-69H4 - targetedESPplate3A10TR : chr9 @ 38944527 etc - multiple HSPs!!!
               But $multiple = 1 and $longest = 0
               0 666
               $q = 1
```

No English please

This report is over 6,000 lines long but contains phrases designed for legibility. Nobody will read 6,000 lines!

Motivational Example


```
#
MCF7_1-100G11
               BES:            targetedESPplate5B02TF
                               Mapping success: 1
                               Reason for failure: 0
                                        BES with 270 bp of unique sequence is loca
starting at 24 in BES at 99.7772828507795% identity)
                                                        orientation: Plus
               BES:            targetedESPplate5B02TR
                               Mapping success: 1
                               Reason for failure: 0
                                        BES with 184 bp of unique sequence is loca
starting at 3 in BES at 99.0632318501171% identity)
                                                        orientation: Minus

                               PAIRED!!!This clone has apparent length of 123561 bp
. . .
clone: MCF7_1-124I17
               BES: targetedESPplate5F05TF of > 672 bp: chr17 @ 59314680 (Minus)
                               15 BES within +/- 50000, of which 4 from translocations,
from clones with wrong apparent size
               BES: targetedESPplate5F05TR of > 405 bp: chr4 @ 129284290 (Plus)
                               0 BES within +/- 50000, of which 0 from translocations, 0
from clones with wrong apparent size
clone: MCF7_1-124I19
               BES: targetedESPplate5G05TF of > 519 bp: chr20 @ 53161812 (Plus)
                               11 BES within +/- 50000, of which 2 from translocations, 0 from clones with wrong end orientation, 0
from clones with wrong apparent size
               BES: targetedESPplate5G05TR of > 462 bp: chr3 @ 63951454 (Plus)
                               24 BES within +/- 50000, of which 9 from translocations, 0 from clones with wrong end orientation, 0
from clones with wrong apparent size
. . .
Screwed up clone: MCF7_1-69H4 - targetedESPplate3A10TR : chr9 @ 38944527 etc - multiple HSPs!!!
               But $multiple = 1 and $longest = 0
               0 666
               $q = 1
```

No complex grammar please

Parsing this report is a nightmare. What is the grammar? I have to write a parser (or at least describe the grammar) to make sure that I don't miss anything.

Single-line records please.

Avoid multi-line records. Parsing single-line records can be done in a **stateless** way – I don't have to remember the last line. This file requires that I keep track of at least two levels of context (clone and BES).

Motivational Example


```
#
MCF7_1-100G11
               BES:           targetedESPplate5B02TF
                              Mapping success: 1
                              Reason for failure: 0
                                             BES with 270 bp of unique sequence is loca
starting at 24 in BES at 99.7772828507795% identity)
                                                          orientation: Plus
               BES:           targetedESPplate5B02TR
                              Mapping success: 1
                              Reason for failure: 0
                                             BES with 184 bp of unique sequence is located on chr1 starting at 111122200 (427 bp
starting at 3 in BES at 99.0632318501171% identity)
                                                          orientation: Minus

                              PAIRED!!!This clone has apparent length of 123561 bp
. . .
clone: MCF7_1-124I17
               BES: targetedESPplate5F05TF of > 672 bp: chr17 @ 59314680 (Minus)
                              15 BES within +/- 50000, of which 4 from translocations, 0 from clones with wrong end orientation, 0
from clones with wrong apparent size
               BES: targetedESPplate5F05TR of > 405 bp: chr4 @ 129284290 (Plus)
                              0 BES within +/- 50000, of which 0 from translocations, 0 from clones with wrong end orientation, 0
from clones with wrong apparent size
clone: MCF7_1-124I19
               BES: targetedESPplate5G05TF of > 519 bp: chr20 @ 53161812 (Plus)
                              11 BES within +/- 50000, of which 2 from translocations, 0 from clones with wrong end orientation, 0
from clones with wrong apparent size
               BES: targetedESPplate5G05TR of > 462 bp: chr3 @ 63951454 (Plus)
                              24 BES within +/- 50000, of which 9 from translocations, 0 from clones with wrong end orientation, 0
from clones with wrong apparent size
. . .
Screwed up clone: MCF7_1-69H4 - targetedESPplate3A10TR : chr9 @ 38944527 etc - multiple HSPs!!!
               But $multiple = 1 and $longest = 0
               0 666
               $q = 1
```

Consistent format

This report is trying to communicate too much information and does so in at least three different formats.

Motivational Example


```
#
MCF7_1-100G11
               BES:           targetedESPplate5B02TF
                              Mapping success: 1
                              Reason for failure: 0
                                        BES with 270 bp of unique sequence is loca
starting at 24 in BES at 99.7772828507795% identity)
                                                        orientation: Plus
               BES:           targetedESPplate5B02TR
                              Mapping success: 1
                              Reason for failure: 0
                                        BES with 184 bp of unique sequence is loca
starting at 3 in BES at 99.0632318501171% identity)
                                                        orientation: Minus

                              PAIRED!!!This clone has apparent length of 123561 bp
. . .
clone: MCF7_1-124I17
               BES: targetedESPplate5F05TF of > 672 bp: chr17 @ 59314680 (Minus)
                              15 BES within +/- 50000, of which 4 from translocations,
from clones with wrong apparent size
               BES: targetedESPplate5F05TR of > 405 bp: chr4 @ 129284290 (Plus)
                              0 BES within +/- 50000, of which 0 from translocations, 0
from clones with wrong apparent size
clone: MCF7_1-124I19
               BES: targetedESPplate5G05TF of > 519 bp: chr20 @ 53161812 (Plus)
                              11 BES within +/- 50000, of which 2 from translocations,
from clones with wrong apparent size
               BES: targetedESPplate5G05TR of > 462 bp: chr3 @ 63951454 (Plus)
                              24 BES within +/- 50000, of which 9 from translocations,
from clones with wrong apparent size
. . .
Screwed up clone: MCF7_1-69H4 - targetedESPplate3A10TR : chr9 @ 38944527 etc - multiple HSPs!!!
               But $multiple = 1 and $longest = 0
               0 666
               $q = 1
```

Controlled vocabulary

Choose meaningful, short text flags instead of complicated descriptions. I found no less than 4 different ways in which a clone name is displayed

MCF7_1-100G11
MCF7_1-25e22
MCF7_37_F_I03
MCF737FI03TF

Are some entries redundant?

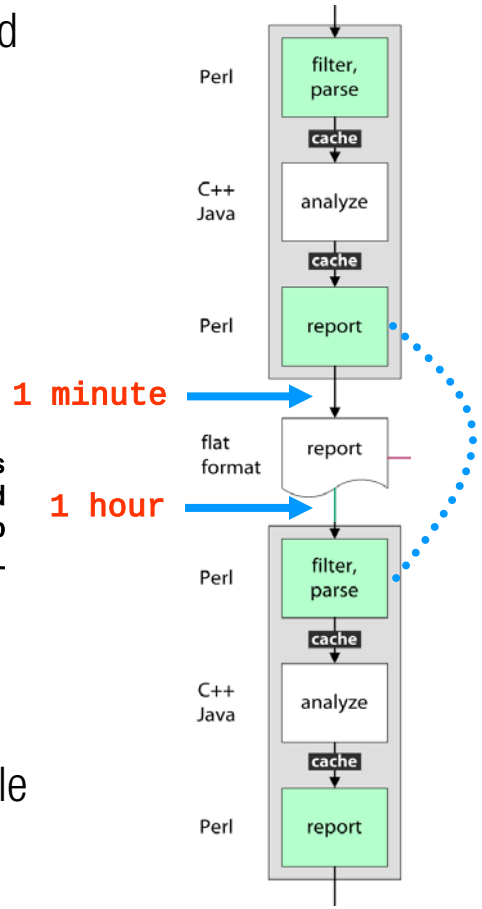
Mapping success: 1
Reason for failure: 0

Alternate Format

- had received the data in a simpler format, a lot of effort would be saved


```
. . .
M0154O21 10 81747525 10 81873318
M0155D17 - - - -
M0155F02 17 58506078 - -
M0155L05 - - - -
M0155O05 17 60433004 20 56433350
M0156B17 20 46815385 20 46975655
M0156I16 17 60402624 20 56433371
M0156K22 3 63983906 17 59333658
M0156N14 20 55865922 20 55984334
M0157C08 20 55834458 20 56005390
M0157C23 20 56412109 20 56476173
M0157E01 17 59766670 17 59913499
. . .
```

collaborator sends you their data and sends you to parsing hell



- if you are communicating data to someone, do it in a format that will allow them to recover your original data structure as quickly as possible
  - serialized object using Storable
  - CSV file, single-line records
  - XML

Lessons Learned?

- break the SHIFT keys on your keyboard
  - do we really need capital letters? no!
    - if it's not written in full English, skip capitalization
  - do not use capital letters in
    - your report files
    - your directory or file names

    - BASH will autocomplete filenames and commands when you hit TAB, but you need to know the case
    - /home/JDoe/Work/projects/SPECIAL/backup_Today/report.TXT – this is very annoying

- make parsing of your files as easy as possible for your collaborators
  - single-line records
  - same number of fields on each line
  - what is your **data-to-ink ratio**?
  - how quickly can *you* parse your own files?
  - comment with standard prefixes (e.g. # or //)

- are your files meant for a human or computer?
  - not both!
  - send the human a figure or diagram – they'll like you more :)

Report Formats

	pros	cons	example
serialized data structure	communicate complex data structures; extremely simple **easy to reconstitute** data; obviates parsing step; usually high data-to-ink ratio	requires sender/recipient **share same platform**; cannot be examined directly; a priori knowledge of format required to access data	highly targeted audience *e.g. application cache file*
XML, ASN.1	grammar is self-describing (sometimes); many parsers and viewers exist;	(can be) verbose - **abysmal** data-to-ink ratio; advanced features may be incompatible with some parsers; data payload is encapsulated and generally difficult to read directly; requires knowledge of format to manipulate;	sophisticated audience, complex records *e.g. Pubmed citation*
flat text file *application format*	parser may already exist (e.g. BLAST output); may be partially human-readable	may be difficult to parse if no parser exists; may be overwhelming in detail; sender has no (little) control over format; low data-to-ink ratio	target audience *e.g. SQL dump, BLAST alignments*
flat text file *CSV*	viewable at the prompt; no technical knowledge required; accessible by command-line tools; sender optimizes content for portability and clarity; easy to make, read and manipulate; cut/paste into applications	depending on format, some parsing is required; may lack detail and granularity; can have high data-to-ink ratio	simple records, all audiences

Example Report

- consider UCSC's genome assembly report (.agp)
  - compact
  - format is self-explanatory
    - gaps in assembly are reported in slightly different format, but this is ok because overall complexity of the file is low
  - lines do not have a constant number of fields
    - gap lines *may* have a comment
    - this isn't a big problem in this case because the optional comment is at the end of the line

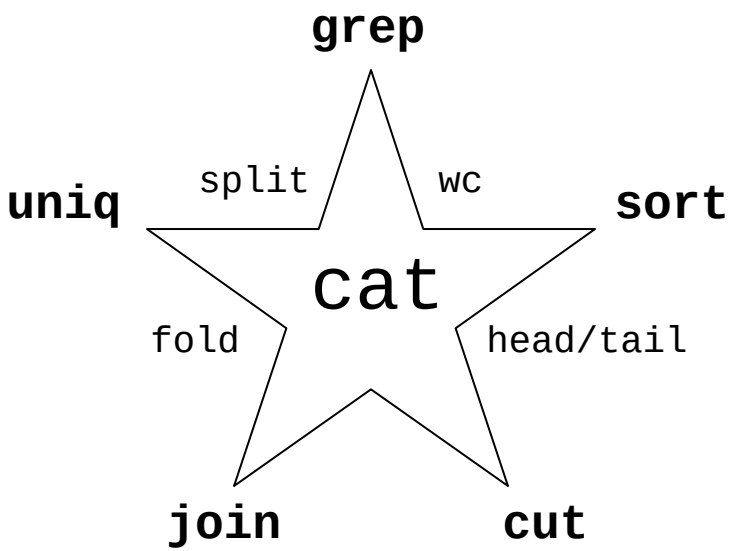
```
chr1   1      616    1      F      AP006221.1    36116         36731    -
chr1   617    167280 2      F      AL627309.15   241           166904   +
chr1   167281 217280 3      N      50000         clone         no       # Unfinished_sequence
chr1   217281 257582 4      F      AP006222.1    1             40302    +
chr1   257583 307582 5      N      50000         clone         no
chr1   307583 357582 6      N      50000         clone         no       # Unfinished_sequence
chr1   357583 511231 7      F      AL732372.15   1             153649   +
chr1   511232 561231 8      N      50000         clone         no
chr1   561232 672780 9      F      AC114498.2    1             111549   +
chr1   672781 852347 10     F      AL669831.13   1             179567   +
```

Basic Command Line Tools

- 10 text processing tools will suffice for most of your command-line processing
  - **grep**, **sort**, **cut**, **join**, **uniq** (extremely common)
  - **wc**, **head/tail** (common)
  - **fold**, **split** (infrequent)
  - **cat** (goes without saying)

- in addition, two text utilities are used for more complex tasks but still can be deployed at the command-line
  - **tr** – replace characters
  - **sed** – stream editor
  - **awk** – programming language designed for text processing

- heavy-weights can fit the bill, but don't their power keep you from knowing their lighter command line brethren
  - command-line **perl**



tr
sed
awk

perl

2.1.2.4 – Command-Line Data Analysis and Reporting

break down a complex command to its constituent elements, which perform tractable steps

think about the overall command in terms of simple steps like search, extract, sort, etc.

Command Line Idioms

- command-line tools are frequently combined to form **idioms**
  - patterns of commands that perform a specific, commonly needed task
  - relax – these look more complicated then they are

```
# list sorted by first column
sort file.txt

# extract the first column, sorted
sort file.txt | cut –d “ “ –f 1

# list of unique values seen in the first column
sort file.txt | cut –d “ “ –f 1 | uniq –c

# number of unique values seen in the first column
sort file.txt | cut –d “ “ –f 1 | uniq –c | wc
```

```
# number of unique values seen in
# the first column
sort –u –k 1,1 file.txt | wc
```

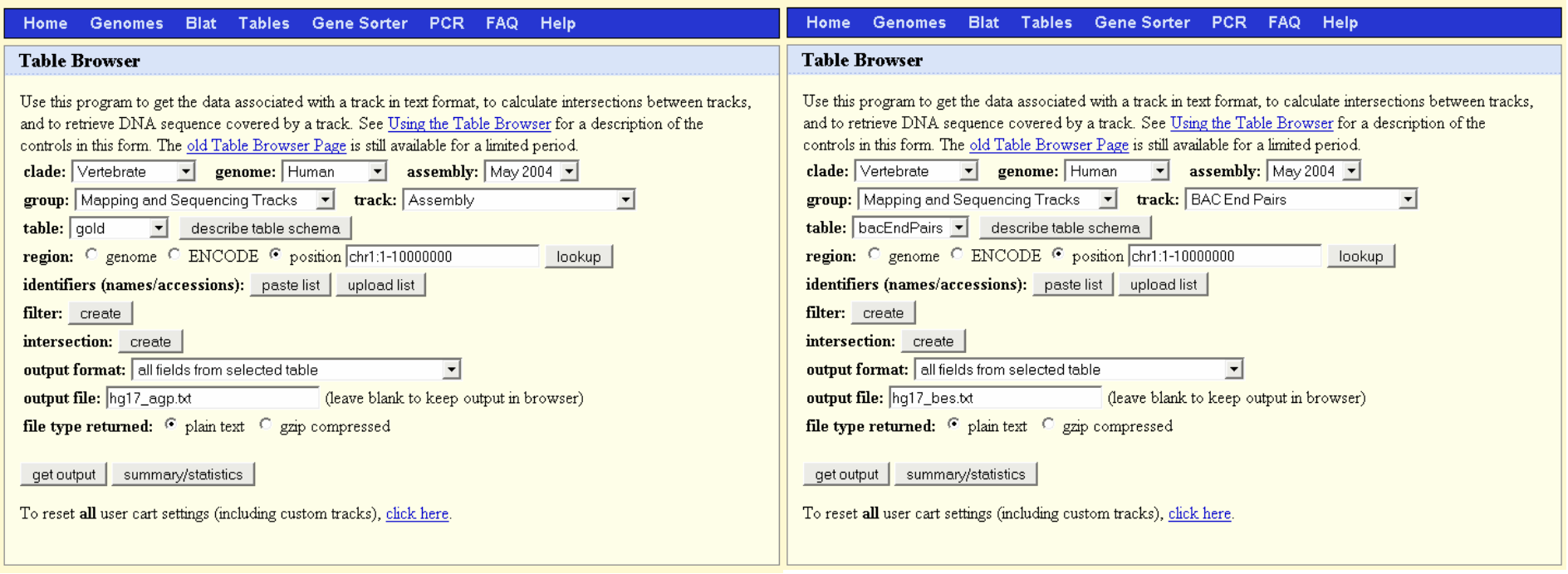
- the pipe “|” sends the output of one command to another

```
sort file.txt > tmp.1
cut –d “ “ –f 1 tmp.1 > tmp.2
uniq –c tmp.2

sort file.txt | cut –d “ “ –f 1 | uniq -c
```

Downloading Genomic Data – Substrate for Text Processing

- UCSC's table browser is ideal for downloading data in plain text format
  - let's download some human genome data for chr1:1-10,000,000
    - golden path assembly
    - BAC end sequence alignments



Home Genomes Blat Tables Gene Sorter PCR FAQ Help

Table Browser

Use this program to get the data associated with a track in text format, to calculate intersections between tracks, and to retrieve DNA sequence covered by a track. See Using the Table Browser for a description of the controls in this form. The old Table Browser Page is still available for a limited period.

clade: Vertebrate **genome:** Human **assembly:** May 2004
group: Mapping and Sequencing Tracks **track:** Assembly
table: gold describe table schema
region: genome ENCODE position chr1:1-10000000 lookup
identifiers (names/accessions): paste list upload list
filter: create
intersection: create
output format: all fields from selected table
output file: hg17_agp.txt (leave blank to keep output in browser)
file type returned: plain text gzip compressed

get output summary/statistics

To reset **all** user cart settings (including custom tracks), click here.

Home Genomes Blat Tables Gene Sorter PCR FAQ Help

Table Browser

Use this program to get the data associated with a track in text format, to calculate intersections between tracks, and to retrieve DNA sequence covered by a track. See Using the Table Browser for a description of the controls in this form. The old Table Browser Page is still available for a limited period.

clade: Vertebrate **genome:** Human **assembly:** May 2004
group: Mapping and Sequencing Tracks **track:** BAC End Pairs
table: bacEndPairs describe table schema
region: genome ENCODE position chr1:1-10000000 lookup
identifiers (names/accessions): paste list upload list
filter: create
intersection: create
output format: all fields from selected table
output file: hg17_bes.txt (leave blank to keep output in browser)
file type returned: plain text gzip compressed

get output summary/statistics

To reset **all** user cart settings (including custom tracks), click here.

Exploring the Files

- use **head** and **wc** to examine structure of files
  - downloaded **hg17_agp.txt** and **hg17_bes.txt**
    - hg17_agp.txt is tab-delimited with a header line, 104 lines

idioms

head FILE
first 10 lines in a file

head –NUM FILE
first NUM lines in a file

wc –l FILE
the number of lines in a file

```
» ls
-rw-r--r--    1 martink  users        5535 2005-04-25 13:19 hg17_agp.txt
-rw-r--r--    1 martink  users       38624 2005-04-25 13:19 hg17_bes.txt

» head hg17_agp.txt
#bin  chrom chromStart  chromEnd        ix          type    frag      fragStart fragEnd strand
585   chr1  0     616   1     F          AP006221.1  36115   36731     -
73    chr1  616   167280      2          F           AL627309.15       240         166904      +
586   chr1  217280257582      4          F           AP006222.1        0           40302       +
73    chr1  357582511231      7          F           AL732372.15       0           153649      +
73    chr1  561231672780      9          F           AC114498.2        0           111549      +
73    chr1  672780852347      10         F           AL669831.13       0           179567      +
73    chr1  8523471038212     11         F           AL645608.29       2000        187865      +
9     chr1  1038212     1167191          12          F       AL390719.47           2000        130979      +
74    chr1  1167191     1277350          13          F       AL162741.44           2000        112159      +

» wc -l hg17_agp.txt
    104 hg17_agp.txt
```

Exploring Line Fields

- converting tabs to spaces – use **expand**
  - **expand –t NUM** will replace each tab with NUM spaces

```
» expand -t 1 hg17_agp.txt | head
#bin chrom chromStart chromEnd ix type frag fragStart fragEnd strand
585 chr1 0 616 1 F AP006221.1 36115 36731 -
73 chr1 616 167280 2 F AL627309.15 240 166904 +
586 chr1 217280 257582 4 F AP006222.1 0 40302 +
73 chr1 357582 511231 7 F AL732372.15 0 153649 +
73 chr1 561231 672780 9 F AC114498.2 0 111549 +
73 chr1 672780 852347 10 F AL669831.13 0 179567 +
73 chr1 852347 1038212 11 F AL645608.29 2000 187865 +
9 chr1 1038212 1167191 12 F AL390719.47 2000 130979 +
74 chr1 1167191 1277350 13 F AL162741.44 2000 112159 +
```

- show the second line

```
» expand -t 1 hg17_agp.txt | head -2 | tail -1
585 chr1 0 616 1 F AP006221.1 36115 36731 -
```

idioms

expand –t NUM FILE
replace each tab with NUM spaces

tail FILE
last 10 lines

tail –NUM FILE
last NUM lines

head –NUM FILE | tail -1
NUMth line

idioms

tr CHR1 CHR2

replace each instance of character CHR1 with character CHR2 (transliterate)

Exploring Line Fields

- it is easier to explore a single line when the each field is reported on a different line
  - replace spaces (or the file's delimiter) with a newline (\n)

```
» expand -t 1 hg17_agp.txt | head -1 | tr " " "\n"
#bin
chrom
chromStart
chromEnd
ix
type
frag
fragStart
fragEnd
strand
```

headers

```
» expand -t 1 hg17_agp.txt | head -2 | tail -1 | tr " " "\n"
585
chr1
0
616
1
F
AP006221.1
36115
36731
-
```

first data line

```
» head -2 hg17_agp.txt | tail -1 | tr “\t" "\n"
585
chr1
0
616
1
F
AP006221.1
36115
36731
-
```

last data line

Exploring Line Fields

- let's number the fields

```
» head -2 hg17_agp.txt | tail -1 | tr "\t" "\n" | cat –n
     1   585
     2   chr1
     3   0
     4   616
     5   1
     6   F
     7   AP006221.1
     8   36115
     9   36731
    10   -
```

idioms

cat –n FILE
prefix each line by the line's number

sed 's/REGEX/STRING/'
replace first match of REGEX with STRING

sed 's/^ *//'
remove leading spaces

sed 's/ *$//'
remove trailing spaces

- some utilities (e.g. uniq) indent the first field for clarity
  - this may break your parsing, if you're not expecting it
    - use **sed** to remove leading spaces
  - TAB is the typical output delimiter
    - use **expand** or **tr** to get rid of newly introduced tabs

```
» head -2 hg17_agp.txt | tail -1 | tr "\t" "\n" | cat –n
  | sed 's/^ *//'
1   585
2   chr1
. . .
```

Exploring Line Fields

- let's use this recipe for the second file
  - glean file's format
    - e.g. clone's name is in the 5th column

```
» head -2 hg17_bes.txt | tail -1 | tr "\t" "\n" | cat -n
  | sed 's/^ *//' | expand -t 1
1 585
2 chr1
3 5875
4 129658
5 CTD-3214E10
6 1000
7 -
8 all_bacends
9 2
10 5875,129237
11 509,421
12 AQ805270,AQ889555
```

idioms

head -2
first 2 lines

tail -1
last line

tr "\t" "\n"
replace all tabs with new lines

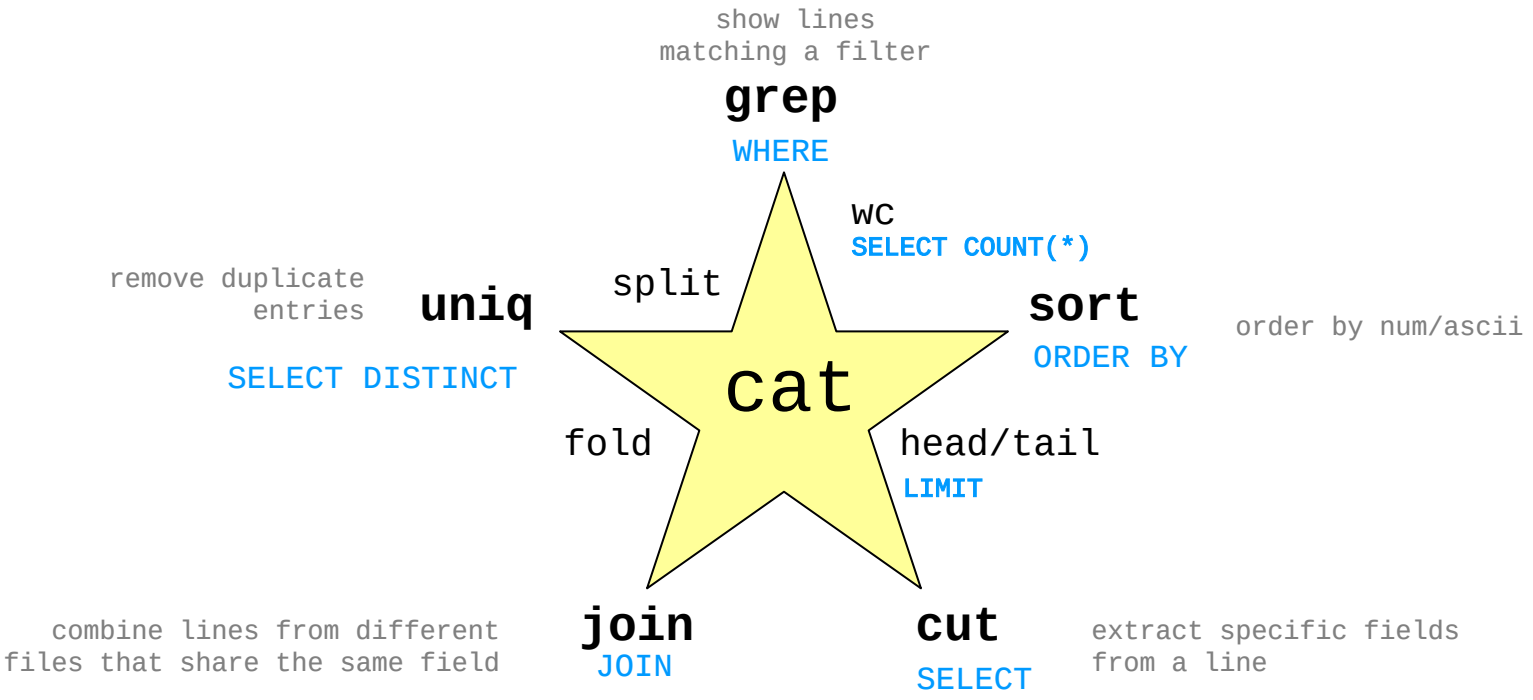
cat –n
prefix lines with their number

sed 's/^ *//'
remove leading spaces

expand –t 1
replace tabs by one space

Complex Recipe From a Few Simple Transformations

- basic command-line utilities effect a primitive transformation
  - most have SQL equivalents
- think of what you need to do in terms of these "atomic" steps



Fun with tr

idioms

tr –d CHR1
delete instances of CHR1

fold –w NUM
split a line into multiple lines every NUM characters

- visualize sequences with tr and sed
- reformat a FASTA file to 120 lines to fill the screen

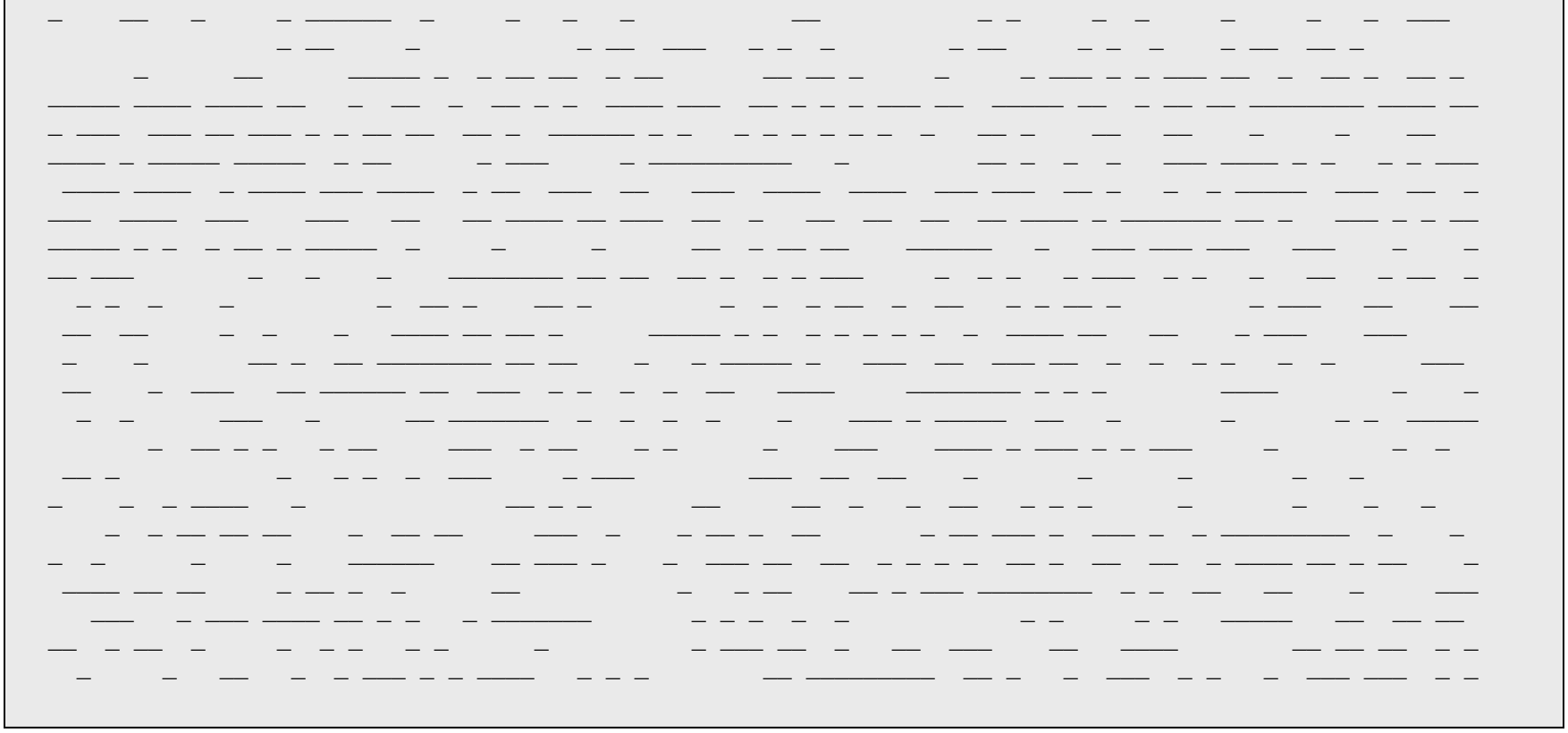
```
» head ~/work/fly/fasta/bac/BACR06L13.release4
Contig15 ./D744.fasta.screen.ace.10 from 2974 to 166304
GAATTCGTAACATTTTCTGGGGCGTACTAAAAGTTACTTTCAAAAATATT
ATGCATATATTTATTGTCTTTATGTTCATTAAGATTTACATTCATGGCAT
TTAAATATAATAAATACAGCATTAAGAATTTTTAAAAGTGCTTGCCAATG
```

```
grep -v ^Contig ~/work/fly/fasta/bac/BACR06L13.release4 | tr -d "\n" | fold -w 120 > tmp.1
head -2 tmp.1
GAATTCGTAACATTTTCTGGGGCGTACTAAAAGTTACTTTCAAAAATATTATGCATATATTTATTGTCTTTATGTTCATTAAGATTTACATTCATGGCAT
TTAAATATAATAAATACAGCATTAAGAATTTTTAAAAGTGCTTGCCAATGTGTTCTTAATTAACTCCTAAATCTCTACTATTCACCATGCTCTTAAATTA
```

- let's replace some base pairs with **tr** and see what happens

Show GC Content as Islands

```
tr ATGC "  __" tmp.1
```



idioms

grep ^CHR

report lines that start with character CHR (^ is the start-of-line anchor)

Isolate GC Islands

- let's report each GC island on its own line
  - replace all spaces by newlines
  - report only lines that start with an underscore (i.e. are a GC island)

```
tr " " "\n“ tmp.1 | grep ^_ | head
__
_
_
_____
_
_
_
_
__
```

- how many islands are there?

```
tr " " "\n“ tmp.1 | grep ^_ | wc -l
  37513
```

Count Islands by Size

- to count identical lines use **uniq –c**
  - lines must be pre-sorted, since uniq –c reports runs of duplicates

```
tr " " "\n“ tmp.1 | grep ^_ | sort | uniq -c
  20577      _
   9503      __
   4096      ___
   1789      ____
    821      _____
    356      ______
    175      _______
    102      ________
     45      _________
     26      __________
     12      ___________
      3      ____________
      4      _____________
      2      ______________
      1      _______________
      1      ________________
```

idioms

uniq –c FILE

report number of adjacent duplicate lines

sort FILE | uniq –c FILE

report number of duplicate lines in a file, regardless of their position

EXAMPLE

>cat nums.txt
11112232333

>fold –w 1 nums.txt | uniq –c
4 1
2 2
1 3
1 2
3 3
>fold –w 1 nums.txt | sort | uniq –c
4 1
3 2
4 3

Count Island by Size

- to get the size of each island, we want the length of the line
  - awk comes in handy here – replace each line by its length
  - -n flag asks sort for numerical sorting

```
tr " " "\n“ tmp.1 | grep ^_ | awk ‘{print length($0)}’ | sort –n | uniq –c
  20577     1
   9503     2
   4096     3
   1789     4
    821     5
    356     6
    175     7
    102     8
     45     9
     26     10
     12     11
      3     12
      4     13
      2     14
      1     15
      1     16
```

idioms

sort -n

sort lines numerically by the first column

awk `{ print length($0) }`

print the length of each line

sort –n | uniq –c -vs- sort | uniq -c

idioms

sort [-n] +NUM
sort lines by the NUM column (0-indexed)

```
tr " " "\n“ tmp.1 | grep ^_ | awk ‘{print length($0)}’ |
sort –n | uniq –c
  20577     1
   9503     2
   4096     3
   1789     4
    821     5
    356     6
    175     7
    102     8
     45     9
     26     10
     12     11
      3     12
      4     13
      2     14
      1     15
      1     16
```

```
tr " " "\n“ tmp.1 | grep ^_ | awk ‘{print length($0)}’ |
sort | uniq –c
  20577     1
     26     10
     12     11
      3     12
      4     13
      2     14
      1     15
      1     16
   9503     2
   4096     3
   1789     4
    821     5
    356     6
    175     7
    102     8
     45     9
```

```
tr " " "\n“ tmp.1 | grep ^_ | awk ‘{print length($0)}’ |
sort | uniq –c | sort –n +1
  20577     1
   9503     2
   4096     3
   1789     4
    821     5
    356     6
    175     7
    102     8
     45     9
     26     10
     12     11
      3     12
      4     13
      2     14
      1     15
      1     16
```

Counting Frequencies

- what are the most common triplets (e.g. AAA, AAC, AAT, etc) in a given sequence?
  - create triplets – non-overlapping
  - sort triplets
  - count duplicated triplets
  - sort by frequency of occurrence
  - report top 5

idioms

tr –d "\n" FILE | fold –w NUM
report all characters in a file, NUM characters at a time

sort [-n] -r FILE
sort in descending order

```
grep -v ^Contig ~/work/fly/fasta/bac/BACR06L13.release4 | tr -d "\n" | fold -w 3
GAA
TTC
GTA
ACA
TTT
TCT
```

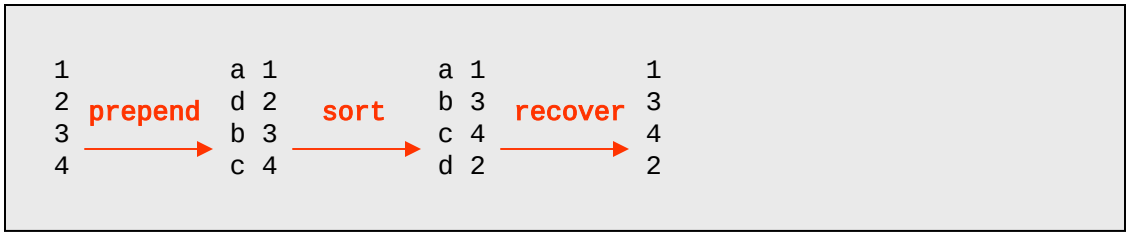
```
grep -v ^Contig ~/work/fly/fasta/bac/BACR06L13.release4 | tr -d "\n" | fold -w 3 |
sort | uniq -c | head -5
   1959     AAA
    900     AAC
    942     AAG
   1489     AAT
    961     ACA
```

```
grep -v ^Contig ~/work/fly/fasta/bac/BACR06L13.release4 | tr -d "\n" | fold -w 3 |
sort | uniq -c | sort -nr | head -5
   2088     TTT
   1959     AAA
   1561     ATT
   1489     AAT
   1341     TAA
```

Schwartzian Transform – at the command line

- the ST is a Perl idiom used to sort elements of an array based on the result of a function applied to each element
  - start with array [1,2,3]
  - create a new array that is a list of arrays containing both
    - original elements, and
    - argument to sort created by applying some function to the original elements
    - [[a,1], [c,2], [b,3]]
  - apply sort to the new element; here acb->abc to give [[a,1], [b,3], [c,2]]
  - recover elements from original array [1,3,2]

- this idiom can be used at the command line
  - prepend each line with result of some function applied to the line
  - sort by the result
  - recover the line



Counting Frequencies – cont'd

- we found the most frequent triplets
- how about 6-mers sorted by the number of Gs in them?
  - we want to apply the function "number_of_G(string)" to the second field of each line and sort by the result
- first, let's get all the 6-mers and their frequencies

```
grep -v ^Contig ~/work/fly/fasta/bac/BACR06L13.release4 | tr -d "\n" | fold -w 6 |
sort | uniq -c | head
     47     AAAAAA
     19     AAAAAC
     16     AAAAAG
     45     AAAAAT
     23     AAAACA
     21     AAAACC
      8     AAAACG
     24     AAAACT
      8     AAAAGA
     11     AAAAGC
```

Counting Frequencies – cont'd

- make a new line, with a copy of the 2nd field

```
grep -v ^Contig ~/work/fly/fasta/bac/BACR06L13.release4 | tr -d "\n" | fold -w 6 |
sort | uniq -c | awk '{print $2,$0}'
AAAAAA      47    AAAAAA
AAAAAC      19    AAAAAC
AAAAAG      16    AAAAAG
. . .
```

- transform the first field into the number of Gs in that field

```
grep -v ^Contig ~/work/fly/fasta/bac/BACR06L13.release4 | tr -d "\n" | fold -w 6 |
sort | uniq -c | awk '{print $2,$0}' |
awk ' { gsub(/[^G]/,"",$1) ; print length($1),$2,$3} ' | head
0 47 AAAAAA
0 19 AAAAAC
1 16 AAAAAG
0 45 AAAAAT
0 23 AAAACA
0 21 AAAACC
1 8 AAAACG
0 24 AAAACT
1 8 AAAAGA
1 11 AAAAGC
```

Counting Frequencies – cont'd

- sort by the first and second fields

```
1  grep -v ^Contig ~/work/fly/fasta/bac/BACR06L13.release4 | tr -d "\n" |
2  fold -w 6 | sort | uniq -c |
3  awk '{print $2,$0}' |
   awk '{gsub(/[^G]/,"",$1);print length($1),$2,$3}' |
   sort -nr +0 +1 | head -10
   5 8 GGGGTG
   5 7 GGGTGG
   5 7 GGGGCG
   5 6 GGTGGG
   5 6 GGGGGT
   5 6 GGCGGG
   5 6 GCGGGG
   5 5 GGGGAG
   5 5 GGGAGG
   5 4 GTGGGG
```

number of Gs

frequency of 6-mer

idioms

sort +NUM1 +NUM2

sort lines in a file first by field NUM1 then by NUM2

RECIPE

1 create a copy of the relevant field(s)
2 apply a function to the field(s)
3 sort by the result of the function

Listing Cluster Jobs - qstat

- process output of qstat (on oscar) to stay on top of your jobs

```
>qstat

job-ID  prior name       user         state submit/start at     queue      master  ja-task-ID
------------------------------------------------------------------------------------------
2240714     0 runBlast.s ahe          r     04/25/2005 20:08:43 o0001.q    SLAVE
2240714     0 runBlast.s ahe          r     04/25/2005 20:08:43 o0002.q    SLAVE
2240714     0 runBlast.s ahe          r     04/25/2005 20:08:43 o0003.q    SLAVE
2240714     0 runBlast.s ahe          r     04/25/2005 20:08:43 o0004.q    SLAVE
2240714     0 runBlast.s ahe          r     04/25/2005 20:08:43 o0005.q    SLAVE
2240714     0 runBlast.s ahe          r     04/25/2005 20:08:43 o0006.q    SLAVE
2240714     0 runBlast.s ahe          r     04/25/2005 20:08:43 o0007.q    SLAVE
2240714     0 runBlast.s ahe          r     04/25/2005 20:08:43 o0008.q    SLAVE
2240714     0 runBlast.s ahe          r     04/25/2005 20:08:43 o0009.q    SLAVE
2240714     0 runBlast.s ahe          r     04/25/2005 20:08:43 o0010.q    SLAVE
2240714     0 runBlast.s ahe          r     04/25/2005 20:08:43 o0011.q    SLAVE
2240765     0 WBDMwgs3x. rwarren      r     04/26/2005 09:34:53 o0011.q    MASTER
2240714     0 runBlast.s ahe          r     04/25/2005 20:08:43 o0012.q    SLAVE
2240782     0 WBDMwgs3x. rwarren      r     04/26/2005 09:34:53 o0012.q    MASTER
. . .
```

r=running
qw=queued

Listing Cluster Jobs - qstat

- what are the fields in each column?
  - take the first dataline and prefix each column field with its index

```
qstat | grep ^[0-9] | head -1 | tr -s " " "\n" | cat -n
     1  2240714
     2  0
     3  runBlast.s
     4  ahe
     5  r
     6  04/25/2005
     7  20:08:43
     8  o0001.q
     9  SLAVE
```

- user name is in column 4 – apply **sort** and **uniq** to it to list jobs per user

```
qstat | grep ^[0-9] | tr -s " " | cut -d " " -f 4 | sort | uniq -c
     98 ahe
      1 martink
     22 mbilenky
     18 rwarren
```

Listing Cluster Queue Details – qstat -f

```
>qstat -f

queuename            qtype used/tot. load_avg arch       states
---------------------------------------------------------------------------
o0001.q              BIP   1/2       0.01     glinux
2240714     0 runBlast.s ahe          r     04/25/2005 20:08:43 SLAVE
---------------------------------------------------------------------------
o0002.q              BIP   1/2       2.00     glinux
2240714     0 runBlast.s ahe          r     04/25/2005 20:08:43 SLAVE
---------------------------------------------------------------------------
o0003.q              BIP   1/2       2.00     glinux
2240714     0 runBlast.s ahe          r     04/25/2005 20:08:43 SLAVE
---------------------------------------------------------------------------
o0004.q              BIP   1/2       0.00     glinux
2240714     0 runBlast.s ahe          r     04/25/2005 20:08:43 SLAVE
---------------------------------------------------------------------------
o0005.q              BIP   1/2       2.00     glinux
2240714     0 runBlast.s ahe          r     04/25/2005 20:08:43 SLAVE
---------------------------------------------------------------------------
o0006.q              BIP   1/2       0.00     glinux
2240714     0 runBlast.s ahe          r     04/25/2005 20:08:43 SLAVE
---------------------------------------------------------------------------
o0007.q              BIP   1/2       0.00     glinux
2240714     0 runBlast.s ahe          r     04/25/2005 20:08:43 SLAVE
```

when parsing output in which records span multiple lines, try to identify some unique feature of each part of the record that will extract a given line

queue lines have a ".q" in them – use **grep "\.q"** to extract these

job lines have a ":" in the time – use **grep :** to extract these

Counting free/busy CPUs

- each machine appears on its own line
  - M/N, M=used CPU, N=total CPU
  - load (e.g. 0.73)

```
qstat -f | grep "\.q"
o0001.q              BIP   1/2       0.00     glinux
o0002.q              BIP   1/2       2.00     glinux
o0003.q              BIP   1/2       2.00     glinux
o0004.q              BIP   1/2       0.00     glinux
o0005.q              BIP   1/2       2.00     glinux
o0006.q              BIP   1/2       0.73     glinux
o0007.q              BIP   1/2       0.00     glinux
o0008.q              BIP   1/2       2.86     glinux
. . .
```

- let's extract the 3rd field and remove the "/"

```
qstat -f | grep "\.q" | tr -s " " | cut -d " " -f 3 | tr "/" " "
1 2
1 2
2 2
2 2
. . .
```

used CPUs

CPUs

collapse multiple adjacent spaces

extract 3rd field

replace / with a space

```
qstat -f | grep "\.q" | tr -s " " | cut -d " " -f 3 | tr "/" " " | sort | uniq -c
     80     0 2
     79     1 2
     30     2 2
```

report frequencies of unique M/N combinations

Start awking!

```
qstat -f | grep "\.q" | tr -s " " | cut -d " " -f 3 | tr "/" " " |
sort | uniq -c
     80     0 2
     79     1 2
     30     2 2

qstat -f | grep "\.q" | tr -s " " | cut -d " " -f 3 | tr "/" " " | sort | uniq -c |
awk '{print $1*$2,$1*$3}'
0 160
79 158
60 60

qstat -f | grep "\.q" | tr -s " " | cut -d " " -f 3 | tr "/" " " | sort | uniq -c | awk '{print $1*$2,$1*$3}' |
sums
139 378
```

- we find 139/378 CPUs are used
- sums is part of the Perl prompt tools
  - set of scripts that reduce the drudge work of manipulating lines and fields at the prompt
  - we'll see those in a few lectures

Today's Idioms

idioms

head FILE
first 10 lines in a file

tail FILE
last 10 lines in a file

head –NUM FILE
first NUM lines in a file

tail –NUM FILE
last NUM lines in a file

head –NUM FILE | tail -1
NUMth line

wc –l FILE
number of lines in a file

idioms

sort FILE
sort lines asciibetically by first column

sort +COL FILE
sort lines asciibetically by COL column

sort –n FILE
sort lines numerically in ascending order

sort –nr FILE
sort lines numerically in descending order

sort +NUM1 +NUM2
sort lines in a file first by field COL1 then COL2

idioms

cat –n FILE
prefix lines with their number

tr CHR1 CHR2 FILE
replace all instances of CHR1 with CHR2

tr ABCD 1234 FILE
replace A->1, B->2, C->3, D->4

tr –d CHR1
delete instances of CHR1

fold –w NUM
split a line into multiple lines every NUM characters

expand –t NUM FILE
replace each tab with NUM spaces

idioms

grep ^CHR FILE
report lines that start with character CHR (^ is the start-of-line anchor)

grep –v ^CHR FILE
lines that don't start with CHR

sed 's/REGEX/STRING/'
replace first match of REGEX with STRING

sed 's/^ *//'
remove leading spaces

uniq –c FILE
report number of adjacent duplicate lines

2.1.2.4.1

Command-Line Data Analysis and Reporting – Session 1

- read man pages for tools covered today
  - "man tr"
- become proficient at the command line is like learning a very tiny language with very simple grammar
- see you next time!