UNIVERSITY OF GEORGIA
College of Agricultural & Environmental Sciences

Unix commands for data editing

Ignacio Aguilar
INIA Uruguay

Daniela Lourenco
UGA USA

UGA TEAM, 08/2019

Other popular commands

`head file`	prints first 10 lines
`head -20 file`	prints first 20 lines
`tail file`	prints last 10 lines
`less file`	lists *file* line-by-line or page-by-page
`wc -l file`	counts the number of lines
`grep text file`	finds lines that contains text
`cat file1 file2`	concatenates files
`sort`	sorts a file
`cut`	cuts specific columns
`join`	joins lines of two files on specific columns
`paste`	pastes lines of two files
`expand`	replaces TAB with spaces
`uniq`	retains unique lines on a sorted file

head / tail

```
$ head pedigree.txt
1 0 0
2 0 0
3 0 0
4 0 0
5 0 0
6 0 0
7 0 0
8 0 0
9 0 0
10 0 0
```

Huge volume of information

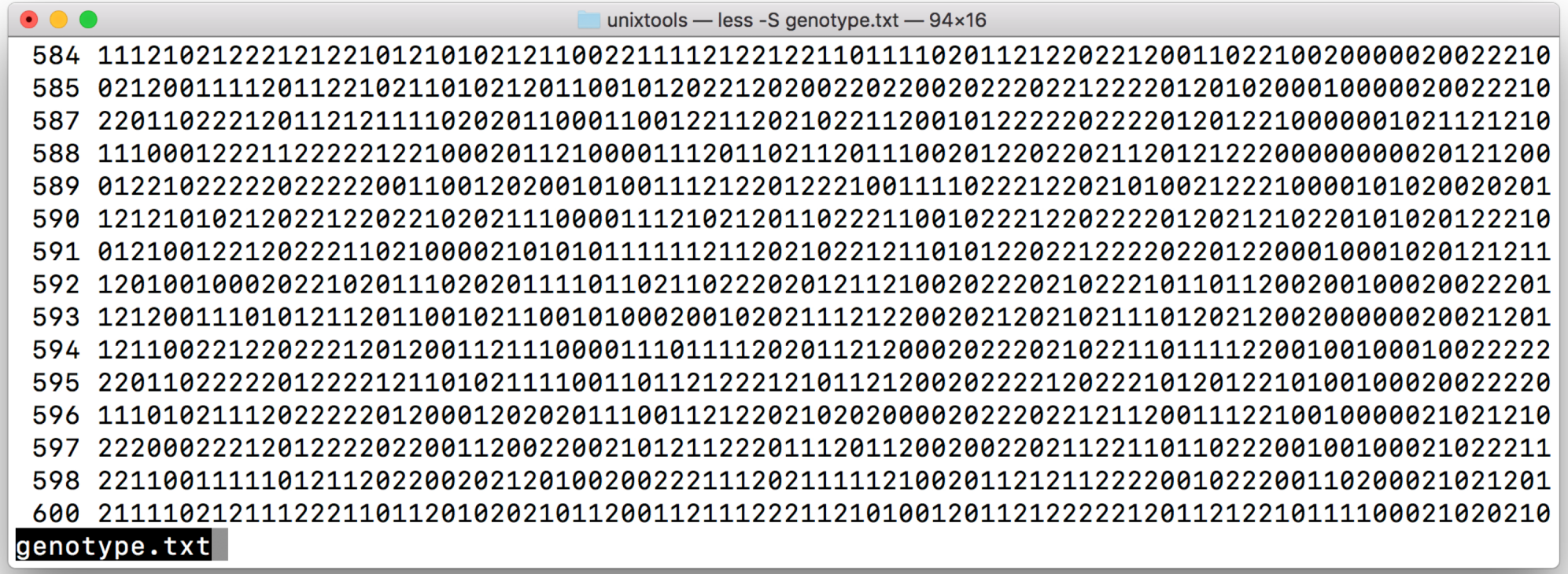
- Example 50kv2 (54609 SNP)
  - For 104 individuals
  - Illumina final report file:
    - 5,679,346 records
    - 302 MB

- Not efficient to read/edit with regular editors (vi, vim, gedit…)

less command

- Allows to view the content of file and move forward and backward
- For files with long lines use option `-S`

```
less -S genot.txt
```



Counting lines/characters inside files

- Command **`wc`** counts the number of words/lines/characters/bytes

- Number of lines of a file(s)

```
$ wc -l genotype.txt pedigree.txt
    3534 genotype.txt
    6473 pedigree.txt
```

Concatenating files

Put content of file1 and file2 in output_file

```
cat file1 file2  > output_file
```

```
==> file1 <==
1
2
3

==> file2 <==
a
b
c

==> output_file <==
1
2
3
a
b
c
```

Add content of file3 to output_file using **>>** redirection

Append content at the end of the file

```
cat file3 >> output_file
```

```
==> file3 <==
x
y
z

==> output_file <==
1
2
3
a
b
c
x
y
z
```

paste / expand

`paste` merges files line by line with a TAB delimiter

`expand` replaces TAB with spaces

`Paste -d " "` merges files line by line with a space delimiter

```
head sol_a sol_b
```

```
==> sol_a <==
trait/effect level  solution
   1   1         1        10.23612694
   1   1         2        12.09158350
   1   1         3        13.08139319
   1   2         1        -0.00804515
   1   2         2         0.00804485
   1   2         3        -0.01608983
   1   2         4         0.01609003
   1   2         5         0.03218022
   1   2         6        -0.03218023

==> sol_b <==
trait/effect level  solution
   1   1         1        14.09761235
   1   1         2        15.95388962
   1   1         3        16.94297738
   1   2         1        -0.00806027
   1   2         2         0.00805982
   1   2         3        -0.01608645
   1   2         4         0.01608642
   1   2         5         0.03217834
   1   2         6        -0.03217873
```

```
paste sol_a sol_b | expand | head
```

```
trait/effect level  solution    trait/effect level  solution
   1   1         1        10.23612694     1   1         1         14.09761235
   1   1         2        12.09158350     1   1         2         15.95388962
   1   1         3        13.08139319     1   1         3         16.94297738
   1   2         1        -0.00804515     1   2         1         -0.00806027
   1   2         2         0.00804485     1   2         2          0.00805982
   1   2         3        -0.01608983     1   2         3         -0.01608645
   1   2         4         0.01609003     1   2         4          0.01608642
   1   2         5         0.03218022     1   2         5          0.03217834
   1   2         6        -0.03218023     1   2         6         -0.03217873
```

sort / join

- Sorts a file in alphanumeric order
  - specifying which column should be sorted

```
sort -k 2,2 file1  > a  or sort +1 -2 file1 > a
sort -k 1,1 file2  > b  or sort +0 -1 file2 > b
```

- Sorts a file in numeric order

```
sort -nk 2,2 file1  > a  or sort -n +1 -2 file1 > a
sort -nk 1,1 file2  > b  or sort -n +0 -1 file2 > b
```

- Merges both files by column 2 and column 1

```
join -1 2 -2 1 a b > new_file
```

grep

- grep finds patterns within a file and lists all lines that match the pattern

```
grep UGA42014 pedigree.txt
```

```
UGA42014 UGA41101 UGA37367
```

- grep -v shows all lines that do not match the pattern

```
grep -v UGA42014 pedigree.txt
```

```
UGA42011 UGA41101 UGA34199
UGA42012 UGA41101 UGA38407
UGA42013 UGA41101 UGA39798
UGA42015 UGA41101 UGA40507
```

- Pattern with spaces use -e

```
grep -e “pattern with spaces” file1
```

sed

- Sed is a stream editor

  It reads input file and apply command that match pattern

- Substitution of a pattern

```
sed ‘s/pattern 1/new pattern/g’ file > newfile
```

- Substitution of a pattern in the same file

```
sed -i ‘s/pattern 1/new pattern/g’ file
```

- Substitution of a pattern in a specific line (e.g., line 24)

```
sed ‘24s/pattern 1/new pattern/’ file > newfile
```

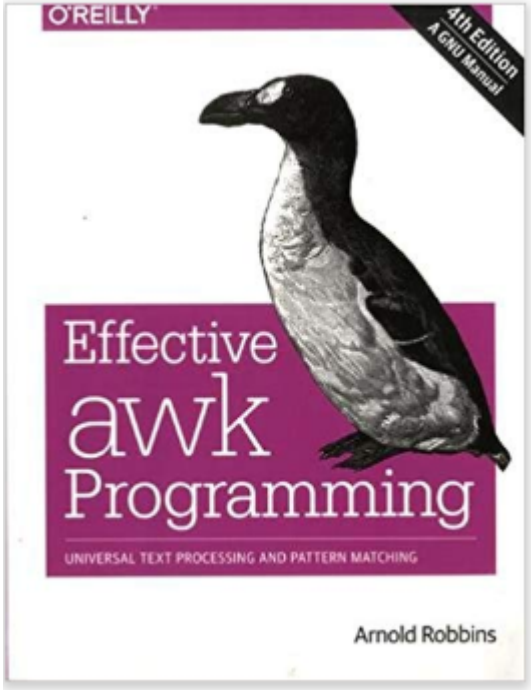
- Deletes lines that contain “pattern to match”

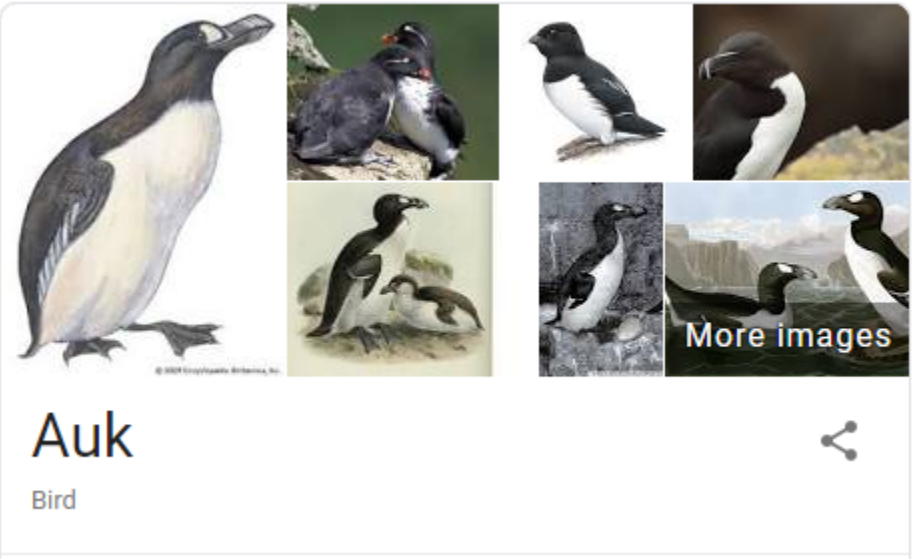
```
sed '/pattern to match/d' file
```

awk

AWK is a language for text processing and typically used as a data extraction and reporting tool

Alfred **A**ho
Peter **W**einberger
Brian **K**ernighan





awk

- Interpreted program language, that process data stream of a file line by line
- Very useful and fast command to work with text files
- Can be used as a database query program
  - Selects specific columns or create new ones
  - Selects specific rows matching some criteria
- Can be used with **if/else** and **for** structures

awk

Implicit variables
NF - number of fields
NR - record number
FS - input field separator
OFS - output field separator

- Print columns 1, 2, and last of solutions file

```
awk '{print $1,$2,$NF}' solutions > sol_out
```

- Print all the columns:

```
awk '{print $0}' solutions > sol_out
```

- Print columns 3 and 4 based on occurrence in column 2:

```
awk '{if ($2==2) print $3,$4}' solutions > sol_out
```

- Print columns 3 and 4 skipping the first line:

```
awk '{if (NR>1) print $3,$4}' solutions > sol_out
```

- Print length of column 2 from line 1:

```
awk '{if (NR==1) print length($2)}' snp.txt
```

- Process CSV files

```
awk 'BEGIN {FS=","} {print $1,$2,$3}' pedigree.txt > ped_out
```

awk hash tables

- Arrays can be indexed by alphanumeric variables in a efficient way

- awk version to count progeny by sire
  - sire id is column 2

```
awk '{ sire[$2]+=1} END { for (s in sire)
      {print "Sire " s, sire[s]}}' pedigree.txt
```

```
Sire UGA45217 400
Sire UGA43767 400
Sire UGA38476 200
Sire UGA41101 400
Sire UGA48548 200
Sire UGA45825 400
Sire UGA44642 400
Sire UGA45179 400
```

awk

- awk can be used for pretty much anything related to data processing in Unix

- Sum of elements in column 1

```
awk '{ sumf += $1 } END { print sumf}' data.txt
```

```
45
```

```
1
2
3
4
5
6
7
8
9
```

- Sum of squares of element in column 1

```
awk '{ sumf += $1*$1 } END { print sumf}' data.txt
```

```
285
```

- Average of element in column 1

```
awk '{ sumf += $1 } END { print sumf/NR}' data.txt
```

```
5
```

uniq

- Command **uniq** lists all unique lines of a file
- Option –c counts number of times each level occurs in a file

Example: counting progeny by sire in a pedigree file

```
awk '$2>0{ print $2}' ped  | sort | uniq -c > s.temp

awk '{ if ($2>0) print $2}' ped  | sort | uniq -c >
s.temp
```

Run in background + Save output

```
$vi blup.sh
#type the following commands inside ai.sh
      #!/bin/bash
      blupf90 <<AA > blup.log
      renf90.par
      AA
#save and exit
$bash ai.sh &    #can replace bash by sh
```

```
$vi gibbs.sh
#type the following commands inside ai.sh
      #!/bin/bash
      gibbs2f90 <<AA > gibbs.log
      renf90.par
      1000 0
      10
      AA
#save and exit
$bash gibbs.sh &  #can replace bash by sh
```

Useful commands for Linux

- Several tutorials on the WEB !!

- unixcombined.pdf from Misztal web site
  - http://nce.ads.uga.edu/~ignacy/ads8200/unixcombined.pdf

- genomeek blog (F. Guillaume)
  - http://genomeek.wordpress.com