Unix data manipulation

Jon Chernus (adapted from Ryan Minster)

Department of Human Genetics
School of Public Health
University of Pittsburgh

Document created: September 30, 2024

Location

This slide set is called `unix_data_manipulation` and is located in the “19_unix_data_manipulation” folder of our Lectures repository.

Objectives

- To learn Unix tools like sed and awk that can be used to manipulate data

Review: things you already know

Some commands for interacting with text/files:

Command	Purpose	Useful options
`echo`	Printing stuff to stdout	
`cat`	Printing files to stdout	
`less` and `more`	Scrolling through files	
`head` and `tail`	Looking at beginning/end of files	● `head -nN` - print first N lines ● `tail -nN` - print last N lines ● `tail -n+N` - Print starting at line N
`wc`	Get line, word, and byte count of files	● `wc -l` Print number of lines only

What you know about grep

`grep pattern file` prints all lines of `file` matching `pattern`.

- `grep -v pattern file` - invert selection (get lines without matches)
- `grep -i pattern file` - ignore case of pattern/match

More grep options

More options for how matching is done and how results are displayed:

Option	Meaning	Example command	Example output
`-e`	Use to search for multiple patterns	`grep -e pattern1 -e pattern2 file`	find lines matching `pattern1` or `pattern2`
`-w`	Only find whole-word matches	`grep -w rs123`	matches `rs123` but not `rs1234`
`-o`	Only print the matching part (not the whole line)	`echo rs1234 | grep rs123`	prints only `rs123`
`-n`	Also show line numbers of matches	`echo rs1234 | grep -n rs123`	prints `1:rs1234`
`-c`	Only print the number of matching lines	`echo rs1234 | grep -c rs123`	prints 1

More grep options: context

Here are some options for printing matches in context by showing surrounding lines

Option	Meaning
`-B N`	Also print `N` lines before matches
`-A N`	Also print `N` lines after matches
`-C N`	Also print `N` lines before and after matches

grep “context” examples

```
# The whole file
cat data/snp_list.txt
rs100
rs101
rs102
rs103
rs104
rs105
rs106
rs107
rs108
rs109

# Match for rs105 and preceding 1 line
grep -B 1 "rs105" data/snp_list.txt
rs104
rs105

# Match for rs105 and following 2 lines
grep -A 2 "rs105" data/snp_list.txt
rs105
rs106
rs107
```

Using `grep` with multiple files

You can also search multiple files and modify the behavior of `grep`.

- `grep rs123 file file2` searches for the pattern `rs123` in both files
- or you could use `grep rs123 file*`

Option	Meaning
`-r`	search directories recursively
`-l`	print only the names of files containing matches

grep with multiple files examples

```
# The two files
head -n3 data/snp_list*.txt
==> data/snp_list.txt <==
rs100
rs101
rs102

==> data/snp_list2.txt <==
rs200
rs201
rs202


# Which of them contain a 5 in them? (both)
grep -l "5" data/snp_list*.txt
data/snp_list.txt
data/snp_list2.txt
```

regex

expression	what it matches
`.`	any single character (except newline, `\n`)
`^`	start of line
`$`	end of line
`\b`	word boundary
`\`	\ is an escape character - put it in front of a special character like `.` to search for it
`[...]`	any character in the brackets

More sources:

- A cheat sheet for regex
  http://web.mit.edu/hackl/www/lab/turkshop/slides/regex-cheatsheet.pdf
- Additional info
  https://www.regular-expressions.info
- Regex crossword puzzles
  https://regexcrossword.com/

More grep examples

```
# The whole file
cat data/example.txt
bio
bioinfo
bioinformatics
computational biology

# Matches for info
grep --color info data/example.txt
bioinfo
bioinformatics

# Non-matches for inf
grep --color -v info data/example.txt
bio
computational biology

# Whole-word matches for bio
grep --color -w bio data/example.txt
bio

# Matches for u or for tics
grep --color -e u -e tics data/example.txt
bioinformatics
computational biology
```

More grep examples (2)

```
# First line before a match
grep -B1 --color comp data/example.txt
bioinformatics
computational biology

# First line after a match
grep -A1 --color tics data/example.txt
bioinformatics
computational biology

# Show 2 lines around each match
grep -C2 --color 68 data/chroms.txt
chrom1  3214482 3216968
chrom1  3216025 3216968
chrom1  3216022 3216024
chrom1  3671349 3671498
--
chrom1  3466587 3513553
chrom1  3466587 3513553
chrom1  3466587 3466687
chrom1  3513405 3513553
chrom1  3783876 3783933
```

More grep examples (3)

```
# The whole file:
cat data/example.txt
bio
bioinfo
bioinformatics
computational biology

# Count the number of lines without bio
grep -v -c bio data/example.txt
0

# Show line number of lines matching info
grep -n info data/example.txt
2:bioinfo
3:bioinformatics

# Show matched string only:
grep -o "info" data/example.txt
info
info

# Show matched string only:
grep -o "info.*" data/example.txt
info
informatics
```

cut

`cut` is for extracting columns from a data file. The two most important flags:

- `-d` specifies the file's delimiter
  - default: `\t` (tab)
  - Use `-d " "` for space or `-d ","` for comma
- `-f` specifies what column numbers to extract
  - `-f 1,3-5` gets columns 1, 3, 4, and 5 (in that order)
  - `-f 4,5,1,3,3` also gets columns 1, 3, 4, and 5 (in that order again!)

paste

`paste` is for adding columns

- `-d` specifies the delimiter (for the output)
- usually used with `cut`, e.g., `cut -f1 file.txt | paste - file.txt`

cut and paste examples

```
# The columns are in the wrong order and are comma-separated
cat data/ids.txt
ind1,fam1
ind2,fam2
ind3,fam3
ind4,fam4

# Extract the FID column to a new file
cut -d"," -f2 data/ids.txt > data/FID.txt
cat data/FID.txt
fam1
fam2
fam3
fam4

# Paste it together with the IID column (looks better now)
cut -d"," -f1 data/ids.txt | paste -d"\t" data/FID.txt -
fam1    ind1
fam2    ind2
fam3    ind3
fam4    ind4

# This time use a space delimiter:"
cut -d"," -f1 data/ids.txt | paste -d" " data/FID.txt - > data/corrected_ids.txt
cat data/corrected_ids.txt
fam1 ind1
fam2 ind2
fam3 ind3
fam4 ind4
```

cut and paste warning

Why not just do `cut -f2,1 -d"," data/ids.txt`? Because it doesn't work!

```
# Cut refuses to re-order the columns
# We get columns 1 and 2, but not in the desired order
cut -f2,1 -d"," data/ids.txt
ind1,fam1
ind2,fam2
ind3,fam3
ind4,fam4
```

Use hexdump -c to identify delimiters

Also handy for finding problematic and/or invisible characters in your files.

```
# File 1
cat data/mystery_delimiter_1.txt
What is my delimiter?

# File 2
cat data/mystery_delimiter_2.txt
What    is  my  delimiter?

# See file 1 is space-delimited
hexdump -c data/mystery_delimiter_1.txt
0000000   W   h   a   t       i   s       m   y       d   e   l   i   m
0000010   i   t   e   r   ?  \n
0000016

# See file 2 is tab-delimited
hexdump -c data/mystery_delimiter_2.txt
0000000   W   h   a   t  \t   i   s  \t   m   y  \t   d   e   l   i   m
0000010   i   t   e   r   ?  \n
0000016
```

Use column -t to make output more human-readable

`column -t` simply adds whitespace to align columns when printing to stdout:

```
# Hard to read
grep -v "^#" data/M.gtf | cut -f1-4 | head -n3
1	pseudogene	gene	3054233
1	unprocessed_pseudogene	transcript	3054233
1	unprocessed_pseudogene	exon	3054233

# Easy to read
grep -v "^#" data/M.gtf | cut -f1-4 | head -n3 | column -t
1  pseudogene              gene        3054233
1  unprocessed_pseudogene  transcript  3054233
1  unprocessed_pseudogene  exon        3054233
```

sort basics

```
# Top of an unsorted file
cat data/example.bed | head -n4
chr1    26  39
chr1    32  47
chr3    11  28
chr1    40  49

# Now sort it
sort -s data/example.bed | head -n4
chr1    10  19
chr1    26  39
chr1    32  47
chr1    40  49
```

- -s option means “if there are ties, keep them in the original order”
- -t designates delimiter (default is whitespace)

sort by specific columns with `-k`

`-k1,1` means "for the first sorting key, use col1."

`-k2,2` means "for the second sorting key, use col2."

Notice that you need to tell it `sort` to treat numbers numerically. Otherwise it sorts numbers "alphabetically" by their digits (so 9 comes after 10 since $9 > 1$).

```
# Sort by column 2 (doesn't work as intended!)
sort -s -k2,2 data/example.bed
chr1    10  19
chr3    11  28
chr3    16  27
chr1    26  39
chr1    32  47
chr2    35  54
chr1    40  49
chr1    9   28

# Sort by column 2 (as desired)
sort -s -k2,2n data/example.bed
chr1    9   28
chr1    10  19
chr3    11  28
chr3    16  27
chr1    26  39
chr1    32  47
chr2    35  54
chr1    40  49
```

sort by multiple columns and V

You use multiple -k options to sort by multiple columns. You need to include V to sort alphanumeric strings like chr1, chr2, . . .

```
# Sorting by chromosome and then start position (not as intended)
sort -s -k1,1 -k2,2n data/example2.bed | column -t
chr1   34  49
chr10  30  42
chr10  31  47
chr11  6   16
chr2   15  19
chr2   17  22
chr2   27  46
chr22  32  46

# Sorting by chromosome and then start position (as desired)
sort -s -k1,1V -k2,2n data/example2.bed | column -t
chr1   34  49
chr2   15  19
chr2   17  22
chr2   27  46
chr10  30  42
chr10  31  47
chr11  6   16
chr22  32  46
```

sort in reverse order with -r

```
# Sort by columns 1 and 2, but reverse order for 2
sort -s -k1,1V -k2,2nr data/example2.bed | column -t
chr1   34  49
chr2   27  46
chr2   17  22
chr2   15  19
chr10  31  47
chr10  30  42
chr11  6   16
chr22  32  46
```

Use `sort -c` to check if a file is already sorted how you want

```
# Is it already sorted by col1 and col2?
# sort -c says no and tells us why:
sort -s -k1,1V -k2,2n -c data/example2.bed
sort: data/example2.bed:3: disorder: chr10  31  47

# Let's make a sorted version (example2_sorted.bed) and ask again
# No output from sort -c means the file is sorted
sort -s -k1,1V -k2,2n data/example2.bed > data/example2_sorted.bed
sort -s -k1,1V -k2,2n -c data/example2_sorted.bed
```

`uniq`

`uniq` drops duplicate rows ***if they are consecutive***.

```
# There are adjacent duplicate rows
cat data/letters.txt
A
A
B
C
B
C
C
C

# uniq removes the adjacent duplicate rows (non-adjacent duplicates remain)
uniq data/letters.txt
A
B
C
B
C
```

Pipe `sort` into `uniq` to remove all duplicate lines

`sort` first and pipe to `uniq` to remove all duplicates and end up with unique lines.

```
# sort | uniq leaves only the unique lines
sort data/letters.txt | uniq
A
B
C
```

`sort uniq sort` to count duplicate lines

`uniq -c` counts duplicate lines. You can sort *again* to put the most common lines first.

You can also use `uniq -d` to output a single copy of each duplicated line.

```
# Counting the duplicate lines
sort data/letters.txt | uniq -c
   2 A
   2 B
   4 C

# Better yet, put the most common ones first
sort data/letters.txt | uniq -c | sort -k1,1nr
   4 C
   2 A
   2 B

# Or use uniq -d to help count the number unique duplicated rows
# uniq -d prints a single copy of each duplicated row only
sort data/letters.txt | uniq -d | wc -l
       3
```

Example

Summarizing the lines of a file:

```
grep -v "^#" data/M.gtf | cut -f3 | sort | uniq -c | sort -rn
36128 exon
25901 CDS
7588 UTR
4993 transcript
2299 stop_codon
2290 start_codon
2027 gene
```

awk

- a whole programming language
- works a bit like `filter` and `select`/`mutate` in tidyverse
- rows are “records”, columns are “fields”
- $0 refers to an entire records
- $1, $2, $3, etc., are columns 1, 2, 3...
- commands look like: `awk 'pattern { action }' file`
  - `awk` scans each record, applying the action of the record matches the pattern
  - the pattern selects the records
    (leaving out the pattern selects all records)
  - the action says what to do
    (leaving out the action results in a default action of printing)
- `-F` specifies the field separator (default: whitespace)

awk example (no pattern given)

```
# No pattern given; the action is to print each record
awk '{ print $0 }' data/example.bed
chr1    26  39
chr1    32  47
chr3    11  28
chr1    40  49
chr3    16  27
chr1    9   28
chr2    35  54
chr1    10  19

# No pattern given; print col2, a tab, then col3
awk '{ print $2 "\t" $3 }' data/example.bed
26  39
32  47
11  28
40  49
16  27
9   28
35  54
10  19
```

awk logical operators for patterns

Pattern	Meaning
a == b	ais equal to b
a != b	a is not equal to b
a < b	a is less than b
a > b	a is greater than b
a <= b	a is less than or equal to b
a >= b	a is greater than or equal to b
a ~ /b/	a matches the regular expression b
a ~! /b/	a doesn't match the regular expression b
a && b	a and b are both true
a || b	a or b (or both) are true
!a	a is not true

awk pattern examples

```
# Print all lines where col1 matches regex chr3
awk '$1 ~ /chr3/' data/example.bed
chr3    11  28
chr3    16  27

# Print all lines where the difference between col3 and col2 is at least 18
awk '$3 - $2 > 18' data/example.bed
chr1    9   28
chr2    35  54
```

awk examples with pattern and action

```
# Pattern - field 1 is either chr2 or chr3
# Action - print the record, a tab, and then difference between col3 and col2
awk '$1 ~ /chr2|chr3/ { print $0 "\t" $3 - $2 }' data/example.bed
chr3    11  28  17
chr3    16  27  11
chr2    35  54  19

# Notice that including the \t delimiter is necessary
awk '$1 ~ /chr2|chr3/ { print $1 $2 $3 $3 - $2 }' data/example.bed
chr3112817
chr3162711
chr2355419

# Or we can use a comma, which awk interprets as a space delimiter
awk '$1 ~ /chr2|chr3/ { print $1,$2,$3,$3 - $2 }' data/example.bed
chr3 11 28 17
chr3 16 27 11
chr2 35 54 19
```

Counting columns with `awk`

Use `awk '{print NF; exit}'` to get the number of columns. (This command assumes every row has the same number of columns!)

```
# Obviously there are 3 columns
head -n3 data/Mus_musculus.GRCm38.75_chr1.bed
1	3054233 3054733
1	3054233 3054733
1	3054233 3054733

# Let’s have awk check for us
awk ’{print NF; exit}’ data/Mus_musculus.GRCm38.75_chr1.bed
3
```

sed for substitution

- stream editor
- good for substitution (default: only the first occurrence per line)
- sed commands look like: `sed 's/pattern/replacement/options' file`
- options include `i` (ignore case) and `g` (replace all occurrences per line)

```
# The original file
head -n3 data/chroms2.txt
chrom1 POS3214482 POS3216968
chrom1 POS3216025 POS3216968
chrom1 POS3216022 POS3216024

# Replace chrom with chr
sed 's/chrom/chr/' data/chroms2.txt | head -n3
chr1 POS3214482 POS3216968
chr1 POS3216025 POS3216968
chr1 POS3216022 POS3216024

# Also delete POS
sed 's/chrom/chr/' data/chroms2.txt | sed 's/POS//' | head -n3
chr1 3214482 POS3216968
chr1 3216025 POS3216968
chr1 3216022 POS3216024

# Oops, delete POS globally
sed 's/chrom/chr/' data/chroms2.txt | sed 's/POS//g' | head -n3
chr1 3214482 3216968
chr1 3216025 3216968
chr1 3216022 3216024
```

sed for deletion

- `sed '/pattern/d' file` deletes any lines matching the pattern
- `sed 'Nd' file` deletes line N
- `sed` streams the results by default - it doesn't change the source file
- `sed` has options for editing files in place (over-writing the original)

tr for "translating" characters

To use `tr` to replace `a` with `b` in a file, do either:

- `cat file | tr 'a' 'b'` or
- `tr 'a' 'b' < file` (redirect standard in; that's just how it is)

Example: replace `D` with `R`:

```
echo "DNA" | tr 'D' 'R'
RNA
```

You can translate multiple characters simultaneously:

```
# Notice the results are very different here!
echo 'abc' | tr 'a' 'c' | tr 'b' 'd' | tr 'c' 'e'
ede
echo 'abc' | tr 'abc' 'cde'
cde
```

tr -d for deleting characters

```
# The file as-is
cat data/letters.txt
A
A
B
C
B
C
C
C

# Delete newline characters
cat data/letters.txt | tr -d "\n"
AABCBCCCprintf "\n"

# Delete C, too
cat data/letters.txt | tr -d "\nC"
AABB
```

tr -s for collapsing repeated characters

```
# Remove extra spaces, e.g.

echo 'Five     spaces     between     each     word?'
Five     spaces     between     each     word?

echo 'Five     spaces     between     each     word?' | tr -s ' '
Five spaces between each word?
```

`join`

You can merge data sets with `join`.

- Both files have to be sorted first
- `Syntax: join -1 file1key -2 file2key file1 file2`
- `file1key` and `file2key` are the column numbers to be used as the merging key
- Default: inner join
- You aren't expected to use `join` in this course

`diff -s` to determine if two files are identical

```
# This pair of files are obviously the same
cp data/letters.txt data/letters_copy.txt
diff -s data/letters.txt data/letters_copy.txt
Files data/letters.txt and data/letters_copy.txt are identical

# This pair of files is obviously different
tail -n+3 data/letters.txt > data/letters_shortened.txt
diff -s data/letters.txt data/letters_shortened.txt
1,2d0
< A
< A
Files data/letters.txt and data/letters_shortened.txt are identical
```

- Note that `diff -s` has nonzero exit status when there is a difference! (So it can cause scripts to fail if you have `set -euo pipefail`.)
- Google `diff` if you need to understand the output