GEOS 436 / 636
Programming and Automation for Geoscientists

– Week 10: Unix – Scrubbing Data –

Ronni Grapenthin
rgrapenthin@alaska.edu
Elvey 413B
x7682

We now know how to get data!

How do we:

- get it into the format that we need,
- extract the interesting bits, or
- analyze it

. . . preferably on the command line?

It's easy with some key command line tools!

Common Operations on Plain Text

- Filtering lines or fields
- Replacing / deleting values
- Merging data

These operations require generic ways to find text patterns of interest:

Regular Expressions

Regular Expressions

- Set of characters that specify a generic pattern
- Example: Format of a simple URL is *www-dot-bunchOfCharactersNumbers-dot-someTopLevelDomain*
- Regular Expressions help formalize that
- Allow to search and substitute text on the command line easy
- Are accepted by many Unix commands, incl. grep, sed, awk, . . . (syntax may vary slightly)

Simple Regular Expression Symbols

(surround regular expressions with single quotes to prevent interpretation by the shell)

- `.` (single period): matches any single character (wildcard)
- `B`: matches uppercase ‘B’ (as an example for any letter)
- `b`: matches uppercase ‘b’ (as an example for any letter)
- `*`: match **zero or more** of the previous pattern
- `?`: match **zero or one** of the previous pattern
- `+`: match **one or more** of the previous pattern
- `{ }`: match **number of occurrences** of the previous pattern

Simple Regular Expression Symbols

- `^`: search for pattern at beginning of string (line)
- `$`: end of the string (line)
- `\`: escape special character
- `[ ]`: match range of characters inside brackets
- `[^ ]`: DO NOT match range of characters inside brackets

Non-printable characters:

- `\t` – tabulator character
- `\r` – carriage return
- `\n` – new line
- `\s` – white space

Examples of uses of regular expressions in applications of the unix commandline tools below.

Powerful Commandline Tools

- `tr`: translate or delete characters
- `grep`: print lines that match patterns
- `sed`: stream editor for filtering and transforming text
- `awk`: pattern scanning and text processing language
- usually piped into each other in various combinations

tr

- `tr` is a character-based translator (`sed` is word based)
- Common uses: change uppercase to lowercase text, delete specific characters

```
tr OPTIONS SET1 [SET2]
```

- translate characters given in `SET1` to those in `SET2`
- `-d`: delete characters given in `SET1` (SET2 not needed)

tr Example

```
roon@rn-xps:~$ cat station.txt
ANMO 34.9500 -106.4600 1820 1 2
BMT 34.2750 -107.2600 1987 1 4
CBET 32.4200 -103.9900 1042 1 6
WTX 34.0722 -106.9460 1555 1 22
roon@rn-xps:~$
roon@rn-xps:~$ cat station.txt | tr -d '-'
ANMO 34.9500 106.4600 1820 1 2
BMT 34.2750 107.2600 1987 1 4
CBET 32.4200 103.9900 1042 1 6
WTX 34.0722 106.9460 1555 1 22
roon@rn-xps:~$ cat station.txt | tr '[A-Z]' '[a-z]'
anmo 34.9500 -106.4600 1820 1 2
bmt 34.2750 -107.2600 1987 1 4
cbet 32.4200 -103.9900 1042 1 6
wtx 34.0722 -106.9460 1555 1 22
roon@rn-xps:~$
```

grep

- match lines from input based on search pattern
- input can be stdin, one or multiple files!
- different versions used to exist, all captured by various options of `grep`

```
grep pattern in_file
```

- All lines in `in_file` that contain `pattern` are printed
- `pattern` can be a simple string or regular expression (`-E` provides extended implementation)
- `grep -v` inverts the pattern and returns only lines that DO NOT match

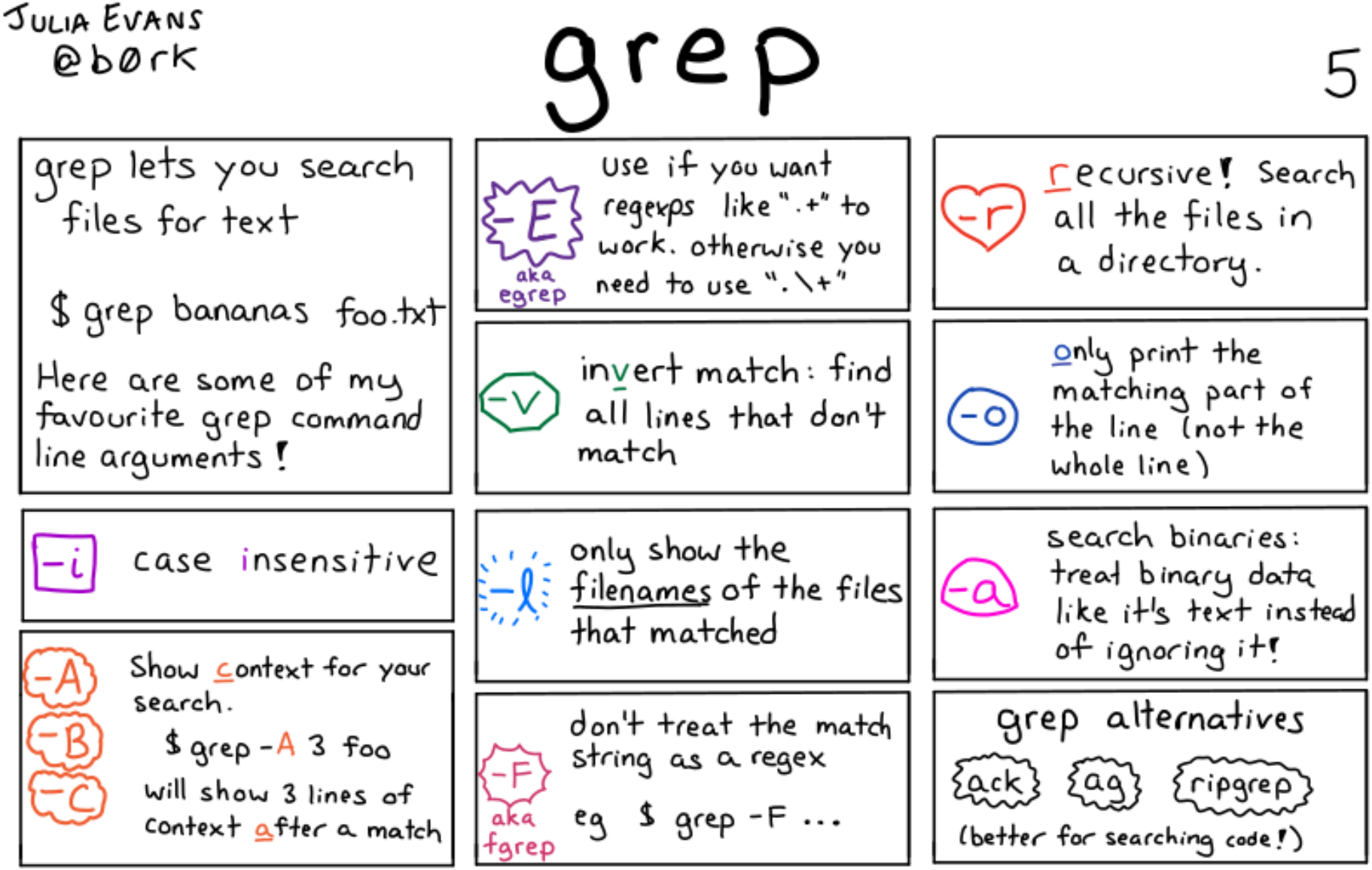
grep Example

```
roon@rn-xps:~$ cat station.txt | wc -l
23
roon@rn-xps:~$ grep 'B' station.txt
BAR 34.1500 -106.6280 2121 1 3
BMT 34.2750 -107.2600 1987 1 4
CBET 32.4200 -103.9900 1042 1 6
CL2B 32.2300 -103.8800 2121 1 7
SBY 33.9752 -107.1810 3230 1 16
roon@rn-xps:~$
roon@rn-xps:~$ grep '^B' station.txt
BAR 34.1500 -106.6280 2121 1 3
BMT 34.2750 -107.2600 1987 1 4
roon@rn-xps:~$
roon@rn-xps:~$ grep 'M.M' station.txt
MLM 34.8100 -107.1450 2088 1 15
```

grep Example

```
roon@rn-xps:~$ grep '.*S' station.txt
HTMS 32.4700 -103.6000 1192 1 12
SBY 33.9752 -107.1810 3230 1 16
SMC 33.7787 -107.0190 1560 1 17
SRH 32.4914 -104.5150 1276 1 18
SSS 32.3500 -103.4100 1072 1 19
roon@rn-xps:~$
roon@rn-xps:~$ grep '.\+S' station.txt
HTMS 32.4700 -103.6000 1192 1 12
SSS 32.3500 -103.4100 1072 1 19
roon@rn-xps:~$
roon@rn-xps:~$ grep '0\{2\}' station.txt
ANMO 34.9500 -106.4600 1820 1 2
BAR 34.1500 -106.6280 2121 1 3
BMT 34.2750 -107.2600 1987 1 4
CBET 32.4200 -103.9900 1042 1 6
CL2B 32.2300 -103.8800 2121 1 7
CL7 32.4400 -103.8100 1032 1 8
GDL2 32.2003 -104.3640 1213 1 11
HTMS 32.4700 -103.6000 1192 1 12
MLM 34.8100 -107.1450 2088 1 15
SSS 32.3500 -103.4100 1072 1 19
roon@rn-xps:~$
```

grep Summary



https://wizardzines.com/comics/

`sed` stream editor

- Edit input line by line via command line
- Most common use: find-and-replace on text files

```
sed 's/string1/string2/g' in_file > out_file
```

- Replace every occurrence of `string1` in `in_file` with `string2`, output is redirected to `out_file`
- `s` in command string is for *subsitute*
- `g` means *global* – subsitute for every match, not just the first one
- `sed -i` will change file in place (`in_file` would contain `string2`)

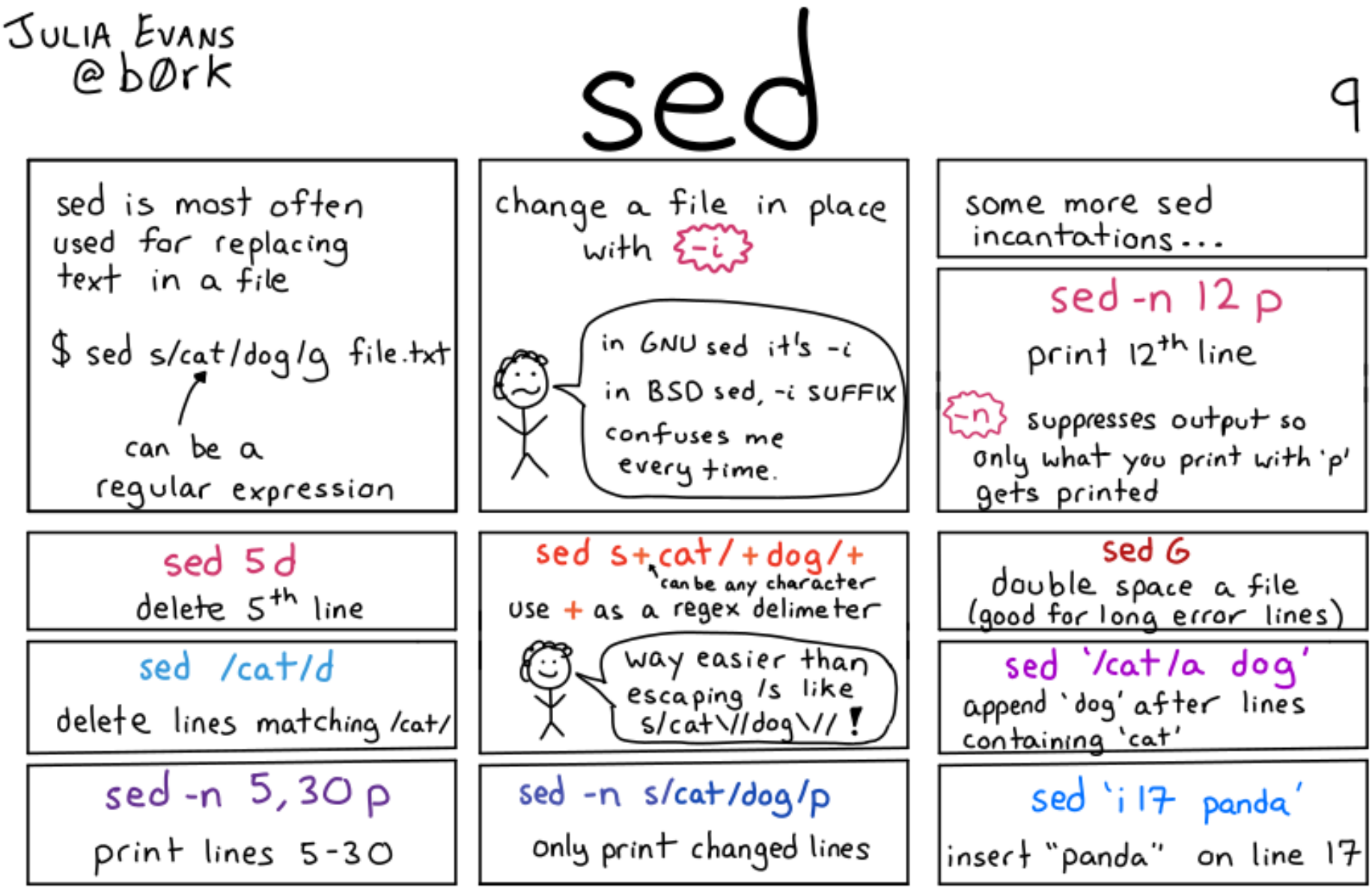
sed Example

```
roon@rn-xps:~$ cat station.txt
Name Lat Lon Elevation Type Number
ANMO 34.9500 -106.4600 1820 1 2
BAR 34.1500 -106.6280 2121 1 3
BMT 34.2750 -107.2600 1987 1 4
CAR 33.9525 -106.7340 1658 1 5
CBET 32.4200 -103.9900 1042 1 6
roon@rn-xps:~$
roon@rn-xps:~$ sed 's/ /\t/g' station.txt
Name    Lat     Lon     Elevation       Type    Number
ANMO    34.9500 -106.4600       1820    1       2
BAR     34.1500 -106.6280       2121    1       3
BMT     34.2750 -107.2600       1987    1       4
CAR     33.9525 -106.7340       1658    1       5
CBET    32.4200 -103.9900       1042    1       6
roon@rn-xps:~$
roon@rn-xps:~$ sed 's/ /,/g' station.txt > station.csv && cat station.csv
Name,Lat,Lon,Elevation,Type,Number
ANMO,34.9500,-106.4600,1820,1,2
BAR,34.1500,-106.6280,2121,1,3
BMT,34.2750,-107.2600,1987,1,4
CAR,33.9525,-106.7340,1658,1,5
CBET,32.4200,-103.9900,1042,1,6
roon@rn-xps:~$
```

sed Example

```
roon@rn-xps:~$ cat station.txt
Name Lat Lon Elevation Type Number
ANMO 34.9500 -106.4600 1820 1 2
BAR 34.1500 -106.6280 2121 1 3
BMT 34.2750 -107.2600 1987 1 4
CAR 33.9525 -106.7340 1658 1 5
CBET 32.4200 -103.9900 1042 1 6
roon@rn-xps:~$
roon@rn-xps:~$ sed 's/ [0-9]\{4\} [0-9] [0-9]//g' station.txt
Name Lat Lon Elevation Type Number
ANMO 34.9500 -106.4600
BAR 34.1500 -106.6280
BMT 34.2750 -107.2600
CAR 33.9525 -106.7340
CBET 32.4200 -103.9900
roon@rn-xps:~$
```

sed Summary



https://wizardzines.com/comics/

awk

- Programming language to process files of text, reads input a line at a time
- Does floating point math
- Uses:
  - Get the fourth, sixth, and 27th column of the file?
  - Reorder columns
  - Calculate the difference between numbers in 5th and 6th columns, divided by the square root of the sum of squares of the numbers inthe first 3 columns
  - Get the all entries that are within a certain range of numbers (e.g., geographic bounding boxes)
  - . . .

awk

- awk program is a sequence of `pattern {action}` statements operating on a file
- If `pattern` matches, `action` is executed
- File treated as sequence of records (by default each line is a record)
- Each record is broken into a sequence of fields (columns), separated by FS (field separator), whitespace by default
- Each field is assigned to a variable: `$1` through `$NF` where `NF` is a special variable for total number of fields; `$0` contains the complete line

awk

```
awk 'command string' files
```

- `command string` is of the form `pattern {action}` and can have many pattern-action pairs
- Example: `awk 'NF > 3 {print $4}' some_file.txt`
- If there are more than 3 fields, print out the 4th field.

awk -F/-f/-v

Change Field Separator:

- `awk -F` option changes field separator
- To work on CSV files: `awk -F, '{...}' somefile.csv`

Provide program in separate file

- `awk -f awk-script somefile.txt`
- instead of defining the program text on the command line, provide it in a separate file.

Assign a value to a variable:

- `awk -v var=value '{...}' somefile.txt`
- assigns `value` to program variable `var`
- easy way to pass values from shell scripts into awk (e.g. lat-lon bounds)
- can use `var` in script (no leading $ just '`var`' to reference)

`awk` print (& other functions)

- `print` is most common action
- awk implements range of string & arithmetic functions, e.g.:
  - `substr(s,i,n)` return substring of string s, starting at index i, of length n (length can be ommitted)
  - `sqrt(x)` returns square root of x
  - `atan2(x,y)` returns arctan of y/x between $-\Pi$ and Π
  - check man pages for more
- you can write your own functions (with if and while and for)
- worth to read the documentation / manuals / books to dive deeper

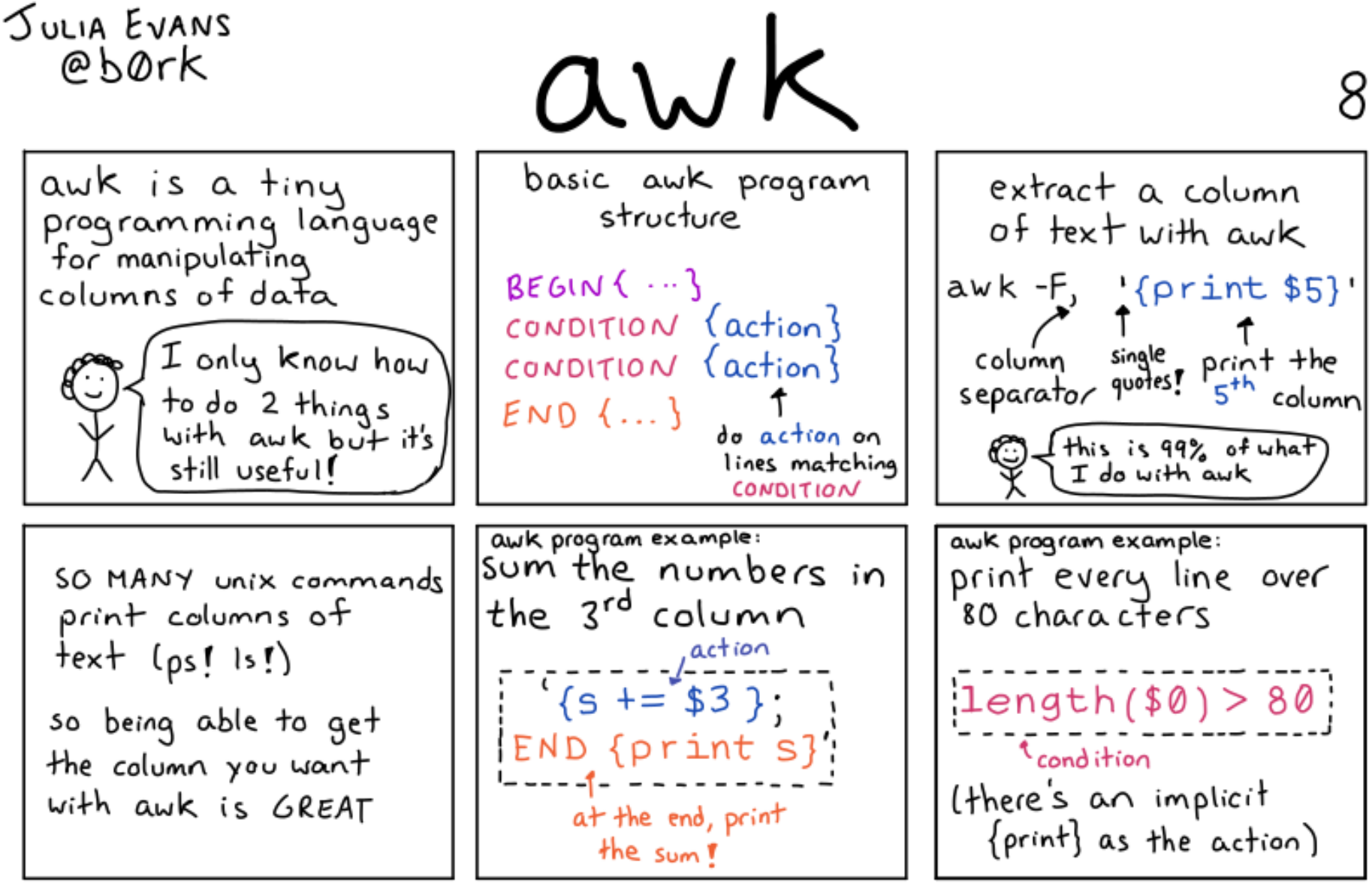
awk Example

```
roon@rn-xps:~$ cat station.txt
ANMO 34.9500 -106.4600 1820 1 2
BMT 34.2750 -107.2600 1987 1 4
CBET 32.4200 -103.9900 1042 1 6
WTX 34.0722 -106.9460 1555 1 22
roon@rn-xps:~$ awk '{print $1}' station.txt
ANMO
BMT
CBET
WTX
roon@rn-xps:~$ awk '$2<34.1 {print $1}' station.txt
CBET
WTX
roon@rn-xps:~$ awk '$2<34.1 {print $0}' station.txt
CBET 32.4200 -103.9900 1042 1 6
WTX 34.0722 -106.9460 1555 1 22
```

awk Example

```
roon@rn-xps:~$ awk '$2<34.1 {print $0}' station.txt
CBET 32.4200 -103.9900 1042 1 6
WTX 34.0722 -106.9460 1555 1 22
roon@rn-xps:~$ awk '$2<34.1 && $3>-106 {print $0}' station.txt
CBET 32.4200 -103.9900 1042 1 6
roon@rn-xps:~$ awk '{print $3,$2,$1}' station.txt
-106.4600 34.9500 ANMO
-107.2600 34.2750 BMT
-103.9900 32.4200 CBET
-106.9460 34.0722 WTX
roon@rn-xps:~$ awk '{print $3,$2,$1}' station.txt  | sort
-103.9900 32.4200 CBET
-106.4600 34.9500 ANMO
-106.9460 34.0722 WTX
-107.2600 34.2750 BMT
roon@rn-xps:~$
```

awk Summary



https://wizardzines.com/comics/