LING 408/508: Computational Techniques for Linguists

Lecture 14

Today's Topics

- Reading assignment:
  - https://www.gnu.org/software/gawk/manual/html_node/Regexp.html

- *Let's make sure everyone is on board with associative arrays in* awk
- A fundamental data structure the same as:
  - dict in Python
  - hash in Perl

- gsub
- gensub

awk: regex

3 Regular Expressions

A *regular expression*, or *regexp*, is a way of describing a set of strings. Because regular expressions are such a fundamental part of `awk` programming, their format and use deserve a separate chapter.

A regular expression enclosed in slashes (‘`/`’) is an `awk` pattern that matches every input record whose text belongs to that set. The simplest regular expression is a sequence of letters, numbers, or both. Such a regexp matches any string that contains that sequence. Thus, the regexp ‘`foo`’ matches any string containing ‘`foo`’. Thus, the pattern `/foo/` matches any input record containing the three adjacent characters ‘`foo`’ *anywhere* in the record. Other kinds of regexps let you specify more complicated classes of strings.

Initially, the examples in this chapter are simple. As we explain more about how regular expressions work, we present more complicated instances.

awk doesn't support `\d` (digit) etc. from Perl, use `[[:digit:]]` instead

awk: regex

```
1  RESOURCE      PHONE NUMBER
2  University of Arizona Police Department (UAPD)    520-621-8273
3  Risk Management and Safety   520-621-1790
4  Office of Radiation, Chemical and Biological Safety      520-626-6850
5  Arizona Institutional Biosafety Committee   520-621-5279
6  Campus Health Service 520-621-6490
7  Dean of Students Office   520-621-7057
8  Facilities Management    520-621-3000
9  Arizona Poison and Drug Information Center  800-222-1222
10 Recorded updates during campus emergencies 520-626-1222 (Tucson) 800-362-0101 (Toll free)
```

NR = number of lines
NF = number of fields in a line
variable[key] associative array
variable[key] = value

not case insensitive

Build a table of all the words used:

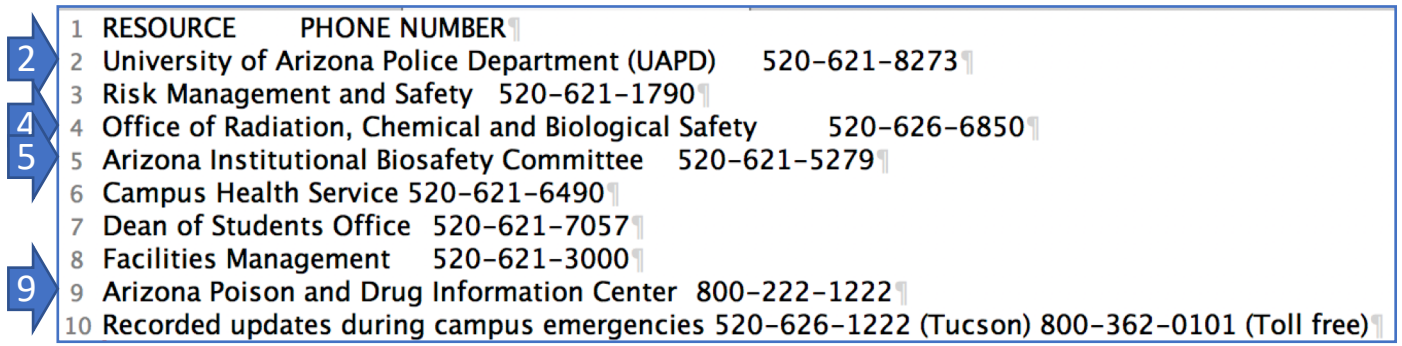
```
awk 'NR!=1 {for (i=1; i<=NF; i++) {word[$i]+=1}}
END {for (x in word) { printf "%12s %d\n", x, word[x]}}'
uanumbers.txt | sort –k 2 –n
```

`NR!=1` (pattern) skip 1st line (`!=` means *not equal to*)
`NF` = number of fields on a line
`word` = associative array of frequencies
`|` = pipe (output of `awk` into `sort`)
`sort –k2 –n` = command to sort on field 2 numerically (–n)

```
Arizona 3
and 3
of 3
Management 2
Safety 2
Office 2
Institutional 1
800-362-0101 1
800-222-1222 1
520-626-6850 1
…
520-621-1790 1
emergencies 1
…
during 1
campus 1
Police 1
Poison 1
Health 1
Center 1
Campus 1
…
```

awk: regex

- As we read each $1, $2, etc.. field, we populate the associative array word:
  1. word["University"] = 1
  2. word["of"] = 1
  3. word["Arizona"] = 1
  4. …
  5. word["of"] = 2
  6. …
  7. word["Arizona"] = 2
  8. …
  9. …
  10. word["Arizona"] = 3
  11. …



awk: regex

```
1  RESOURCE       PHONE NUMBER
2  University of Arizona Police Department (UAPD)     520-621-8273
3  Risk Management and Safety   520-621-1790
4  Office of Radiation, Chemical and Biological Safety       520-626-6850
5  Arizona Institutional Biosafety Committee    520-621-5279
6  Campus Health Service 520-621-6490
7  Dean of Students Office   520-621-7057
8  Facilities Management     520-621-3000
9  Arizona Poison and Drug Information Center  800-222-1222
10 Recorded updates during campus emergencies 520-626-1222 (Tucson) 800-362-0101 (Toll free)
```

Build a table of all the words used (case-insensitive):

```
awk 'NR!=1 {for (i=1; i<=NF; i++) {word[tolower($i)]+=1}}
END {for (x in word) { printf "%12s %d\n", x, word[x]}}'
uanumbers.txt | sort –k 2 –nr
```

tolower(*string*)

Return a copy of *string*, with each uppercase character in the string replaced with its corresponding lowercase character. Nonalphabetic characters are left unchanged. For example, tolower("MiXeD cAsE 123") returns "mixed case 123".
https://www.gnu.org/software/gawk/manual/html_node/String-Functions.html

```
arizona 3
and 3
of 3
management 2
safety 2
office 2
campus 2
institutional 1
800-362-0101 1
…
520-621-1790 1
information 1
emergencies 1
university 1
…
biosafety 1
students 1
recorded 1
chemical 1
(tucson) 1
…
```

awk: regex

```
1  RESOURCE       PHONE NUMBER
2  University of Arizona Police Department (UAPD)     520-621-8273
3  Risk Management and Safety   520-621-1790
4  Office of Radiation, Chemical and Biological Safety       520-626-6850
5  Arizona Institutional Biosafety Committee    520-621-5279
6  Campus Health Service 520-621-6490
7  Dean of Students Office   520-621-7057
8  Facilities Management     520-621-3000
9  Arizona Poison and Drug Information Center  800-222-1222
10 Recorded updates during campus emergencies 520-626-1222 (Tucson) 800-362-0101 (Toll free)
```

Build a table of all the words used (no numbers, no punctuation):

```
gawk 'NR!=1 {for (i=1; i<=NF; i++) {gsub(/[^A-Za-z]/, "", $i);
word[tolower($i)]+=1}} END {for (x in word) { printf "%12s %d\n",
x, word[x]}}' uanumbers.txt | sort –k 2 –nr
```

gsub(*regexp*, *replacement* [, *target*])
Search *target* for *all* of the longest, leftmost, *nonoverlapping* matching substrings it can find and replace them with *replacement*. The 'g' in gsub() stands for "global," which means replace everywhere.
https://www.gnu.org/software/gawk/manual/html_node/String-Functions.html

```
arizona 3
and 3
of 3
management 2
safety 2
office 2
campus 2
institutional 1
information 1
emergencies 1
university 1
facilities 1
department 1
biological 1
radiation 1
committee 1
biosafety 1
students 1
recorded 1
chemical 1
updates 1
service 1
tucson 1
police 1
```

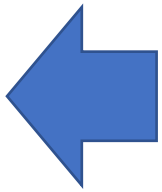
awk: regex

3 Regular Expressions

A *regular expression*, or *regexp*, is a way of describing a set of strings. Because regular expressions are such a fundamental part of `awk` programming, their format and use deserve a separate chapter.

A regular expression enclosed in slashes (‘`/`’) is an `awk` pattern that matches every input record whose text belongs to that set. The simplest regular expression is a sequence of letters, numbers, or both. Such a regexp matches any string that contains that sequence. Thus, the regexp ‘`foo`’ matches any string containing ‘`foo`’. Thus, the pattern `/foo/` matches any input record containing the three adjacent characters ‘`foo`’ *anywhere* in the record. Other kinds of regexps let you specify more complicated classes of strings.

Initially, the examples in this chapter are simple. As we explain more about how regular expressions work, we present more complicated instances.



awk: gensub

- https://www.gnu.org/software/gawk/manual/html_node/String-Functions.html
- gensub(*regexp*, *replacement*, *how* **[**, *target***]**)
  - Search the target string *target* for matches of the regular expression *regexp*.
  - If *how* is a string beginning with ‘g’ or ‘G’ (short for “global”), then replace all matches of *regexp* with *replacement*. Otherwise, *how* is treated as a number indicating which match of *regexp* to replace.
  - If no *target* is supplied, use $0.
  - It returns the modified string as the result of the function and the original target string is *not* changed.
  - gensub() provides an additional feature that is not available in sub() or gsub(): the ability to specify components of a regexp in the replacement text. This is done by using parentheses in the regexp to mark the components and then specifying ‘*N*’ in the replacement text, where *N*is a digit from 1 to 9.

awk: gensub

- If only a BEGIN section, no need to provide a file (to process line by line).
  ```
  gawk 'BEGIN {print "hello"}'
  ```
  hello
- –v (sets variable):
  ```
  gawk –v n="hello" 'BEGIN {print n, "\n"}'
  ```
  hello
- gensub(regex, replacement,how, target)
- regex = `/(.+) (.+)/`
- replacement = `"\\2 \\1"`
- how = "g"
- target = (variable) n
  ```
  gawk –v n="hello goodbye" 'BEGIN {print gensub(/(.+) (.+)/, "\\2 \\1", "g", n), "\n"}'
  ```
- What happens?