STAT 605
Data Science Computing

Introduction to `sed` and `awk`

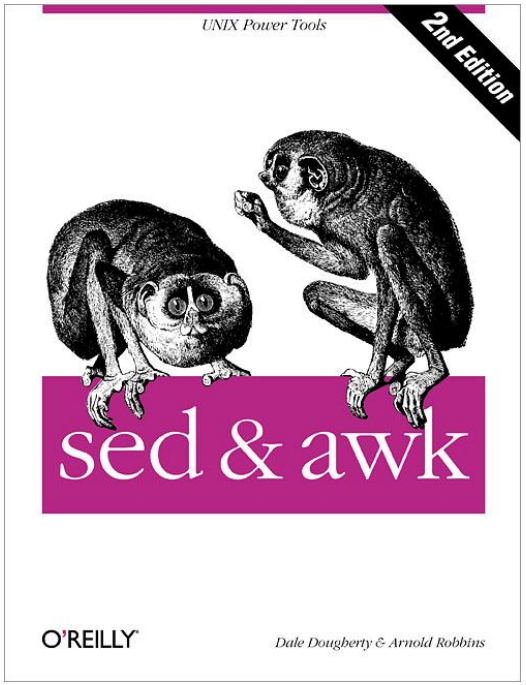
Editing text streams: `sed`

`sed` is short for **stream editor**

One of the most powerful and versatile UNIX tools

Commonly paired with `awk`
small command line language for string processing

Has lots of features, but we'll focus on one: **substitutions**




```
keith:~$ echo "hello world" | sed 's/hello/goodbye/g'
goodbye world
```

`s` for substitute

Replace this...

...with this.

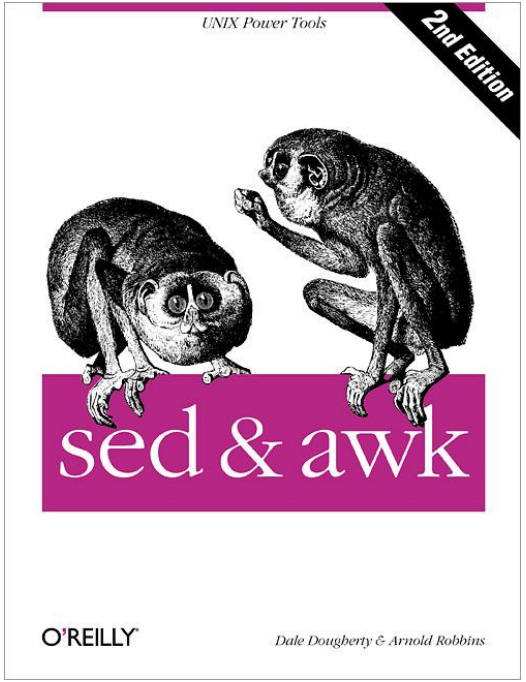
`g` for globally, meaning everywhere in the input.

Editing text streams: `sed`

`sed` commands can include regular expressions

```
keith:~$ echo "a aa aaa" | sed 's/a*/b/g'
b b b
```

`'*'` works like in `egrep`



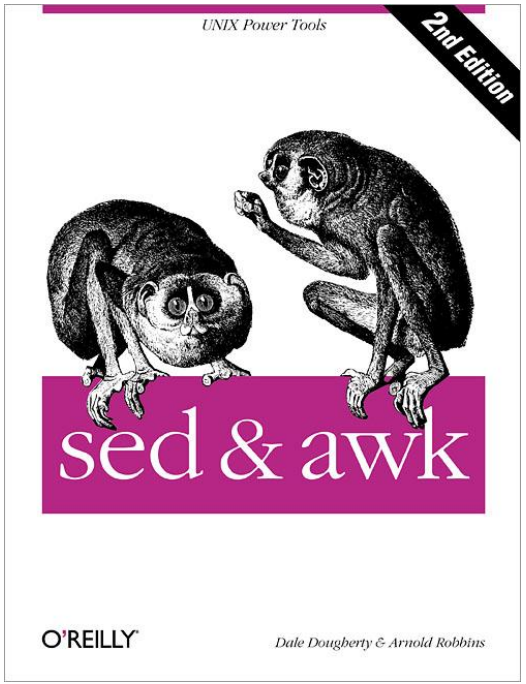
Editing text streams: `sed`

`sed` commands can include regular expressions

```
keith:~$ echo "a aa aaa" | sed 's/a*/b/g'
b b b
```

`'*'` Works like in `egrep`

Test your understanding: is the `sed` regex matcher greedy?



Editing text streams: `sed`

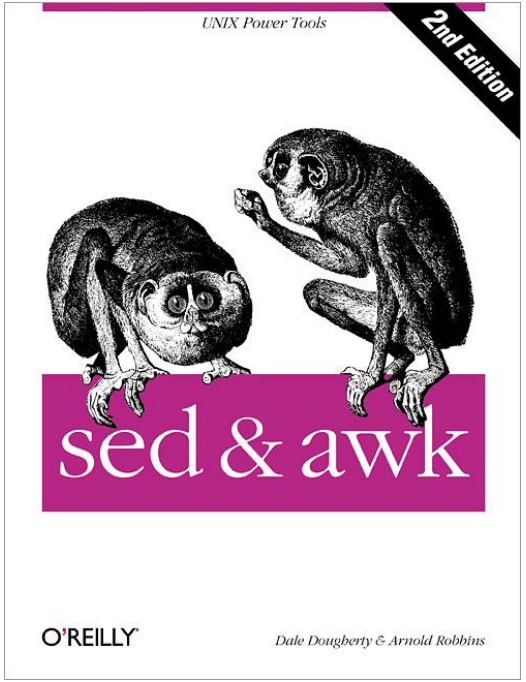
`sed` commands can include regular expressions

```
keith:~$ echo "a aa aaa" | sed 's/a*/b/g'
b b b
```

`'*'` Works like in `egrep`

Test your understanding: is the `sed` * operator greedy?

Answer: yes! If it were lazy, above would output just a mess of `'b'`s



Editing text streams: `sed`

`sed` commands can include regular expressions

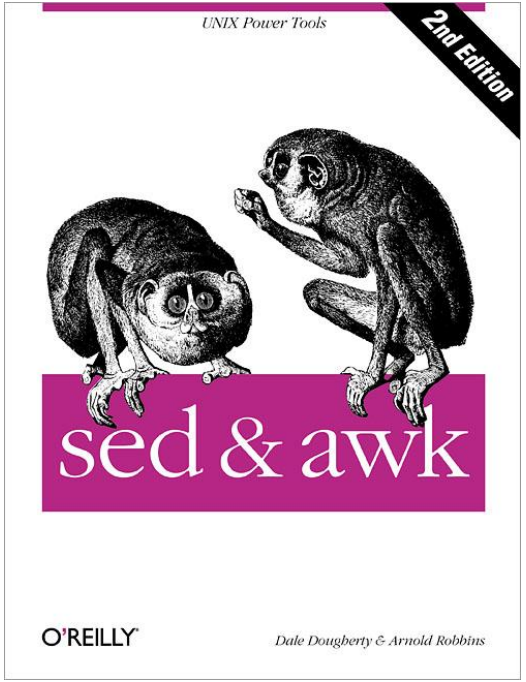
```
keith:~$ echo "a aa aaa" | sed 's/a*/b/g'
b b b
```

`'*'` Works like in `egrep`

Test your understanding: is the `sed` * operator greedy?

Answer: yes! If it were lazy, above would output just a mess of `'b'`s

As promised, most of your knowledge of regexes in `egrep` will transfer directly to `sed`, as well as other tools (e.g., `vim`, `emacs`, `Python` and `perl`)



Editing text streams: `sed`

`sed` commands can include regular expressions

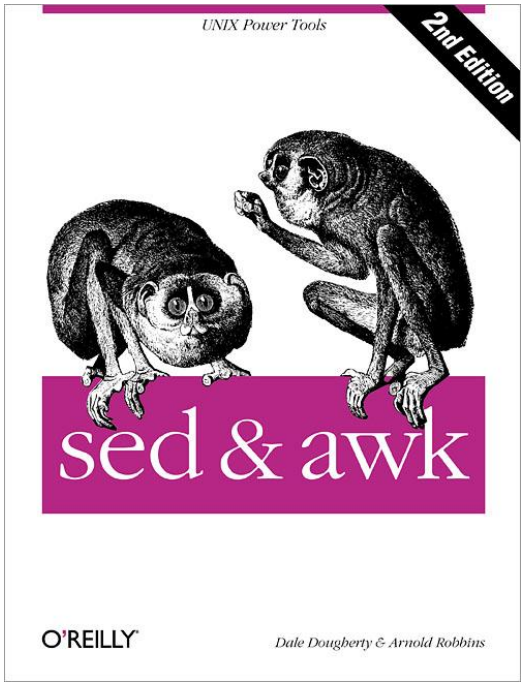
```
keith:~$ echo "a aa aaa" | sed 's/a*/b/g'
b b b
```

'*' Works like in `egrep`

Basic syntax of `sed s` commands:
`sed 's/regexp/replacement/flags'`

```
keith:~$ echo "a aa aaa" | sed -E 's/a+/b/g'
b b b
keith:~$
```

To use "extended" regexes, need to give `-E` flag (there is no `esed`, unfortunately).



Editing text streams: `sed`

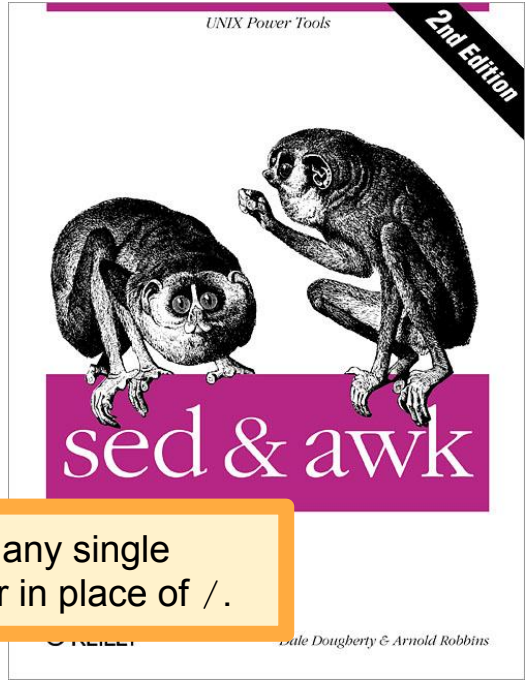
Basic syntax of `sed s` commands:
`sed 's/regexp/replacement/flags'`

```
keith:~$ echo "a aa aaa" | sed -E 's/a+/b/g'
b b b
keith:~$ echo "a aa aaa" | sed -E 's|a+|b|g'
b b b
keith:~$ echo "a| aa| aaa| aaaa" | sed -E 's/a+\|/b/g'
b b b aaaa
keith:~$
```

Can use any single character in place of `/`.

Special characters have to be escaped.

Of course, we're only barely scratching the surface:
https://www.gnu.org/software/sed/manual/html_node/index.html#Top



Quick and dirty text processing: `awk`

`awk` is a command-line program that runs its own programming language, `AWK`

Like `grep` and `sed`, `awk` operates on a data stream, read from its `stdin`
Primarily designed for text processing

`awk` is a **data driven** programming language
"Describe what pattern to look for, and what to do when you find it."
In contrast to **procedural** programming languages (e.g., R and Python)

Much of what follows is based on materials from *The GNU Awk User's Guide* available at https://www.gnu.org/software/gawk/manual/gawk.html

Basic `awk`: patterns and actions

Basic `awk` program: series of (pattern, action) pairs.

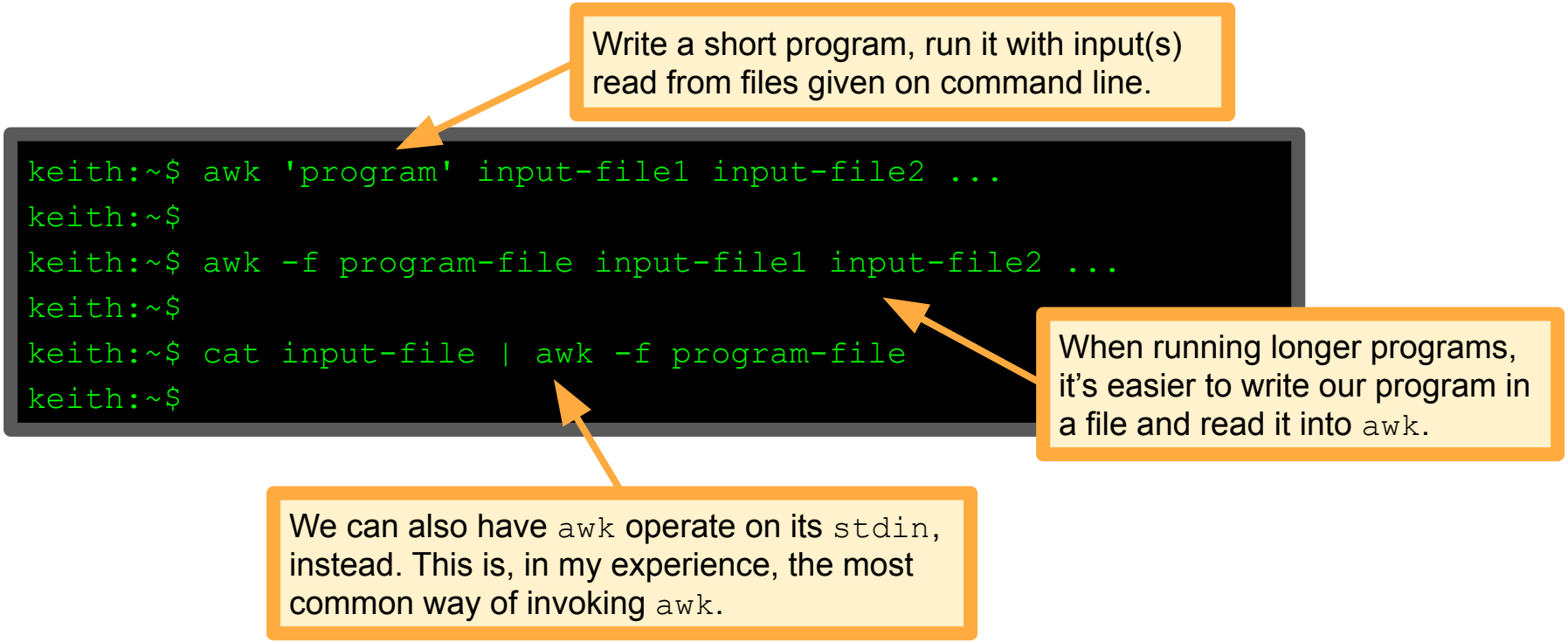
- `awk` reads its input one line at a time
- When input matches a pattern, perform its associated action

```
pattern { action }
pattern { action }
...
```

Written on separate lines, by convention, though this isn't required

Succinctly summarized by A. V. Aho (the A in AWK):
AWK reads the input a line at a time. A line is scanned for each pattern in the program, and for each pattern that matches, the associated action is executed.

Running awk on the command line



Our first `awk` programs

The `BEGIN` pattern tells `awk` to run this command before doing anything with its input (of which there is none).

```
keith:~$ awk 'BEGIN { print "Hello, world." }'
Hello, world.
keith:~$
keith:~$ echo "This is a string." | awk '{ print }'
This is a string.
keith:~$
```

A line with no condition will *always* be executed.

`awk` applies its (condition,action) pairs to every line of input. In this case, we are just printing every line of input that `awk` sees.

We've written the same program, but now it is stored in `print.awk`.

```
keith:~$ cat print.awk
{ print }
keith:~$ echo "dog cat goat bird" | awk -f print.awk
dog cat goat bird
keith:~$
```

Comments in `awk`

`#` is the comment character in `awk` (just like bash, R and Python).

```
keith:~$ cat commented_print.awk
# This program just prints its stdin.
# Not particularly interesting, I'd say.
{ print }
keith@:~/$ echo "dog cat goat bird" | awk -f commented_print.awk
dog cat goat bird
keith:~$ echo "words words words" | awk '{print} # This is a comment.'
words words words
keith:~$
```

`awk` built-in variables

`awk` breaks each line up into fields (i.e., columns), split on whitespace by default

`awk` has some built-in variables to refer to these fields, similar to bash scripts...

- `$0` : the entire current line
- `$1`, `$2`, `$3`, … : the **field variables**

...and also has some other useful variables (these **do not** require dollar signs):

- `NF` : the number of fields in the current line
- `NR` : the number of records read so far
- See documentation for a full list of built-in variables
  - or see https://www.gnu.org/software/gawk/manual/gawk.html

Example file: name, phone number, email, relation

```
keith:~$ cat mail-list.txt
Amelia        555-5553     amelia.zodiacusque@gmail.com  F
Anthony       555-3412     anthony.asserturo@hotmail.com   A
Becky         555-7685     becky.algebrarum@gmail.com    A
Bill          555-1675     bill.drowning@hotmail.com     A
Broderick     555-0542     broderick.aliquotiens@yahoo.com R
Camilla       555-2912     camilla.infusarum@skynet.be   R
Fabius        555-1234     fabius.undevicesimus@ucb.edu  F
Julie         555-6699     julie.perscrutabor@skeeve.com   F
Martin        555-6480     martin.codicibus@hotmail.com  A
Samuel        555-3430     samuel.lanceolis@shu.edu      A
Jean-Paul     555-2127     jeanpaul.campanorum@nyu.edu   R
keith:~$
```

A : acquaintance
F : friend
R : relative

Rules using regexes

We can create rules that apply only to lines matching a regex

```
keith:~$ awk '/\.edu/ { print $0 }' mail-list.txt
Fabius       555-1234     fabius.undevicesimus@ucb.edu  F
Samuel       555-3430     samuel.lanceolis@shu.edu      A
Jean-Paul    555-2127     jeanpaul.campanorum@nyu.edu   R
keith:~$ awk '/[[:space:]]F$/ { print $1, $3 }' mail-list.txt
Amelia amelia.zodiacusque@gmail.com
Fabius fabius.undevicesimus@ucb.edu
Julie julie.perscrutabor@skeeve.com
keith:~$
```

If a line contains the string `'.edu'`, print the whole line.

Print the name and email (fields 1 and 3) of friends. “friend” entries end with a capital F, so that’s what our regex looks for. The comma in the print statement is necessary to put a space between fields 1 and 3.

Comparison patterns

This pattern matches lines whose first field is longer than 6 characters

We didn't specify an action. The default is to print the whole line, like `print $0`.

```
keith:~$ cat mail-list.txt | awk 'length($1) > 6'
Anthony      555-3412     anthony.asserturo@hotmail.com   A
Broderick    555-0542     broderick.aliquotiens@yahoo.com R
Camilla      555-2912     camilla.infusarum@skynet.be     R
Jean-Paul    555-2127     jeanpaul.campanorum@nyu.edu     R
keith:~$ awk '{ if (length($1) > max) max = length($1) }; END { print max }'
mail-list.txt
9
keith:~$
```

This pattern finds the length of the longest name. Note that we did not have to declare the variable `max`.

The `END` pattern runs once we have reached the end of the input.

Multiple rules

Our awk program can include multiple rules. A line can match multiple rules, in which case it gets processed multiple times.

```
keith:~$ awk '/12/ { print $2 }; /21/ { print $2 }' mail-list.txt
555-3412
555-2912
555-1234
555-2127
555-2127
keith:~$
keith:~$ awk '/12/ && /21/ { print $2 }' mail-list.txt
555-2127
keith:~$
```

`2127` matches both `/12/` and `/21/`

`&&` is the AND operator. A line must match both of these regexes to match the pattern.

See https://www.gnu.org/software/gawk/manual/gawk.html#Boolean-Ops for more on Boolean operators.

What else?

`awk` is a kind of command-line swiss army knife

A non-exhaustive list of things we haven't discussed:

- For- and while-loops
- Importing variables from the shell into `awk`
- Defining functions in `awk`

The best place to learn more is

The GNU Awk User's Guide

https://www.gnu.org/software/gawk/manual/gawk.html

Also recommended:

sed & awk, 2nd Edition by D. Dougherty and A. Robbins. O'Reilly Media