

WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.

OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!



BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!

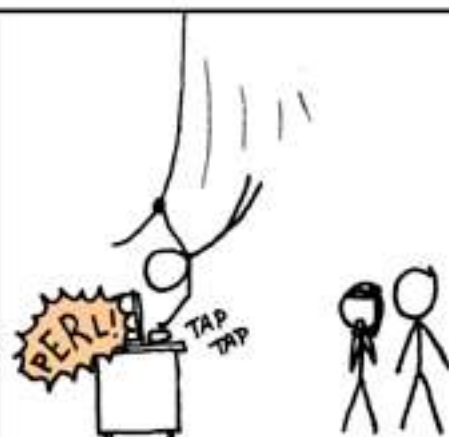


IT'S HOPELESS!

EVERYBODY STAND BACK.



I KNOW REGULAR EXPRESSIONS.



1. Regular expressions

2. sed

3. Editing Files

4. Shell loops

5. Shell scripts

Regular Expressions

Text often follows human conventions

Regular Expressions

Text often follows human conventions

Dates have a country-specific format:

Europe: day-month-year

US: month-day-year

Regular Expressions

Text often follows human conventions

Dates have a country-specific format:

Europe: day-month-year

US: month-day-year

Telephone numbers: xxx-xxx-xxxx

Regular Expressions

Text often follows human conventions

Dates have a country-specific format:

Europe: day-month-year

US: month-day-year

Telephone numbers: xxx-xxx-xxxx

Zip codes must be five digits long

Regular Expressions

Text often follows human conventions

Dates have a country-specific format:

Europe: day-month-year

US: month-day-year

Telephone numbers: xxx-xxx-xxxx

Zip codes must be five digits long

Structure: certain number of columns or rows

Regular Expressions

Text often follows human conventions

Dates have a country-specific format:

Europe: day-month-year

US: month-day-year

Telephone numbers: xxx-xxx-xxxx

Zip codes must be five digits long

Structure: certain number of columns or rows

Conventions can change: prior to 2000, dates were always 3 sets of two numbers

Regular Expressions, ctd.

If we can encode search context we can make much more powerful searches.

What sort of information would we like to specify?

Whether text is:

1. Composed of numbers
2. Composed of letter
3. A certain length
4. Full of spaces or tabs
5. First or last on the line

Regular Expressions, ctd.

Encoding	Modern Equivalent	Pattern Type
.		a single character
+		one or more characters
*		zero or more characters
?		Maybe present
^		first on the line
\$		last on the line
[0-9]	\d	digits
[a-zA-Z]	\w	letters
,	\s \t	space
{ 3 }		must be exactly 3 characters long
{ 3 , 5 }		between 3-5 characters long
[ACGT]		a specific set of characters (a class)

Regular Expressions, ctd.

Expression	Regular Expression
341341	
	[0-9]+
julian catchen	[a-z]+ [a-z]+
541-485-5128	[0-9]{3}\-[0-9]{3}\-[0-9]{4}
	[0-9\-.]+
June 3, 1978	[a-zA-Z]+ [0-9], [0-9]{4}
AGCCCCTAGGACTGAAATTCC	[ACGT]

Regular Expressions, ctd.

Expression	Regular Expression
341341	
	[0-9]+
julian catchen	[a-z]+ [a-z]+
541-485-5128	[0-9]{3}\-[0-9]{3}\-[0-9]{4}
	[0-9\-]+
June 3, 1978	[a-zA-Z]+ [0-9], [0-9]{4}
	[a-zA-Z]+ [0-9]+, [0-9]{4}
	[a-zA-Z]+ ?[0-9]?,[0-9]{4}

1. Download and decompress the file:

`http://creskolab.uoregon.edu/lund/record.tsv.gz`

2. Use grep and search the file for the different patterns above:

```
% grep -E "pattern" record.tsv
```

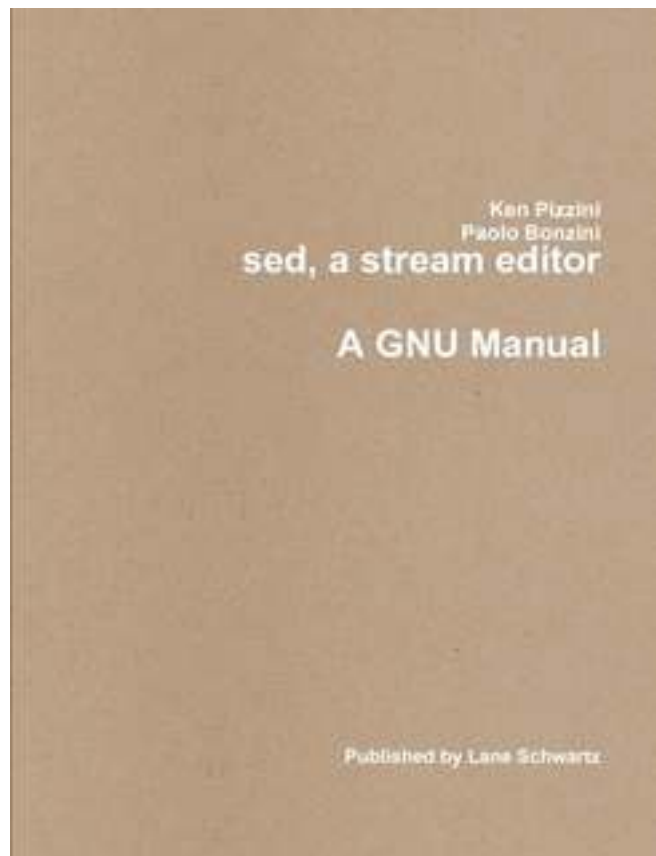
3. cat the file afterwards to examine what does and doesn't match with each pattern.

sed, a stream editor

Program 1

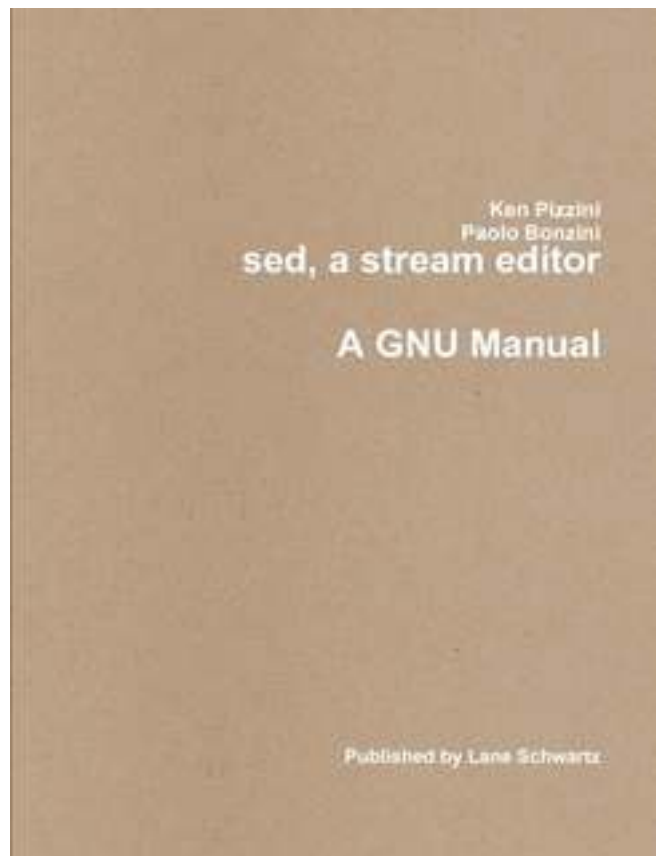
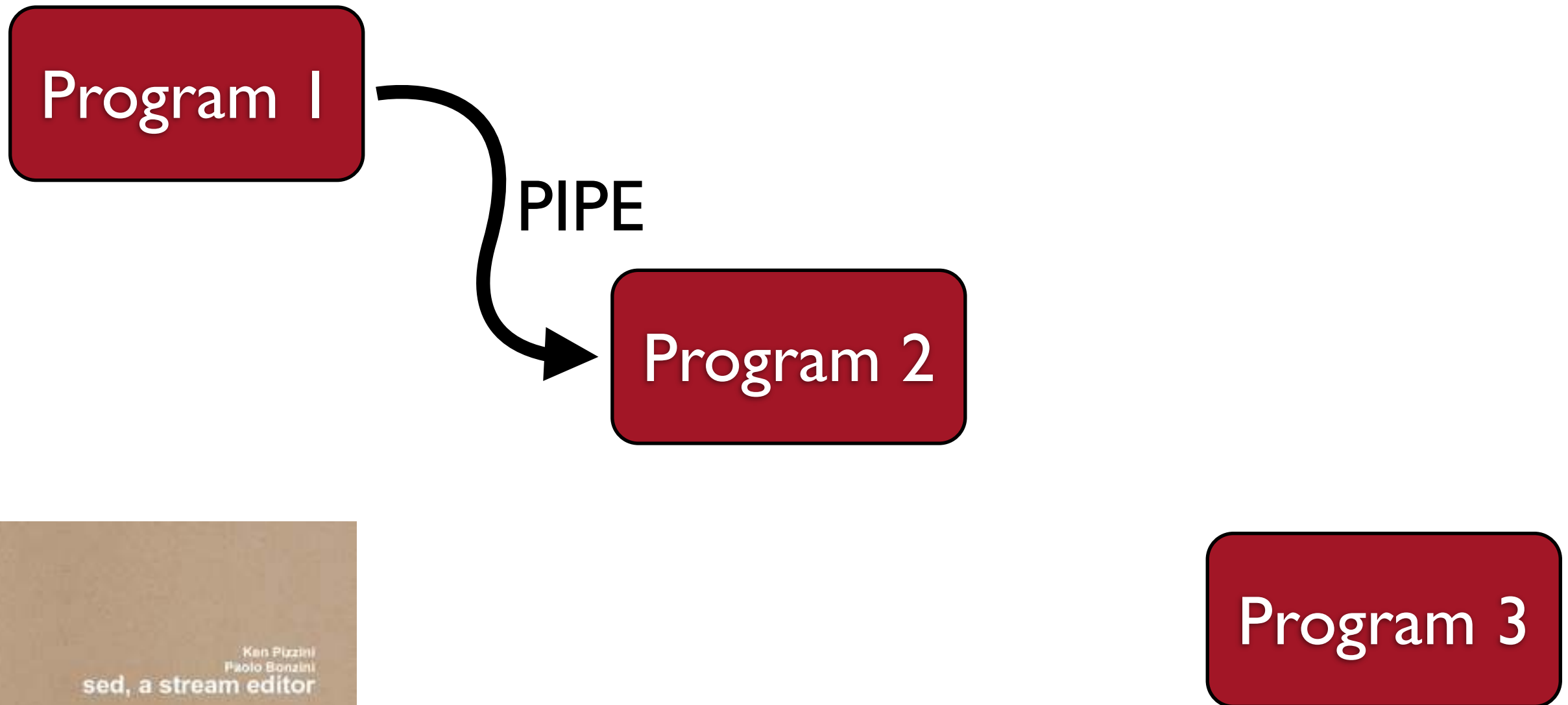
Program 2

Program 3



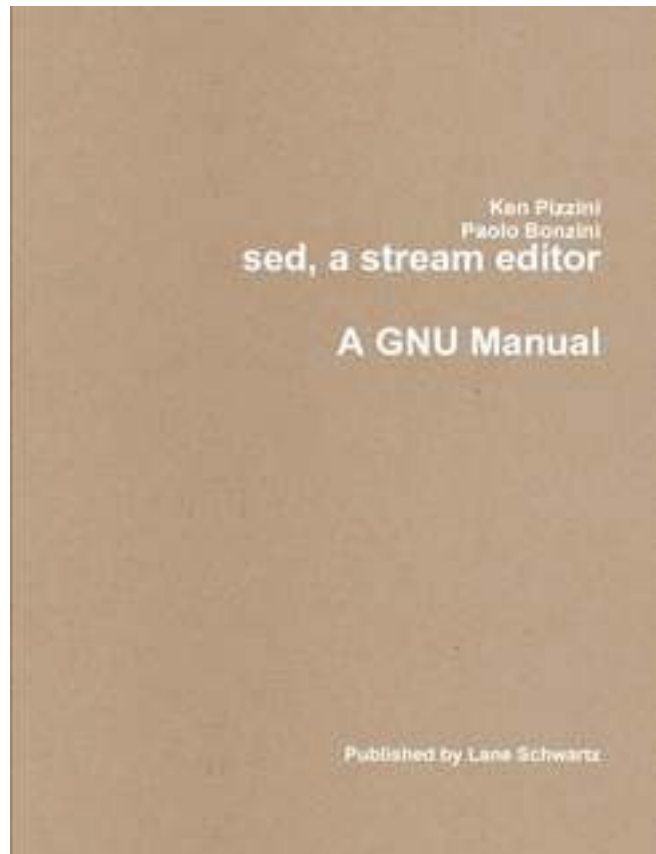
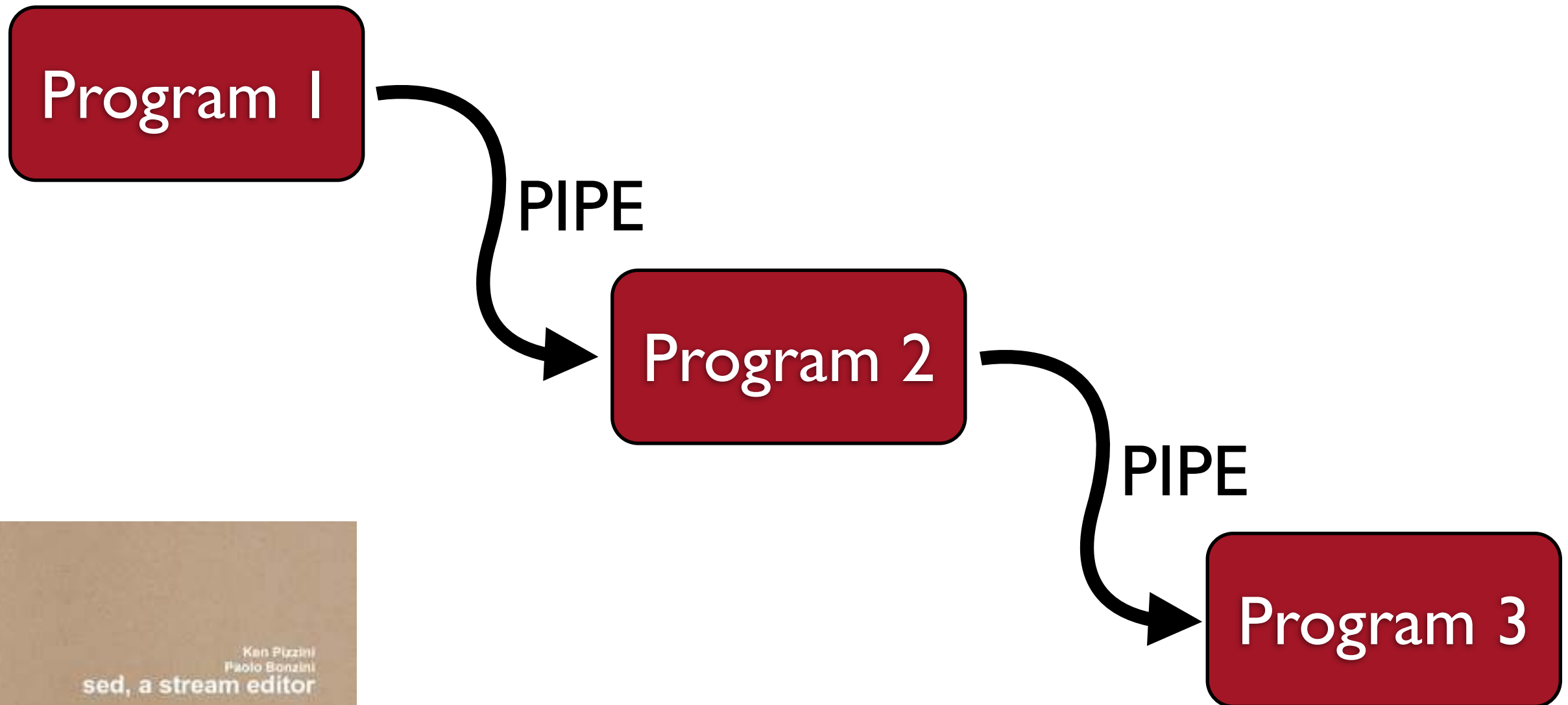
Search and replace on data flowing
through the pipe (a stream)

sed, a stream editor



Search and replace on data flowing
through the pipe (a stream)

sed, a stream editor



Search and replace on data flowing
through the pipe (a stream)

sed, a stream editor, ctd.

`s/pattern/replace/`

Two new encodings:

Record a match: `(pattern)`

Back references: `\1 \2 \3`

```
% sed -E 's/[a-z]+ [a-z]+/foo/'
```


sed, a stream editor, ctd.

`s/pattern/replace/`

```
% cat record.tsv | sed -E 's/[a-z]+ [a-z]+/foo/'  
  
% cat record.tsv | sed -E 's/([a-z]+) [a-z]+/\1/'  
  
% cat record.tsv | sed -E 's/[0-9]+//'  
  
% cat record.tsv | sed -E 's/[0-9]+//g'  
  
% cat record.tsv | sed -E 's/^[0-9]+ //'
```

sed, a stream editor, ctd.

`s/pattern/replace/`

Create a complex command:

```
% cd samples
% ls -l
fish_001.tags.tsv
fish_003.tags.tsv
fish_004.tags.tsv
fish_005.tags.tsv
```

```
% ls -l | sed -E 's/^(fish_[0-9]+\\.tags)\.tsv/mv \1\.tsv \1.loc/'
```

fish_310.tags.tsv.gz

http://creskolab.uoregon.edu/lund/fish_310.tags.tsv.gz

ls
gunzip
man
more
cat
wc
head
cut
grep
sed
tr
>
|

1. Decompress the file
2. Extract out the consensus sequences (2,191 sequences)
3. Extract out the ID and sequence for each consensus
4. Use sed to reverse the columns

`sed -E "s/^([0-9]+) ([ACGTN]+)/\2 \1/"`

5. Use sed/tr to convert to a FASTA file

`sed -E "s/^([0-9]+) ([ACGTN]+)/>\1|\2/"`

`use tr to replace the "|" character with a new line "\n"`

1. use ctrl-v tab to get a literal tab on the command line

Editing on UNIX



Emacs

Richard Stallman - 1976
Founded GNU Project



Vi

Bill Joy - 1976
BSD/Sun Microsystems

Download example file using wget

Decompress the file in your home directory.

```
http://creskolab.uoregon.edu/lund/manifesto.gz
```

```
% emacs <filename>
```

```
% emacs /absolute/path/to/file
```

```
% emacs ../../../relative/path/to/file
```

```
% emacs file1 /path/file2 ../../file3
```

Command mode versus **Text-entry mode**

Your mouse cannot help you! (mostly.)

Emacs commands start with either **ctrl** or **meta** (esc-x)

The **Mark**

1. No mouse, so we need a way to indicate/highlight regions of text.
2. `ctrl-space` sets the mark
3. Use the arrow keys to highlight a region of text
4. Issue a command, e.g. copy/paste, or just press the space bar to unhighlight

Useful Emacs commands

<code>ctrl-x ctrl-s</code>	save file
<code>ctrl-x ctrl-f</code>	open a new file
<code>ctrl-space</code>	set the mark
<code>esc w</code>	copy highlighted text
<code>ctrl-w</code>	cut highlighted text
<code>ctrl-y</code>	paste text
<code>ctrl-x u</code>	undo
<code>ctrl-x b</code>	switch to another file (buffer)
<code>ctrl-s</code>	search for text
<code>esc %</code>	search and replace text
<code>ctrl-]</code>	quit current command
<code>ctrl-v</code>	page-down
<code>esc v</code>	page-up
<code>esc g g</code>	goto line
<code>ctrl-x ctrl-c</code>	quit emacs

manifesto.gz

1. Start emacs, opening `manifesto`
2. Copy the title and paste it one line below.
3. Search for the term 'GNU', how many instances can you find?
4. Search/replace the phrase 'free software' with 'proprietary software'. How many instances did you replace?
5. Now, undo the replacements so that we remain free
6. Cut out the first paragraph of text.
7. Open a new file, `manifesto2`, paste the paragraph, save, quit emacs, view the file with `more`

One “dumb” question.

Download the example file using wget

1. Visit in your web browser:

`http://creskolab.uoregon.edu/stacks/`

2. Right click on the “Download Stacks” link and select
“Copy Link Location” (or a similar variant)

3. Paste the link into the terminal and use `wget` to fetch it.
Untar and decompress the archive.

File Permissions, Users+Groups



File Permissions, Users+Groups

`rwXrwxr-x`
`11111101`



File Permissions, Users+Groups

`rwXrwXr-x`

`111111101`

Owner	Group	Other
-------	-------	-------

<code>rwX</code>	<code>rwX</code>	<code>r-x</code>
------------------	------------------	------------------

<code>111</code>	<code>111</code>	<code>101</code>
------------------	------------------	------------------



File Permissions, Users+Groups

`rwXrwxr-x`
`11111101`

Owner	Group	Other
<code>rwX</code>	<code>rwX</code>	<code>r-x</code>
<code>111</code>	<code>111</code>	<code>101</code>
<code>7</code>	<code>7</code>	<code>5</code>



File Permissions, Users+Groups

`rwXrwxr-x`
`11111101`

Owner	Group	Other
<code>rwX</code>	<code>rwX</code>	<code>r-x</code>
<code>111</code>	<code>111</code>	<code>101</code>
<code>7</code>	<code>7</code>	<code>5</code>

`chmod 777 README`

`chmod 644 README`

`chmod 600 README`

What is a variable?

What is a variable?



What is a variable?



What is a variable?



What is a variable, ctd.

I. A variable can have almost any name:

- foo
- bar
- cats
- line

What is a variable, ctd.

1. A variable can have almost any name:

- `foo`
- `bar`
- `cats`
- `line`

2. You assign it a value like this:

- `foo=32`
- `bar=27.4938193`
- `cats="LOLzzzzz"`
- `line="sample_01.fq"`

What is a variable, ctd.

1. A variable can have almost any name:

- `foo`
- `bar`
- `cats`
- `line`

2. You assign it a value like this:

- `foo=32`
- `bar=27.4938193`
- `cats="LOLzzzz"`
- `line="sample_01.fq"`

3. You refer to it using a dollar sign:

- `$foo`
- `${foo}`

What is a variable, ctd.

Variables often have types, depending on the language

integer	<code>1, 2, 3, 4, 5...</code> <code>-1, -2, -3, -4, -5...</code>
float (double)	<code>3.141592653589790</code>
string (of characters)	<code>"My dog is Billy"</code>

What is a variable, ctd.

1. Try it out: set a variable on the command line:

- `foo=32`

2. Display the value the variable holds:

- `echo $foo`

3. Set a new value and display it:

- `foo="The cat likes thai food"`
- `echo $foo`

Shell Loops

Shell Loops

My favorite command: `ls -1`

Shell Loops

My favorite command: `ls -l`

```
while read line; do command $line; done
```

Shell Loops

My favorite command: `ls -l`

```
while read line; do command $line; done
```

Pipe `ls -l` to a `while` loop and watch magic happen

samples.tar.gz

1. Expand the archive: **tar -xvf**

```
fish_001.tags.tsv  
fish_003.tags.tsv  
fish_004.tags.tsv  
fish_005.tags.tsv
```

2. Move into the samples directory

3. Execute a command that can identify the consensus sequences in this file.

4. Try out the `ls -l` command

5. Combine parts 3 and 4 with a while loop to count the number of consensus sequences in each file

```
4  
5  
4  
6
```

samples.tar.gz

1. Expand the archive: **tar -xvf**

```
fish_001.tags.tsv  
fish_003.tags.tsv  
fish_004.tags.tsv  
fish_005.tags.tsv
```

2. Move into the samples directory

3. Execute a command that can identify the consensus sequences in this file.

4. Try out the **ls -l** command

5. Combine parts 3 and 4 with a while loop to count the number of consensus sequences in each file

```
4  
5  
4  
6
```

while read line; do command \$line; done

Shell Scripts

1. Anything you can do on the shell can be placed in a shell script
2. Shell scripts often end in the suffix “.sh”
3. Shell scripts must be executable (chmod 755)
4. *Comments* can be written in scripts with a “#”
5. Variables can be used to shorten long paths
6. Shell loops can be used to process lots of files
7. “\” can be used to wrap long commands across multiple lines
8. #!/bin/bash must be the first line - specifies interpreter

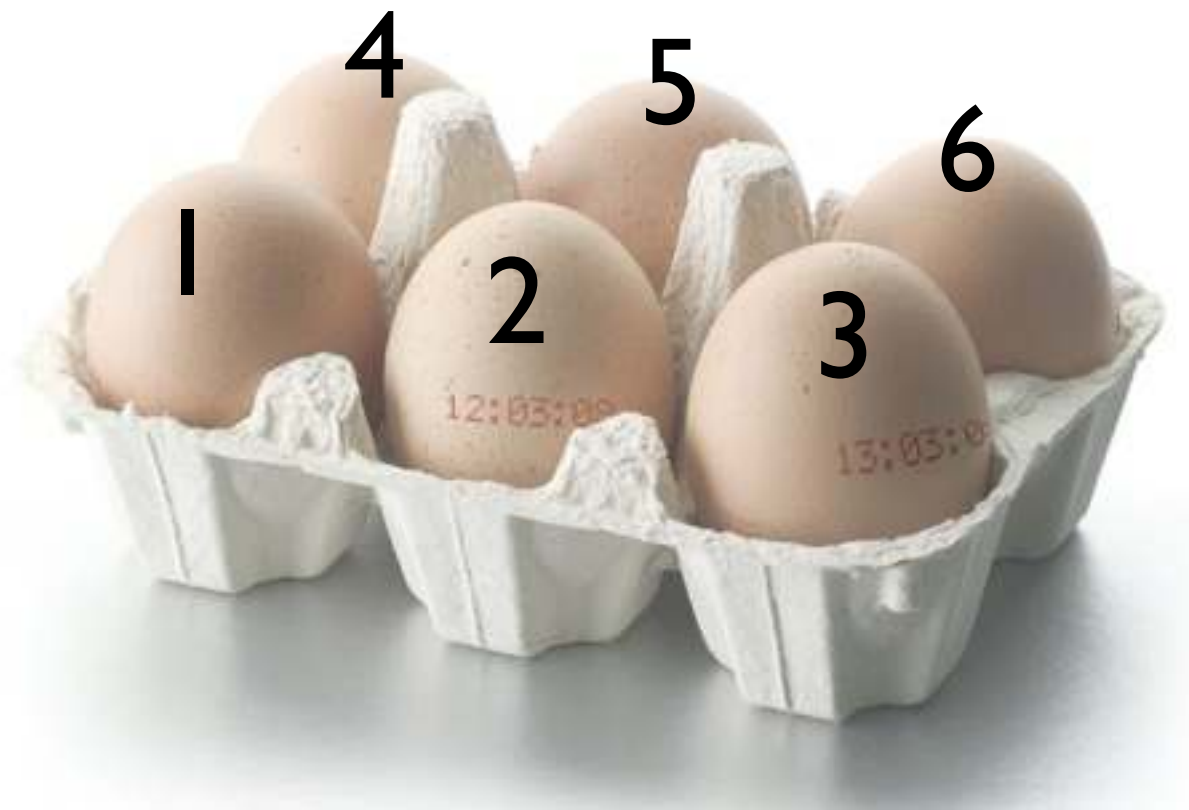
Shell Scripts, ctd.

What is an array?



Shell Scripts, ctd.

What is an array?



Shell Scripts, ctd.

What is an array?

I. A variable can have almost any name:

- foo
- bar
- cats
- line

Shell Scripts, ctd.

What is an array?

1. A variable can have almost any name:

- `foo`
- `bar`
- `cats`
- `line`

2. You assign it a value like this:

- ```
foo="progeny_10.fa
progeny_11.fa
progeny_12.fa
progeny_13.fa
progeny_14.fa
progeny_15.fa"
```

# Shell Scripts, ctd.

```
[catchen@genome]:~/research/seq/dre_hap% more build_tags.sh
#!/bin/bash
```

```
ROOT=$HOME/research/seq
bin=$ROOT/radtags/stacks/trunk/
src=$ROOT/dre_hap
db=dre_hap_radtags
batch_id=1
date=2011-05-17
desc="Danio rerio haploid map"
```

```
cd $bin
```

```
nice -n 19 $bin/scripts/denovo_map.pl -m 3 -B $db -b $batch_id -t -a $date -D "$desc" -e $bin -T 40 \
-o $src/nstacks \
-p $src/samples/female.fq \
-r $src/samples/progeny_01.fq \
-r $src/samples/progeny_02.fq \
-r $src/samples/progeny_03.fq \
-r $src/samples/progeny_05.fq \
-r $src/samples/progeny_06.fq \
-r $src/samples/progeny_08.fq \
-r $src/samples/progeny_09.fq \
-r $src/samples/progeny_10.fq \
-r $src/samples/progeny_13.fq \
-r $src/samples/progeny_14.fq \
-r $src/samples/progeny_16.fq \
-r $src/samples/progeny_17.fq \
-r $src/samples/progeny_18.fq \
-r $src/samples/progeny_19.fq \
-r $src/samples/progeny_20.fq \
-r $src/samples/progeny_23.fq \
-r $src/samples/progeny_24.fq \
-r $src/samples/progeny_25.fq \
-r $src/samples/progeny_27.fq \
-r $src/samples/progeny_33.fq \
-r $src/samples/progeny_34.fq \
-r $src/samples/progeny_35.fq \
-r $src/samples/progeny_36.fq \
-r $src/samples/progeny_37.fq \
-r $src/samples/progeny_38.fq
```

# Shell Scripts, ctd.



# Advanced Bash-Scripting Guide

<http://tldp.org/LDP/abs/html/>



# Download example file using wget

Untar the file in your home directory.

<http://creskolab.uoregon.edu/lund/seqs.tar.gz>



# Shell Scripts, ctd.

`http://creskolab.uoregon.edu/lund/seqs.tar.gz`

1. Move into the seqs directory
2. Use `ls -l` and a shell loop to count the number of lines in each file
3. Use a shell loop to concatenate the files into a single file, use the `>>` operator to redirect the output
4. Count the lines in the final file, make sure they match
5. Create a shell script to do all of the above.