One-Liners

Aims

- Know what a perl one-liner is
- Know how to use one
- Know when they are useful

What Are One-Liners?

- Perl one-liners should be mentioned in an introductory course.
- Now, depending on how you look at it, these are
  - the most useful things possible,
  - or the worst idea ever imagined.
- They are exactly what they say on the tin. A perl script, in one line, on the command line.

Our first One-Liner

- Just try typing the following on the command line:

  >perl -le 'print q{Hello World}'

  Hello World

  >
- Let's take a look at it

Our first One-Liner

perl

- The first argument of any command line is the executable you want to run. In this case, we'll run the perl interpreter.
- This is equivalent to the shebang line telling the os to run perl
- This will use the first perl on our path

Our first One-Liner

-le

- This is actually two options
  - -l => all print statements should automatically have a newline appended to them
  - -e => tells the perl executable to compile and execute the next item (within quotes)

Our first One-Liner

'print q{Hello World}'

- The perl 'script' we want to compile and execute, in this case the statement *print q{Hello World}*
- Note: We must use the q{} form of quoting, as the next ' seen will close the statement we would want to execute

Extending the One-Liner

- You can put more in the one liner. We can use modules, and do multiple statements, including loops

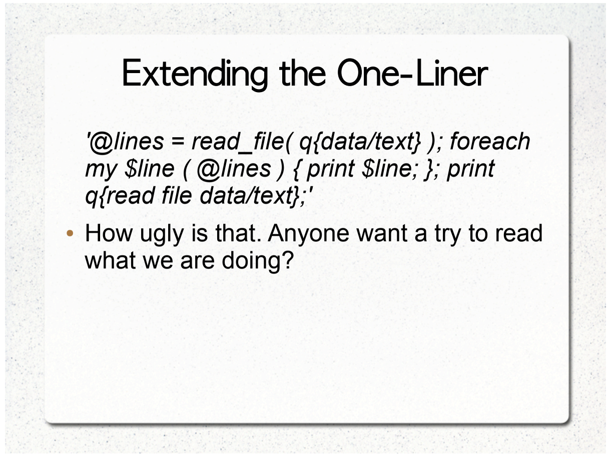
  >perl -MFile::Slurp -le '@lines = read_file(q{data/text}); foreach my $line (@lines) { print $line; }; print q{read file data/text};'
- Lets look at the new parts

Extending the One-Liner

-MFile::Slurp

- This is equivalent to the *use* statement. We can have as many of these as we need

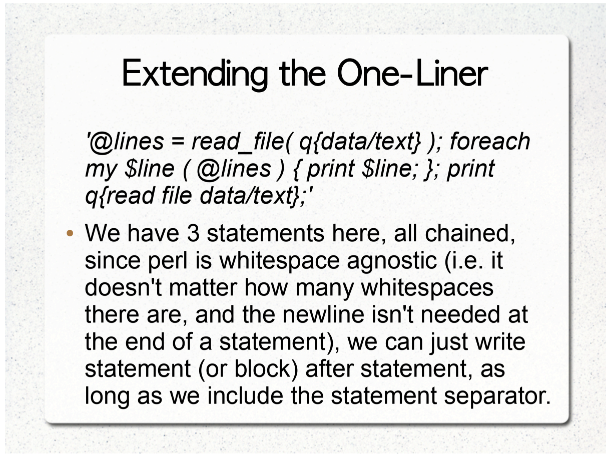
  -MFile::Slurp -MModern:Perl -MTest::More...



Extending the One-Liner

'@lines = read_file(q{data/text}); foreach my $line (@lines) { print $line; }; print q{read file data/text};'

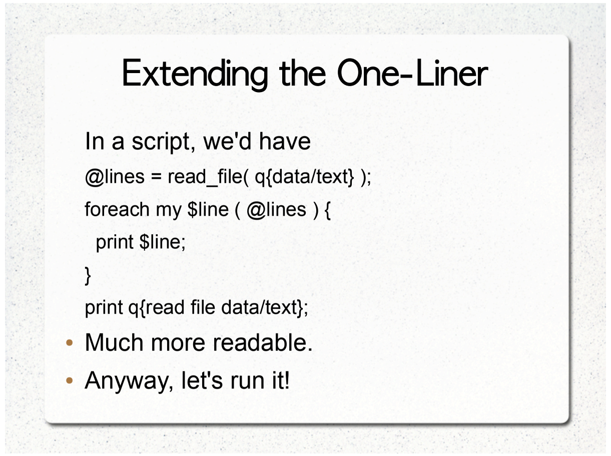
- How ugly is that. Anyone want a try to read what we are doing?



Extending the One-Liner

'@lines = read_file(q{data/text}); foreach my $line (@lines) { print $line; }; print q{read file data/text};'

- We have 3 statements here, all chained, since perl is whitespace agnostic (i.e. it doesn't matter how many whitespaces there are, and the newline isn't needed at the end of a statement), we can just write statement (or block) after statement, as long as we include the statement separator.

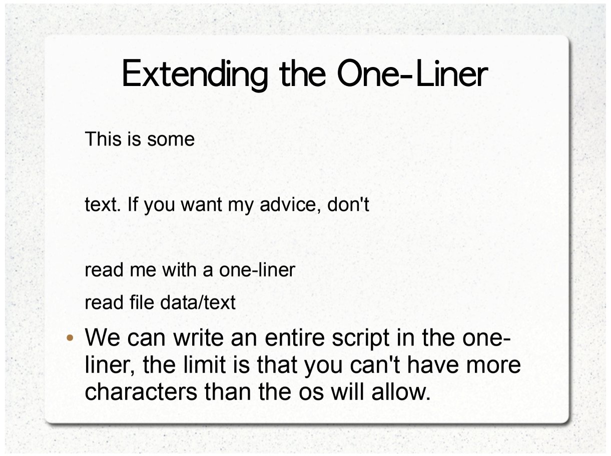


Extending the One-Liner

In a script, we'd have

```
@lines = read_file( q{data/text} );
foreach my $line ( @lines ) {
  print $line;
}
print q{read file data/text};
```

- Much more readable.
- Anyway, let's run it!



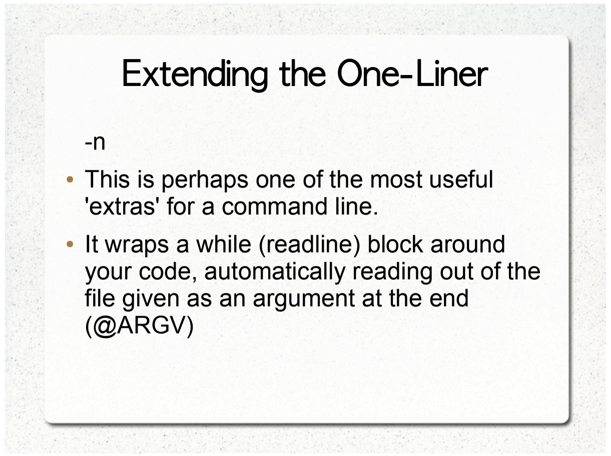
Extending the One-Liner

```
This is some


text. If you want my advice, don't


read me with a one-liner
read file data/text
```

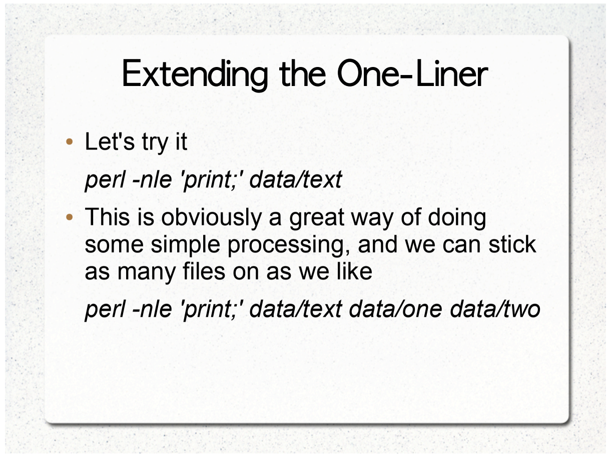
- We can write an entire script in the one-liner, the limit is that you can't have more characters than the os will allow.



Extending the One-Liner

-n

- This is perhaps one of the most useful 'extras' for a command line.
- It wraps a while (readline) block around your code, automatically reading out of the file given as an argument at the end (@ARGV)



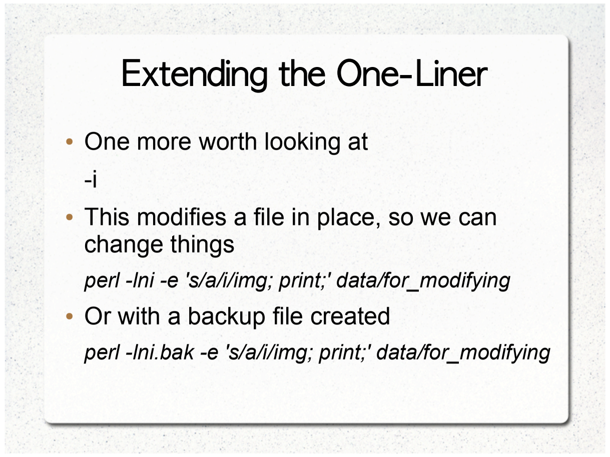
Extending the One-Liner

- Let's try it

perl -nle 'print;' data/text

- This is obviously a great way of doing some simple processing, and we can stick as many files on as we like

perl -nle 'print;' data/text data/one data/two



Extending the One-Liner

- One more worth looking at

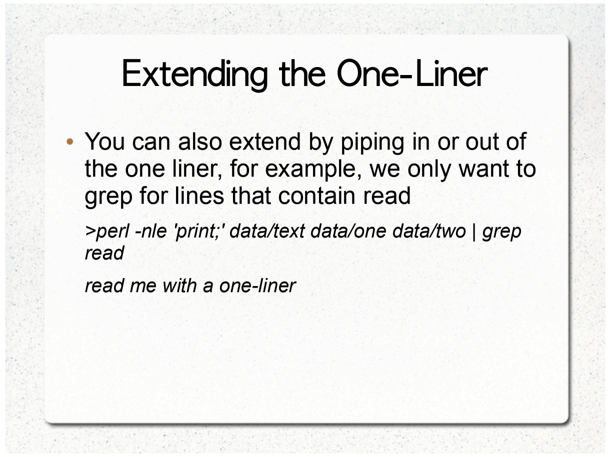
-i

- This modifies a file in place, so we can change things

perl -lni -e 's/a/i/img; print;' data/for_modifying

- Or with a backup file created

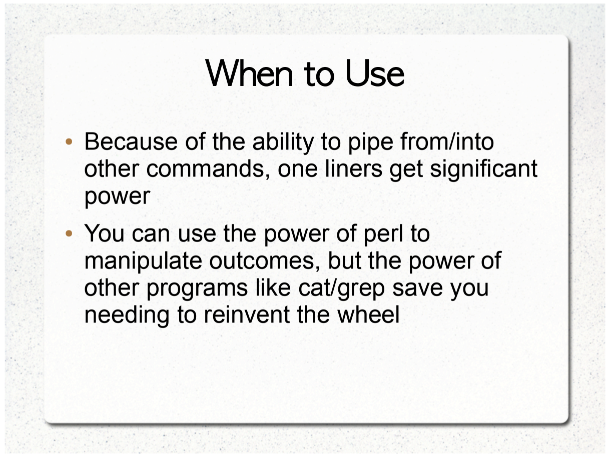
perl -lni.bak -e 's/a/i/img; print;' data/for_modifying



Extending the One-Liner

- You can also extend by piping in or out of the one liner, for example, we only want to grep for lines that contain read

```
>perl -nle 'print;' data/text data/one data/two | grep read
read me with a one-liner
```



When to Use

- Because of the ability to pipe from/into other commands, one liners get significant power
- You can use the power of perl to manipulate outcomes, but the power of other programs like cat/grep save you needing to reinvent the wheel

When to Use

- Lets look for something which could find all files with a size less than 72 bytes in the data

```
ls -l  data/ | grep -v total | perl -le '@lines = <STDIN>;
@capture; foreach $file_info (@lines) { @temp = split
/\s+/xms, $file_info; if ( $temp[4] < 72 ) { push
@capture, $temp[-1]; } }; print join "\n", @capture;'
```

- pipes into the perl gives access to the data via STDIN
- So only use when the power of perl helps you find something

When Not to Use

- Because you have the full power of perl, it can be tempting to just write a one-liner to solve a problem.
- I mean, if
  - you are only going to want to solve this once,
  - or if you just want to see if something is stuck,

then surely it's not necessary to write a script.

When Not to Use

Wrong!

When Not to Use

- If you are writing more than 3 statements in a one-liner, think about why you are writing this.
- If you are writing the one-liner to tell you something about a pipeline you are running, don't you think you'll want to run it tomorrow, or next week?

When Not to Use

- If you expect to run it more than once, you should think about writing a script:
  - 1) Only call the script name, even if you do put it in pipes
  - 2) You can write some tests for it
  - 3) Someone will ask you for it
  - 4) It will become part of a project
  - 5) You can start to refactor and capture the code

How do I, Andy, Use Them

- You can probably tell I'm not overly keen on one-liners.
- I use them solely for testing out something I can't remember what will happen.

How do I, Andy, Use Them

- i.e. If I want check if I *print* the array or scalar with *print @array*
- My code I want to write is

```
...time consuming code which generates an array...
print @array;
```

- I don't want to run my script to check what I want it to print, but I can't remember if I need the scalar keyword to display the number of elements with print.

How do I, Andy, Use Them

- So, on the command line, I quickly write

```
>perl -le '@array = qw{hello goodbye smith jones};
print @array'
hellogoodbyesmithjones
```

- So print just concatenates all the array together and prints it, so I quickly tap up, and add in scalar

```
>perl -le '@array = qw{hello goodbye smith jones};
print scalar @array'
4
```

How do I, Andy, Use Them

- So, I know that in order to print the number of elements, I need to remember to explicitly write in scalar
- Job done. I can move on with my code. Everything else I do, I write in a script/module/object and test! I know I am going to want to do it more than once.
- Even if I do only end up using the script once, it is practice.