Perl for linguists

- Why programming?
Perl for linguists

- Why programming? Why Perl?
Perl for linguists

- Why programming? Why Perl?
- Learn a little Perl.
Perl for linguists

- Why programming? Why Perl?
- Learn a little Perl.
- More advanced Perl...
Why programming?
Why programming?

- Collect data
Why programming?

- Collect data
- Analyze data
Why programming?

- Collect data
- Analyze data
- Model theory
Why programming?

- Collect data
- Analyze data
- Model theory
- General professional skills
Code for this tutorial

All the code for this presentation is available over the web at the following URL:

http://linguistics.arizona.edu/~hammond/berkeley.html
Unzipping the programs

1. Download the file `programfiles.zip` by right-clicking on the link, selecting ‘save target as’, and selecting your desktop as the destination.

2. Unzip the file by double-clicking on the downloaded file on your desktop and moving all the files there to a new directory on the desktop.

3. Open the MS-DOS prompt in the new directory by double-clicking on the file `doswindow.bat`. 
Collecting data
Collecting data

- Running experiments locally (`expprog.pl`)
Collecting data

- Running experiments locally \((\text{expprog.pl})\)
- Running experiments locally with a GUI \((\text{tkexp.pl})\)
Collecting data

- Running experiments locally (*expprog.pl*)
- Running experiments locally with a GUI (*tkexp.pl*)
- Running experiments remotely (*Bailey & Hahn replication, bhrep.cgi*)
Collecting data

- Running experiments locally (`expprog.pl`)
- Running experiments locally with a GUI (`tkexp.pl`)
- Running experiments remotely (`Bailey & Hahn replication`, `bhrep.cgi`)
- Assembling corpora from local static resources (`makecorpus.pl`)

Perl for Linguists – p.6/38
Collecting data

- Running experiments locally (*expprog.pl*)
- Running experiments locally with a GUI (*tkexp.pl*)
- Running experiments remotely (*Bailey & Hahn replication, bhrep.cgi*)
- Assembling corpora from local static resources (*makecorpus.pl*)
- Assembling corpora from nonlocal dynamic resources: the web (*websearch.pl*)
Analyzing data
Analyzing data

- Looking for patterns (`visgrep.pl`)
Analyzing data

- Looking for patterns (`visgrep.pl`)
- Counting things (`neightk.pl`)

Perl for Linguists – p.7/38
Analyzing data

- Looking for patterns (`visgrep.pl`)
- Counting things (`neightk.pl`)
- Finding verbs (`verbs.pl`)
Modeling theory
Modeling theory

- Optimality Theory ([web interface](http://sylpars.pl), sylpars.pl)
Modeling theory

- Optimality Theory ([web interface](https://sylpars.pl), sylpars.pl)
- N-gram models (a bunch of [examples](https://sylpars.pl) from a course on Statistical NLP that I did recently)
General professional skills
General professional skills

- General programming skills
General professional skills

- General programming skills
- Web programming
Why Perl?
Why Perl?

• Free
Why Perl?

- Free
- Multi-platform
Why Perl?

- Free
- Multi-platform
- Easy
Why Perl?

- Free
- Multi-platform
- Easy
- Multiple dialects
Why Perl?

- Free
- Multi-platform
- Easy
- Multiple dialects
- Powerful regular expression tools
Why Perl?

- Free
- Multi-platform
- Easy
- Multiple dialects
- Powerful regular expression tools
- Written by a "linguist"
Why Perl?

- Free
- Multi-platform
- Easy
- Multiple dialects
- Powerful regular expression tools
- Written by a "linguist"
- Perl poetry
Why Perl?

- Free
- Multi-platform
- Easy
- Multiple dialects
- Powerful regular expression tools
- Written by a "linguist"
- Perl poetry
- Obfuscated perl, "japhs", etc.
Learn a little perl. . .
Learn a little perl. . .

- Windows basics
Learn a little perl. . .

- Windows basics
- Perl syntax
Learn a little perl. . .

- Windows basics
- Perl syntax
- IO
Learn a little perl... 

- Windows basics
- Perl syntax
- IO
- Regular expressions
Learn a little perl... 

- Windows basics
- Perl syntax
- IO
- Regular expressions
- Where to find out more
Windows basics
Windows basics

- It is easiest to invoke perl programs from the DOS prompt. (If configured properly, you can just double-click them though.)
Windows basics

- It is easiest to invoke perl programs from the DOS prompt. (If configured properly, you can just double-click them though.)

- `perl myprogram.pl`: invokes the perl interpreter with a program `myprogram.pl`. 
Windows basics

• It is easiest to invoke perl programs from the DOS prompt. (If configured properly, you can just double-click them though.)

• `perl myprogram.pl`: invokes the perl interpreter with a program `myprogram.pl`.

• `ctrl-c`: stops the current program
A goal

- We will build our efforts around an example program: a program that parses a text file in English into sentences and then *tries* to find all the verbs (*verbs.pl*).
- This exemplifies simple data and control structures and what we might call “computational linguistic reasoning”.
- *We won’t* get to the full version of this program—it’s infinitely expandable actually—but we can get a good start.
Programming overview
Programming overview

- Implement some idea as perl code (write the program).
Programming overview

- Implement some idea as perl code (write the program).
- Convert code into something the computer can execute (run `perl` on your code).
Programming overview

- Implement some idea as perl code (write the program).
- Convert code into something the computer can execute (run `perl` on your code).
- Program execution (run `perl` on your code).
Programming overview

• Implement some idea as perl code (write the program).
• Convert code into something the computer can execute (run `perl` on your code).
• Program execution (run `perl` on your code).
• Results (what happens as a consequence).
Perl syntax
Perl syntax

- A perl program is a series of statements.
Perl syntax

- A perl program is a series of *statements*.
- Statements can be organized into *groups*. 
Perl syntax

- A perl program is a series of statements.
- Statements can be organized into groups.
- Statements are operations on some bit of data, e.g. “print this string”, “add these numbers”, etc.
Perl syntax

• A perl program is a series of *statements*.
• Statements can be organized into *groups*.
• Statements are operations on some bit of data, e.g. “print this string”, “add these numbers”, etc.
• Groups of statements can apply:
  1. only when specific conditions hold, or
  2. more than once, etc.
Statements
Statements

- A statement includes some operation (typically marked with following parentheses), ends with a semicolon, and may contain more. For example:
Statements

- A statement includes some operation (typically marked with following parentheses), ends with a semicolon, and may contain more. For example:

  ```perl
  print("Hello!");
  ``` (prints the string “Hello!”)
Statements

- A statement includes some operation (typically marked with following parentheses), ends with a semicolon, and may contain more. For example:
  - `print("Hello!");` (prints the string “Hello!”)
  - `rand(5);` (gets a random number between 0 and 5)
Statements

- A statement includes some operation (typically marked with following parentheses), ends with a semicolon, and may contain more. For example:

- `print("Hello!");` (prints the string “Hello!”)
- `rand(5);` (gets a random number between 0 and 5)
- `localtime();` (gets the current time)
Try it
Try it

1. Open the DOS window (through the start menu or by clicking on the doswindow.bat icon.)
Try it

1. Open the DOS window (through the start menu or by clicking on the doswindow.bat icon.)

2. Type `edit myprogram.pl`
Try it

1. Open the DOS window (through the start menu or by clicking on the doswindow.bat icon.)
2. Type `edit myprogram.pl`
3. Type `print("I am a linguist!");`
Try it

1. Open the DOS window (through the start menu or by clicking on the doswindow.bat icon.)
2. Type `edit myprogram.pl`
3. Type `print("I am a linguist!");`
4. Select `Save` and then `Exit` from the `File` menu.
Try it

1. Open the DOS window (through the start menu or by clicking on the doswindow.bat icon.)

2. Type `edit myprogram.pl`

3. Type `print("I am a linguist!");`

4. Select Save and then Exit from the File menu.

5. Type `perl myprogram.pl` at the prompt.
Variables
Variables

• Notice how `rand()` and `localtime()` don’t seem to do anything.
Variables

- Notice how `rand()` and `localtime()` don’t seem to do anything.
- What they do is return a string. You can save that result and then print it:
Variables

- Notice how `rand()` and `localtime()` don’t seem to do anything.
- What they do is return a string. You can save that result and *then* print it:

```perl
$myvariable = localtime();
print($myvariable);
```
Variables
Variables

• Simple “scalar” variables: $myvar, $aVariable, $x.
Variables

- Simple “scalar” variables: $myvar, $aVariable, $x.

- A value is assigned to a variable with the assignment operator =.
Variables

- Simple “scalar” variables: $myvar, $aVariable, $x.

- A value is assigned to a variable with the assignment operator =.

- Variables can hold numbers, strings, etc.
Reading a file
Reading a file

• First use `open()` to open a file and associate it with a *handle*, e.g.
  
  ```perl
  open(F, "myfile.txt");
  ```
Reading a file

- First use `open()` to open a file and associate it with a *handle*, e.g.: `open(F, "myfile.txt");`.
- Read a line from the file with record-reading operator: `<F>`. 
Reading a file

- First use `open()` to open a file and associate it with a `handle`, e.g.
  ```perl
  open(F, "myfile.txt");
  ```
- Read a line from the file with record-reading operator: `<F>`.
- When you’re done, close the handle: `close(F);`.
Sample code

```perl
open (F,"myfile.txt");
$line = <F>;
print ($line);
close (F);
```
Control structures

- **if**: conditional application
- **while**: iteration as long as some condition is true
- **for**: iteration for a specific number of times
- **foreach**: iteration for every member of a list
More sample code

```perl
open(F,"myfile.txt");
while ($line = <F>) {
    print($line);
}
close(F);
```
Arrays
Arrays

- Array variables: a list of variables grouped together: @myarray, @thisBigArray, @them.
Arrays

- Array variables: a list of variables grouped together: @myarray, @thisBigArray, @them.
- Adding an element to the end of an array: push(@myarray, "hat");.
Arrays

- Array variables: a list of variables grouped together: `@myarray`, `@thisBigArray`, `@them`.
- Adding an element to the end of an array: `push(@myarray, "hat");`.
- Retrieving an element from the end of an array: `$it = pop(@myarray);`. 
Arrays

- Array variables: a list of variables grouped together: `@myarray`, `@thisBigArray`, `@them`.
- Adding an element to the end of an array: `push(@myarray, "hat");`.
- Retrieving an element from the end of an array: `$it = pop(@myarray);`.
- Retrieving a specific element: `print($myarray[4]);` (array indices start at 0).
Sample code again

```perl
open (F,"myfile.txt");
while ($line = <F>) {
    push (@mylines, $line);
}
close (F);
foreach $theline (@mylines) {
    print ($theline);
}
```
Regular expressions
Regular expressions

• “Regular expression”: a restrictive way of indicating string patterns.
Regular expressions

• “Regular expression”: a restrictive way of indicating string patterns.
• $\text{myvar} \sim /\text{regexp}/: \text{does } \text{myvar} \text{ contain the regular expression?}
Regular expressions

- “Regular expression”: a restrictive way of indicating string patterns.
- `$myvar =~ /regexp/`: does $myvar contain the regular expression?
- `$myvar =~ s/regexp/string/:` if $myvar contains the regular expression, replace it with the string.
The verb-finding program
The verb-finding program

- Recall that the program breaks an English text file into sentences and does its best at finding all the verbs.
The verb-finding program

- Recall that the program breaks an English text file into sentences and does its best at finding all the verbs.
- An hour is *not* enough time to figure out how to do all of this, but just the bits we’ve covered are a lot of it (*verbs.pl*).
Where to find out more
Where to find out more

- The free ActiveState Perl we’re using comes with extensive web-based documentation.
Where to find out more

- The free ActiveState Perl we’re using comes with extensive web-based documentation.
- In any perl implementation the `perldoc` command can be used to find out lots and lots of stuff.
Where to find out more

- The free ActiveState Perl we’re using comes with extensive web-based documentation.
- In any perl implementation the `perldoc` command can be used to find out lots and lots of stuff.
- The official and best perl website is [www.cpan.org](http://www.cpan.org), but see also [www.perl.org](http://www.perl.org).
Advanced stuff
Advanced stuff

• Let’s now do some more advanced stuff.
Advanced stuff

- Let’s now do some more advanced stuff.
- ‘Advanced’ in terms of perl.
Advanced stuff

- Let’s now do some more advanced stuff.
- ‘Advanced’ in terms of perl.
- ‘Advanced’ in terms of linguistics.
Advanced perl

- Object-oriented programming and public perl modules
Advanced perl

• Object-oriented programming and public perl modules
• Tk (editperl.pl, visgrep.pl)
Advanced perl

- Object-oriented programming and public perl modules
- Tk (editperl.pl, visgrep.pl)
- Remote computing (bhexp.cgi, dbiex.pl, websearch.pl)
Object-oriented programming
Object-oriented programming

- Object-oriented (OO) programming is another style of programming.
Object-oriented programming

- Object-oriented (OO) programming is another style of programming.
- OO Programs are a network of things or objects, not a list of commands.
Object-oriented programming

- Object-oriented (OO) programming is another style of programming.
- OO Programs are a network of things or objects, not a list of commands.
- Objects have their own data and specialized functions for dealing with their own data.
Object-oriented programming

- Object-oriented (OO) programming is another style of programming.
- OO Programs are a network of things or objects, not a list of commands.
- Objects have their own data and specialized functions for dealing with their own data.
- Objects can be refer to other objects or inherit the properties of other objects.
Translating to OO
Translating to OO

- Original verb-finding program: \texttt{verbs.pl}.
Translating to OO

- Original verb-finding program: `verbs.pl`.
- OO verb-finding program: `verbsOO.pl`.
Caveats

- OO programs are longer.
Caveats

- OO programs are longer.
- OO programs are slower.
Caveats

- OO programs are longer.
- OO programs are slower.
- OO programming in perl is not orthodox OO (lots of “oddments”).
Caveats

- OO programs are longer.
- OO programs are slower.
- OO programming in perl is not orthodox OO (lots of “oddments”).
- 00 programming isn’t intuitive for most folks.
Caveats

- OO programs are longer.
- OO programs are slower.
- OO programming in perl is not orthodox OO (lots of “oddments”).
- 00 programming isn’t intuitive for most folks.
- Unfortunately, 00 programming is essential for some modules.
Tk
Tk

- Tk is an independent language for making graphical user interfaces.
Tk

- **Tk** is an independent language for making graphical user interfaces.
- There is a special perl module so that you can build GUIs indirectly using **Tk**.
Tk

- Tk is an independent language for making graphical user interfaces.
- There is a special perl module so that you can build GUIs indirectly using Tk.
- This module is widely available for Unix/Linux and Windows, but not clear if it’s available for Macs.
Tk

- Tk is an independent language for making graphical user interfaces.
- There is a special perl module so that you can build GUIs indirectly using Tk.
- This module is widely available for Unix/Linux and Windows, but not clear if it’s available for Macs.
- If you want to write programs in Perl that really look like modern computer programs, then you need to use Tk.
GUI coding
GUI coding

- Tk programs use OO programming style; buttons, windows, etc. are all “objects”.
GUI coding

- Tk programs use OO programming style; buttons, windows, etc. are all “objects”.
- You create these objects and then your program waits for the user to interact with them.
GUI coding

- Tk programs use OO programming style; buttons, windows, etc. are all “objects”.
- You *create* these objects and then your program *waits* for the user to interact with them.
- `makecorpus.pl` → `makecorpusTk.pl`
Remote computing
Remote computing

- CGI ("Common Gateway Interface"): programs that run remotely (generating javascript and HTML: `bhexp.cgi`)
Remote computing

- CGI (“Common Gateway Interface”): programs that run remotely (generating javascript and HTML: \texttt{bhexp.cgi})
- Interacting with local or remote databases (generating sql: \texttt{dbiex.pl})
Remote computing

- CGI ("Common Gateway Interface"): programs that run remotely (generating javascript and HTML: `bhexp.cgi`)
- Interacting with local or remote databases (generating sql: `dbiex.pl`)
- Interacting with the web generally (`websearch.pl`)

Perl for Linguists – p.36/38
Modeling
Modeling

- Perl not good for computational clarity; not well-defined
Modeling

- Perl not good for computational clarity; not well-defined
- Perl exceptionally good at string processing
Modeling

- Perl not good for computational clarity; not well-defined
- Perl exceptionally good at string processing
- Computational tasks as string processing: sylpars.pl
What now?
What now?

- Programming and perl hopefully demystified...
What now?

- Programming and perl hopefully demystified...
- Some ideas about what you can do with it if you’re thinking that you need to program.
What now?

- Programming and perl hopefully demystified....
- Some ideas about what you can do with it if you’re thinking that you need to program.
- If you already know some perl, perhaps some other ideas about what to do with what you already know.