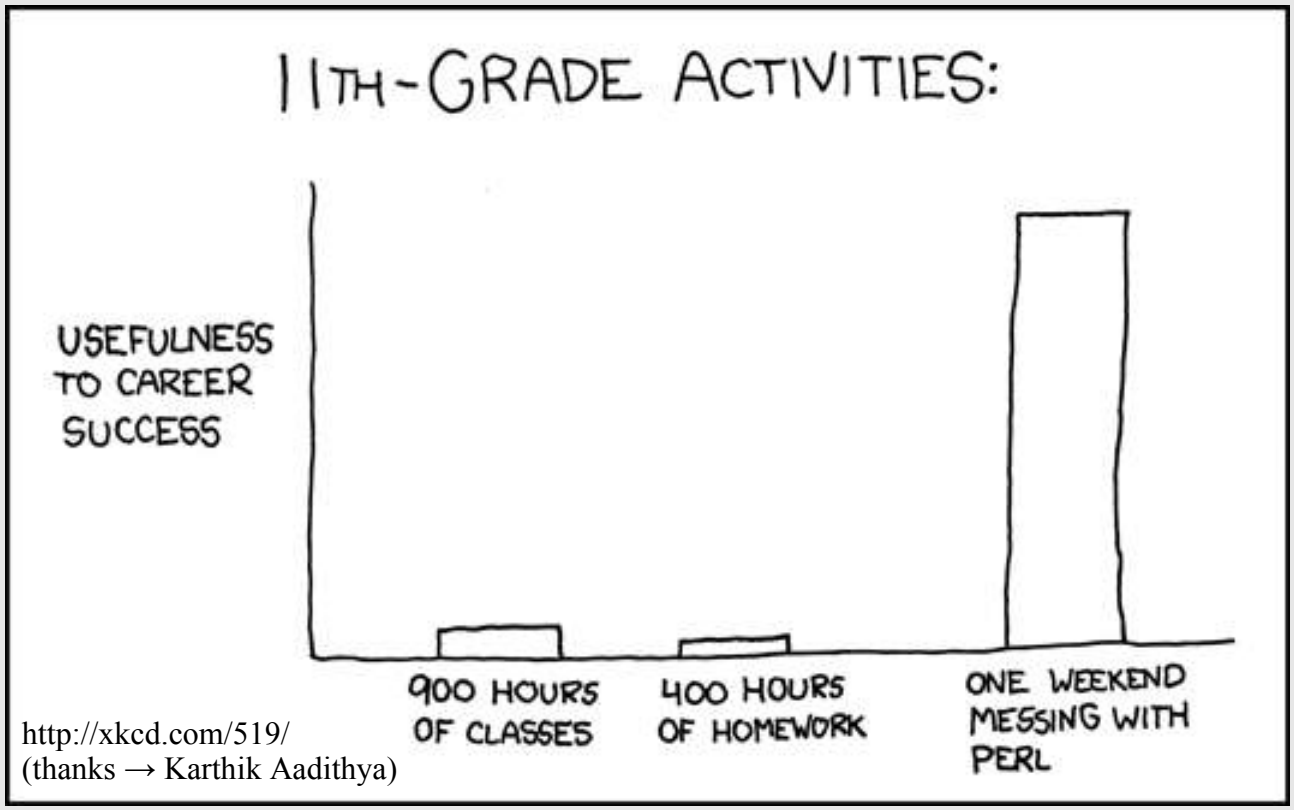
Perl Quickstart

Why Perl?



http://xkcd.com/519/
(thanks → Karthik Aadithya)

- Practical Extraction and Report Language
  - useful features from every prior language: bash, sed, awk, C++, C, Fortran, ...
  - widely used: eg, web, standard in Freescale for modelling, ...
- suggestion: start using it, learn as you go along
  - don't start by making a project of “learning Perl”
- example one-liner: ~/bin/find-installed-packages
- example short program: ~/bin/strip-comments-from-latex

Perl Quickstart

- Documentation: extensive help and tutorial system
  - *man perl* organizes many different help pages. Useful ones:
    - *man perlintro*
    - *man perlcheat*
    - *man perlfunc*
    - *man perlop*
    - *man perlre*
    - *man perlrequick*
    - *man perlretut*
    - *man perlopentut*
- To run perl: (perl -f scriptfile; perl -e '<perl-cmds>'; #!/usr/bin/perl)
  - printing: *print "a b c \n";*
  - variables: *$xyz = "abc"; $abc = 1.3; $abc++; ++$abc;*
  - declaring variables: *my $abc; use strict;*
  - string concat: *print $xyz . "a" . "\n";*
  - reading in something: *$myvar = <STDIN>;*
    - (this gets the newline, too)
  - cleaning strings safely: *chomp($myvar); # changes $myvar*
  - dropping the last character: *chop($myvar); # changes $myvar*

Perl Quick Intro, contd. (2)

- Quick intro contd.:
  - quoted strings:
    - *"$xyz and other stuff \t \n"*: like C printf, variables are substituted
    - '*$xyz and other stuff \t \n*': literal printing
  - operators: numeric vs string
    - numbers: ==, >, <, >=, <=, !=; <=>: *returns -1, 0, or 1*
    - strings: *eq, gt, lt, ge, le, ne*
      - lexicographic comparison: 300 <= 40 is false, 300 le 40 is true
  - lists and arrays
    - list syntax: *("abc", "def", "etc"); qw(abc def etc);*
    - array: @myarr = ("abc", "def", "etc");
    - accessing: *$myarr[3]; @myarr[0..@myarr]; #ranges, returns array*
    - array mode assignment/access:
      - *@newarr = @myarr; @newarr = (@myarr, "append this");*
      - *print @myarr; #array mode: prints array entries, concatenated*
      - *print @myarr . "\n"; # string mode: length of @myarr*
      - *($a, $b, $c) = @myarr;*
    - scalar mode assignment/access:
      - *$a = @myarr; # length of @myarr*
      - *print $#myarr; # idx of last element of @myarr*
    - reading in multiple lines: *@manylines = <STDIN>; # end with EOF=^D*
  - command line argument array: @ARGV

Perl Quick Intro, contd. (3)

- Quick intro contd.:
  - useful functions for arrays
    - *push, pop, reverse*
    - *shift; unshift*
    - *sort; sort {$a <=> $b} @myarr;*
  - if/then/else: *if {...} elsif {...} else {...}*
  - loops
    - *for ($i=0; $i<10; $i++) {...};*
    - *foreach $i (@myarr) {...};*
      - implicit scalar variable $_: *foreach (@myarr) {print; # $_};*
      - perl references: *foreach (@myarr) {$_ *= 2;}; #changes @myarr*
    - *while ($i<100) {...};*
    - *until ($i==100) {...};*
    - *do {...} while ($i<100); do {...} until ($i==100);*
  - functions
    - *sub myfunc {print "@_"; return reverse @_;}*
    - *($a, $b, $c) = myfunc(qw(a b c d e)); # like Matlab*

Perl Quick Intro, contd. (4)

- Quick intro contd.:
  - string matching, substitution, splitting
    - *if ($mystr =~ /$someRE/) { ... }; $mystr =~ s/$myRE/$otherRE/;*
    - *$mystr = "This is a istring"; $oof = "^This(.*)(.*)\\1(.*)\$";*
    - *if ($mystr =~ m@/withslashes/@) { ... };*
    - *$mystr =~ s/$oof/$1,$2,$3/;*
    - Perl regular expressions:
      - spaces, “nonspace”, digits: \s, \S, \w, \W, \d, \D
      - any char, multiple occurrences: ., *, +
      - word boundaries: \b, \B
      - “greediness”: default is max; for min, follow by ?: *?, +?, etc.
      - *m/(...).*(...)/; print “$1 $2”;*
      - OR: |
    - *@myarr = split(/\s+/, $mystring);*
    - *$mystring = join(',', @myarr);*
  - file opening/closing/access
    - *open(FH,”filename”); open(FH,”<”, $filename); open(FH,”>”, ...);*
    - *close(FH);*
    - die: *open(FILEHANDLE,”filename”) || die “open failed: $!”;*
    - *opendir, unlink*, *rename, chmod, chdir*, etc. (man perlfunc)
    - opening/reading all cmdline args as files:
      - *while (<>) {echo $_;} # no args? read stdin*

Perl Quick Intro, contd. (5)

- Quick intro contd.:
  - file existence tests (a la bash's [-X filename]):
    - *-e, -f, -d, -l, -r, -w, -x*
  - hashes (associative arrays): *%myhash*
    - simple assignment: *$myhash{"abc"} = "def"; #sort of like Matlab cell*
    - list of keys: *@mykeys = keys(%myhash); if (keys(%myhash)>5) {...};*
    - list of values: *@myvals = values(%myhash);*
    - *foreach $myval (keys(%myhash)) { ... };*
    - *while (each($mykey,$myval)) { ... };*
    - *delete $myhash{$mykey};*
    - hash to array conversion: *@myarr = %myhash;*
    - array to hash conversion: *%myhash = @myarr; %myhash = (1,2,3,4);*
    - scalar access of hashes: *if (%myhash) {...};*
    - hash slices:
      - *@myhash{@mykeys} = @myvals;*
      - *@existinghash{keys(%myhash)} = values{%myhash};*
      - *print "@myhash{@mykeys} @myhash $myhash";*
  - shell commands and system interaction:
    - *$stdoutput = `date`;*
    - *$retval = system("date"); # $? is returned*
    - environmental variables*: %ENV*
  - eval
    - *$a='$b'; $b='$c'; $c="oof"; eval "\$a=$a";*

Perl Quick Intro, contd. (6)

- Perl references
  - all references are scalars
    - *$myref = \@myarr; $myref = \$myscalar; $myref = \%myhash;*
  - shortcuts for references to arrays and hashes
    - *$myref = ["a", "b", "c"]; # same as @myarr = qw(a b c); $myref=\@myarr;*
    - (anonymous array/hash created: a bit like using malloc/new)
    - *$myref = { "key1" => "val1", "key2" => "val2" };*
    - *$reftoemptyarr = []; $reftoemptyhash = {};*
  - dereferencing: enclose reference within **{}**
    - *$myref = \@myarr; @newarr = @{$myref}; # same as @newarr=@myarr;*
    - *$fourthmem = ${$myref}[3]; # same as $fourthmem = $myarr[3]*
    - *$myref=\%oldhash; %newhash= %{$myref}; # same as %newhash = %oldhash;*
    - *${$myref}{"key"} = "val";*
  - copies of references are still references (think C pointers)
    - *$newref = $myref; # like C pointers, not C references!*
  - C pointer like syntax
    - *$myref->{"key"} = "val"; # equivalent to ${$myref}{"key"} = "val";*
    - *$myref->[2] = 5.6; # equiv to ${$myref}[2] = 5.6;*
  - multidimensional arrays in Perl
    - *@my2darr = ([1, 2, 3], [4, 5, 6], [7, 8, 9]);*
    - *@my3darr = ([[1,2], [3,4]], [[5,6],[7,8]]);*
    - *$my2darr[1]->[2] = "was6";*
    - *$my3darr[1]->[1]->[0] = "was7";*
  - more shortcut notation
    - *$my3darr[1][1][0] = "was7"; # drop multiple ->: same as $my3darr[1]->[1]->[0] !*
    - *@my3darr[1][1]; # same as @my3darr[1]->[1], == (7,8)*