perl is important

- and interesting for who is busy
- and weired for who is clever
- and useful for laziness
- and slow for study
- and practical for learning
- and poor for eye
- and helpful for heart

we should be good at ...

- Java, Oracle, SAP, XML, ASM ...
- being guru proved to be a dream
- know why we learn
- we are poor job seeker
- we have to learn life long
- we need to be healthy
- we are born to be coder

we are jargon hacker

- we are copy-paste worker
- change job just for favorite seat
- programming only in editor
- dream to be user
- hard to show talent at interview
- shame to tell baby my title
- prove ideas only in mind

Summary

- we should be good at everything
- but we are just a jargon hacker
- we need a change in learning
- but what to do?

my perl learning

- picking up little camel book in 200[12]
- let it sleep for (months|years)
- get hired after seeking job for 4 months
- it's a half year out-sourcing project
- anyway work in unix environment
- but find a copy key in keyboard
- happily testing PBX without training

hello world with perl

- counting sizeof memory struct
- recursive, union, 4Bytes padding
- half months learning
- almost finished
- but C has no standard on padding
- go to new job with perl
- build hexdump for sally

goto old project with $_

- taxi ordering with SMS gateway
- building 2 way net proxy
- dig for perl module in cpan
- got solved in 120 lines perl
- parameter file is not an issue
- comments is self document
- leaving without bug phone call

perl for training

- new job for a nerd
- just experience on perl meetup
- call each student to do self-introduce
- Oracle is an oracle in teaching
- can perl help in this task?
- uniread, script, unlock.pl, DBIPR
- writing document with POD

Summary

- perl for TCP programming
- perl for relationship
- perl for learning and hacking
- but how to learn $language?

Story of programming

- should know maths to be developer
- need better editor to enforce nice style
- work OT until monitor smoke
- talk to little bear, and just my bear
- object oriented is everything
- always in beta version
- we need more than coder

Life of administration

- seems laziness but actually busy
- reboot @machine will take an hour
- down time is less than run time
- always the last to get alert
- run to office in midnight to press enter
- we need more than operator

Poem of Blue Collar

- we the unwilling,
- led by the unknowing,
- are doing the impossible
- for the ungrateful.
- we have done so much for so long
- with so little
- we are now qualified to do anything
- with nothing

Python paradox

- perl is ugly & too clever?

有不多的公司能够明智地认识到这点。当然这里还有另一种可能使之得到相同的效果，就是公司里的程序员都非常热爱他们的编程工作而一拍即合。Google举个例子。当他们刊登招聘Java程序员的广告时，同样要求有Python的编程经历。

有一个懂得几乎所有编程语言的朋友，他用Python来做几乎所有的项目。他说主要原因在于他喜欢Python源代码看上去的样子。对于这语言而不是另一种，这是一个看似轻佻的理由。但它没有它听上去那么卤莽：当你编程时，比起书写代码你得花更多的时间去阅读它们。你塑家一样将泥放到的艺术品上，你的代码被一行一行地加上去。因此代码不那么好看的编程语言会让严谨的程序员抓狂，如同雕塑家看到精瑕疵的作品一样。

然提到了丑陋的源代码，许多人理所当然地会认为这是Perl。但是Perl代码所谓的丑陋只是浮于表面的现象，并不是我所说的那种丑类。真正的丑陋不是肤浅地从语法上来评判，而是你被迫基于一种错误的观念来架构你的程序。Perl也许看上去象一堆由卡通字符串组成但是这里有许多案例来证明他有比Python更先进的理念。

今为止，不管怎么说。这两种语言尽管设计的目标不同，但是它们，包括Ruby（以及Icon，Joy，J，Lisp，和Smalltalk）都是为那些真正关心创建并给他们使用的。这是事实。这种程序员总会试图去做得更好。

Summary

- life is hard & full of error
- we need to be wise
- perl => wisdom in %love
- let's talk about hello world

speak slower, listen faster

- print “hello world\n”
- print qq(hello world \n)
- print 'hello world', “\n”
- print q(hello world), qq(\n)
- printf '%s %s %s', 'hello', 'world', "\n"

list and array

- @greets = qw(hello world)
- push @greets, qq(\n)
- print qq(@greets)
- print join ' ', @greets
- print qq($_) for @greets
- $,=qq(\n); print @greets
- use English; $OFS=qq(\n); print @greets

hash map or SQL

- *select distinctive * from ...*
- @score=((80..89), (85..88))
- %dov=map {$_, 1} @score
- print scalar keys %dov
- *select key, count(*) from ... group by key*
- $cov{$_}++ for @score
- @log=map {“$_ $cov{$_}\n”} keys %cov
- print qq(@log)

Summary

- syntax is flexible
- list is strong
- hash is amazing
- why not bigger cake?

Return scalar or array

- sub distinct {
- my (%seen, @row)=();
- @row = grep {++$seen{$_}==1} @_;
- return @row if wantarray;
- return $#row; #let in a bug
- }
- print distinct @score
- print scalar distinct @score

Want hash? return array

- sub count {
- my %cov;
- $cov{$_}++ for @_;
- return %cov
- }
- %cnt = count @score
- print grep {$cnt{$_}>1} keys %cnt
- *select ... group by ... having count(*)>1*

Sorting hash on keys

- print for sort keys %cnt
- @sorted = map {qq($_ => $cnt{$_}) }
- sort keys %cnt;
- print qq(@sorted);
- @rsort = map {qq($_ => $cnt{$_}) }
- sort {$b <=> $a}
- keys %cnt;

Summary

- sort, grep, map works together
- limited loop usage
- a little like pipe
- just reverted, without |
- so what's more on real OS?

OS monitoring

- #/usr/bin/perl -w
- @ps=`ps -ef`;
- @ps_user = map {/(^\w+)/}
- @ps[1..$#ps];
- print count(@ps_user);
- sub count {...}

Running my tool

- % sudo mv psgusr /usr/bin
- % sudo chmod a+x /usr/bin/psgusr
- % file `which psgusr`
- /usr/bin/...: perl script text executable
- % psgusr
- joe8root58oracle12...
- % perl /usr/bin/psgusr

use module

- use Text::Table;
- $tb = Text::Table->new(
- { title => score},
- { title => '| ', is_sep=>1},
- { title => total}
-);
- print $tb->load([100, 3], [99, 4], [90, 8]);

ps group 2.0

- %psg=count(@ps_user);
- @pst=map {[$_, $psg{$_}]} keys %psg;
- $tb = Text::Table->new(
- { title => 'user'},
- { title => 'processes'}
-);
- print $tb->load(@pst);

Summary

- user focus on data
- perl focus on practice
- module can be useful
- powerful thing need better interface
- so how be helpful?

package, where suite hang

- psh% modules
- psh% symbols
- package is a name space
- package holds variable, code and handle
- main:: can be omitted
- module contains one or more packages

import package symbols

- module can influence main package
- use Data::Dumper
- print Data::Dumper::Dumper(@INC)
- use Data::Dumper qw/Dumper/
- print Dumper(\@INC)
- print qq(@Data::Dumper::EXPORT)
- psh% symbols

PSQL.pm

- package PSQL;
- use Exporter 'import';
- our @EXPORT = qw(distinct count);
- sub distinct {...}
- sub count {...}
- 1;
- #put it inside current path
- % perl -MPSQL -MPOE -I. -le 'print for distinct map {/(^\w+)/} keys %INC'

symbol table isa hash

- psh% use PSQL
- psh% symbols
- psh% symbols PSQL
- psh% print *::count{CODE}
- psh% print *PSQL::count{CODE}
- psh% require PSQL
- psh% import PSQL qw/distinct/

Summary

- package is dynamic
- module is package in pm file
- use is compile time require+import
- import that package's @EXPORTed code
- @EXPORT_OK is gentle version @EXPORT
- then how to share it with $world?

Module::Starter

- % module-starter --module=PSQL \
- --author=”Joe Jiang” \
- --email=lamp.purl@gmail.com \
- --builder=Module::Build
- % cd PSQL; perl Build.PL; ./Build
- % rm t/boilerplate.t; ./Build test
- now copy code into lib/PSQL.pm
- % ./Build dist

at the user side

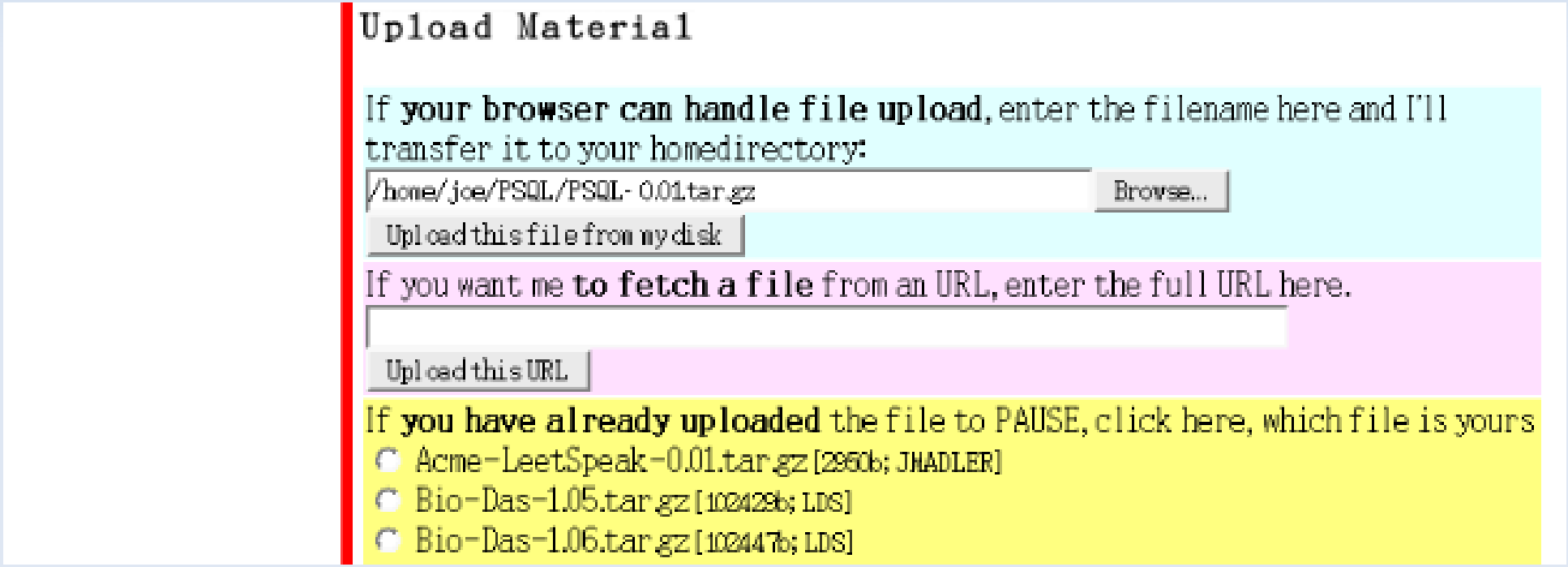
- tar xvfz PSQL-0.0.1.tar.gz
- cd PSQL-0.0.1
- perl Build.PL
- ./Build
- ./Build test
- sudo ./Build install
- perldoc PSQL
- perl -MPSQL -le 'print for distinct (1,1,2)'

pause.cpan.org

- request an PAUSE account
- upload your tar.gz file
- wait for your favorite mirror get synced
- wait for more info from user
- or beg for a victim user
- make ppm for active-perl user
- or just mail randy, ask him do a favor

the click make you famous

- upload file to author/id/J/JO/JOEJIANG
- arrive your mirror in 2 days



beg for a ppm package

Summary

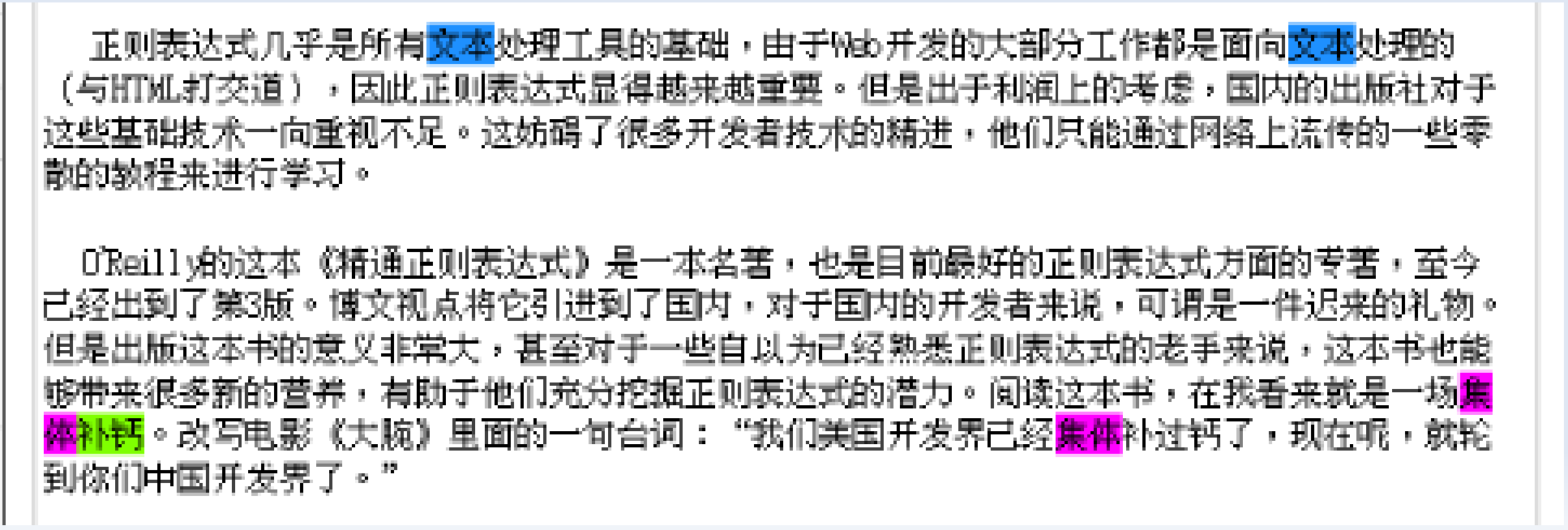
- we can influence the world
- the user will notice it before we dead
- so module quality do matter
- at least we can increase version number
- persuade people use it with better doc
- talk it in perl meetup as me
- even buy them dinner

pause

- next is regular expression

regex

- re => really
- g => grep
- ex => exist in each platform
- ps -ef | grep perl | wc -l



正则表达式几乎是所有文本处理工具的基础，由于Web开发的大部分工作都是面向文本处理的（与HTML打交道），因此正则表达式显得越来越重要。但是出于利润上的考虑，国内的出版社对于这些基础技术一向重视不足。这妨碍了很多开发者技术的精进，他们只能通过网络上流传的一些零散的教程来进行学习。

O'Reilly的这本《精通正则表达式》是一本名著，也是目前最好的正则表达式方面的专著，至今已经出到了第3版。博文视点将它引进到了国内，对于国内的开发者来说，可谓是一件迟来的礼物。但是出版这本书的意义非常大，甚至对于一些自以为已经熟悉正则表达式的老手来说，这本书也能够带来很多新的营养，有助于他们充分挖掘正则表达式的潜力。阅读这本书，在我看来就是一场集体补钙。改写电影《大脑》里面的一句台词："我们美国开发界已经集体补过钙了，现在呢，就轮到你们中国开发界了。"

every thing in unix is file

- and most of them flat file
- even inside the binary => strings
- most of the time grep works for admin
- perl regex is only an advanced sed + awk
- it's a mini language inside perl
- but mixed with love to life

report perl related deb

- strace command-not-found perl 2> errlog
- grep i386 errlog
- cd /usr/share/command-not-found/programs.d
- cat *.db | strings | egrep '^\w+$' | grep perl | sort | uniq
- egrep is grep -E
- sort | uniq makes uniq works here

dont forget to rm $errlog

- but we dont even need to generate $file
- 2>&1 will be even cool
- the two can be combined into one
- without cd here and there
- our module can be used to make distinct
- once it's too long shell, think perl
- but leave it as homework

Summary

- shall avoid grep -v as possible
- with better regex
- text file can pipe smoothly
- let's build the tool chain

basic notes

- ^ and $ means ^a & ^e in bash
- called anchor, slim point in string
- ^$ match blank lines without //sm
- \w means words [A-Z0-9a-z_]
- + means repeat more than *

catch into scalar

- catch matched with ()
- @ps_user = map {/(^\w+)/} @ps
- catch into array index by ('s order
- $ps[1]=~/^(\w+)\W+(\w+)\W+(\w+)/
- $1 => root #uid
- $2 => 1   #pid
- $3 => 0   #ppid

use ^p! in psh

- notation obey your needs
- let it clear by writing m{...}smx
- according to perl best practice
- print map {m{(^[a-z]+)},”\n”} @ps
- use [a-z] to match all lc pid
- use \W to mean [^\w]
- use these less cause we love lower case

Summary

- catch into ($1, $2, $3 ...) for array
- else return match count for scalar
- return true for match even with no ()
- false mean grep -v
- use special char carefully
- regex is case sensitive

match famous process

- print the ps count of root and www
- print count map {m{^(www|root)}sm} @ps
- we are chief admin now
- but how to focus on $cmd?
- split it with spaces => \s+
- p! print map {@f=split /\s+/; "$f[7]\n"} @ps

but i need the parameter

- print map {
- my $b=$_; #get local copy
- @f=split /\s+/; #split as far as possible
- $b=~s/.*$f[6] //;#set @f[1..6] with \0
- $b
- } @ps
- # quick and dirty
- # why so long?

perl one liner

- ps -ef | perl -plane 's/.*$F[6] //'
- -e => code from parameter
- -n => next line till EOF
- -a => auto split
- -l => print ... “\n”
- -p => print $_ for each line

Summary

- perl can be used in air-line
- perl golfer is fun
- funny things is easy with practice
- perl is practical
- so perl is easy