processes

process identifier (PID)

- PID 0: spawns the ‘guts’ of the kernel
  - init system (/sbin/init)
    - memory management system (swap)
    - thread management system (kthreadd)
    - etc.

new PIDs are created for each process

- maximum of 32,768 on Ubuntu (signed 16 bit int)
- PIDs are recycled

process tools

ps	lists active **p**rocesse**s**
top	shows processor (CPU) and memory usage
iotop	shows disk (and network) usage
gpustat	shows processor (GPU) and memory usage
kill x	politely (or not) ends process x
x &	runs command x in the background
wait	waits for background task(s) to end
nice -n x y	sets processor priority x for command y
renice -n x y	changes processor priority x for process y
nohup x	continues to run command x after logout
./x	runs command x

serial computing

processing with a single thread of execution

default for most computer languages and programs

computations (seem as if) completed, in order, one at a time

- (appear to have been) done on a single CPU
- (most) CPUs complete instructions out of order (faster)
- (most) CPUs compute instructions in parallel (faster)
  - (most) CPUs process both parts of a conditional branch
  - return only the correct branch

parallel computing

processing with multiple ‘independent’ threads of execution

- rarely (the apparent) default

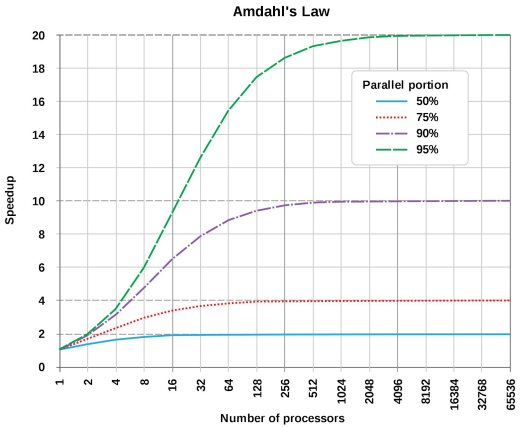
primary/worker architecture

- primary thread creates/destroys worker threads
- worker threads have shared and/or independent resources

requires explicit thread creation/destruction

- language–level: very efficient, sometimes annoying for the programmer, easy for the user
- program–level: sometimes efficient, very annoying for the programmer, easy for the user
- user–level: very efficient, easy for the programmer, sometimes annoying for the user

(useless) parallel computing



often called Amdahl's law

parallel speed is limited by serial tasks

i.e. the creation and destruction of threads has a cost

speed = $(1-\%parallel+(\%parallel/speedup))^{-1}$

very common for resource–constrained computations

I/O bandwidth: disk, network, RAM read and/or write

https://en.wikipedia.org/wiki/

greater speed: optimization priority

1) reduce task size
    prefilter, compress, compute on unique inputs only
2) make computations easier (reduce code complexity)
    simpler (faster to compute) correlated values
    better implementations
3) reduce dependent calculations
    estimate values from a random subsample
4) parallel threads of execution where possible

embarrassing (pleasingly) parallel

very common

the same task(s) to be carried out on many files/lines/etc.

each file/line/etc. can be processed independently

results may, or may not, be combined at the end

    e.g. bootstrap/jackknife analysis

a maximum of $(nproc) faster

a perfect use for user–level parallel computing

user–level parallel computing

GNU parallel

- creates parallel threads from stdin input and/or shell loops
- not ‘standard’ in most LINUX environments
- a Perl script

xargs

- the original: ‘standard’ in POSIX environments
- creates parallel threads from stdin input
  - e.g. ls x/ | xargs -I % -P $(nproc) bash -c 'xz -cd x/% > y/%'
  - e.g. tail -n 1 x | wc -c | xargs -I {} truncate x -s -{}

parallel computing: xargs

xargs e**x**tended **arg**uments: highly–scalable with os–level resource segregation parallel execution on a stdin stream (e.g. file names)

best to use null delimited input (tr '\n' '\0')

arguments:

-0 input is null (**0**) delimited

-I **i**nput substitution character(s) (e.g. %, {})

-P number of parallel **p**rocesses $(nproc)

the compute job

e.g. bash -c 'program0 "{}" | program1 > "{}.out0"; program2 "{}" "{}.out1"'

parallel computing: multiple computers

multiple computers == more I/O bandwidth

- great speed gains are often possible

data transfer is often a major bottleneck

- prefilter, unique, and compress data before transfer

manually synchronizing software and data is annoying

- use containers (e.g. Docker) or virtualization
- store data on a network accessible file server

high–performance computing (HPC)

processing with multiple ‘independent’ computers

- connected via a network
- usually have shared drives
- sometimes have shared memory
- primary/worker architecture

requires a job scheduler

- control the starting/stopping/running of jobs
- control resource use
- (accounting of resource use)

HPC: jobs

can be run as in batch or (uncommonly) in interactive mode

use a shell script that does the entire compute job

- transfer input data to the compute node (perhaps use the job scheduler)
- transfer analysis software to the compute node (if needed)
  - may need to be compiled or locally installed
  - access preinstalled or load a container/virtual machine
- run the compute task(s)
- transfer output data to a safe location (perhaps use the job scheduler)

HPC: job schedulers…

Portable Batch System (PBS)

- originally developed for ‘NASA’ (starting in 1991)
- OpenPBS released in 1998

TORQUE

- an extension of OpenPBS
- released in 2003
- more fault tolerant than PBS

HPC: …job schedulers…

Load Sharing Facility (LSF)

- developed by Platform Computing (now part of IBM)
- OpenLava (open source version) released in 2007
- IBM sued to prevent OpenLava distribution

Sun Grid Engine (SGE; now Univa Grid Engine)

- development started in 1993 (CODINE)
- versions released 2001–2010 are open source
- modern open source derivative: Son of Grid Engine

HPC: …job schedulers

HTCondor

- open source, University of Wisconsin (Madison)
- can be used for ‘cycle scavenging’ (moving jobs around)
- includes power management features
- includes features for job resizing and job moving

Simple Linux Utility for Resource Management (SLURM)

- open source, Lawrence Livermore National Laboratory
- includes power management features
- includes features for job resizing

cloud computing: someone else's HPC

billed by a mixture of time/priority/data

- perhaps cost effective for development and testing
- very expensive for large–scale analyses
  - can buy a computer for the cost of one big analysis

often use a 'proprietary' job scheduler

- typically web and script wrappers around a standard scheduler

often provides access to 'exotic' hardware (e.g. TPU)

typically uses a mixture of containers and virtualization

cloud computing: kubernetes

an open source container–centric job scheduler

developed by Google (2015)

- maintained by the Cloud Native Computing Foundation

geared towards providing stateless web services

- bandwidth is more important than processing power
- each container computes many small jobs
- network traffic in/out of the HPC is managed

text processing utilities…

agrep	**a**pproximate **grep** (tre-agrep)
awk	a pattern scanning programming language
bc	a **b**asic **c**alculator language
bloom	a **Bloom** filter
cat	con**cat**enate files
datamash	an advanced calculator and table manipulation program
diff	find **diff**erences between two files line–by–line
grep	**g**lobally search a **r**egular **e**xpression and **p**rint
head	output the first part of a file
join	**join** lines of two files using a common field
perl	**p**ractical **e**xtraction and **r**eporting **l**anguage

…text processing utilities

paste	a column oriented concatenation text utility
python	a (text processing) programming language
sed	**s**tream **ed**itor for filtering and transforming files
shuf	**shuf**fle lines in a file (do not use)
sort	**sort** lines of files
split	**split** a file into pieces
tail	output the last part of a file
tr	**tr**ansliterate (or delete) characters
uniq	**uniq**ue (or not) lines
wc	**w**ord (and other things) **c**ount

basic tools

finding particular data:

awk, bloom, diff, grep, perl, python, sed, tre-agrep, uniq

moving data:

cat, head, join, paste, sort, split, tail

editing data:

awk, echo, perl, python, sed, tr

counting data:

awk, bloom, datamash, grep, perl, python, sed, tre-agrep, uniq, wc

math:

awk, bc, datamash, perl, python, sed

finding particular data

lines that differ among files

diff, uniq, [awk], [perl], [python], [sed]

(in)exact line matching

grep, tre-agrep, python, [awk], [perl], [sed]

(in)exact column matching

awk, perl, python, sed

approximate line matching

tre-agrep, python, perl, [awk], [sed]

moving data

merging whole files

cat, join, paste, sort, [awk], [perl], [python], [sed]

extracting particular lines

head, tail, [awk], [perl], [python], [sed]

making subfiles

head, split, tail, [awk], [perl], [python], [sed]

merge/split columns

awk, perl, python, sed

editing data

replacing characters

tr, [awk], [perl], [python], [sed]

replacing words

perl, python, [awk], [sed]

adding data

echo, awk, [perl], [python], [sed]

counting data

counting lines

- wc, uniq, [tre-agrep], [awk], [grep], [perl], [python], [sed]

counting words/characters

- wc, [awk], [perl], [python], [sed]

counting instances of a particular thing

- (in)exactly
  - tre-agrep, grep, perl, python, uniq, [awk], [sed]
- approximately
  - tre-agrep, python, [awk], [perl], [sed]

math

sum (etc.) a particular column

awk, datamash, perl, python, sed, [bc]

merge columns by sum (etc.)

awk, perl, python, sed

numeric manipulation

datamash, perl, python, awk, [bc], [sed]

https://xkcd.com/208/

regular expressions

also known as regex or regexp or RE

used to find user defined patterns in text

used in many programs

SEXI (SNOBOL) => ed => grep => awk => perl

search patterns (usually) delimited by slashes e.g. /seek/

or almost any other non–word characters e.g. #seek# in Perl

most programs use Perl Compatible RE (PCRE)

pattern matching: tr///

good for ‘cleaning’ data

character based

e.g. echo 'CBA' | perl -pe 'tr/ABC/abc/' # cba

[ ] \ / have a special meanings

use \ to get the literal meaning e.g. \\ eq \

can delimit ranges of characters e.g. [A-z], [0-9]

pattern matching: tr///

modifier c: complement

e.g. echo 'ABC' | perl -pe 'tr/A/x/c' # Axxx

modifier d: delete

e.g. echo 'ABC' | perl -pe 'tr/A//d' # BC

modifier s: squash

e.g. echo 'ABB' | perl -pe 'tr/B//s' # AB

modifiers can be used in combination

e.g. echo 'ABC' | perl -pe 'tr/A//cd' # A

can be used to count number of occurrences

e.g. echo 'AABC' | perl -lane 'BEGIN{$A=0}{$A+=($_=~tr/A/A/)}END{print($A)}' # 2

pattern matching: m//

useful for finding text

returns 0 or 1 (true or false)

best used explicitly with =~ or !~

e.g. if($x =~ m/A/) not if(/A/)

stops after finding the first match

complex patterns are slower than exact matches

pattern matching: m//

minimalism is good

capitalize on the smallest differentiating feature

fewer errors, much faster

patterns can be simple e.g. m/ABC/

or complex e.g. m/A[A-z]+C/

. * ? + [ ] ( ) { } ^ $ | \ / have special meanings

escape using slash \

pattern matching: m//

[] delimit sets of characters e.g. [ABC], [A-z]

[^] delimit exclusion sets e.g. [^ABC], [^A-z]

\d digit; \D non–digit

\w word character [a-zA-Z0-9_]; \W not word character

\s white space (\n, space, tab, etc); \S not white space

^ start of chunk

$ end of chunk

| or e.g. (A|B|C)

. (almost) anything

pattern matching: m//

* zero or more matches

+ one or more matches

? zero or one match

{} number of matches e.g. {5}, {5,}, {5,10}

pattern matching: m//

modifiers: g, i, m, s, x

g: global = find all occurrences, not just the first one

i: case insensitive = uppercase and lowercase letters as equivalent

m: multiline = use \n to delimit ^ and $ matches

s: single line = ignore \n when making matches

x: free form = count white space when making matches

pattern matching: s///

replaces strings in text

e.g. echo 'ABCD' | perl -pe 's/BC/x/' # AxD

replaces only the first occurrence (by default)

works like m// but finds and replaces

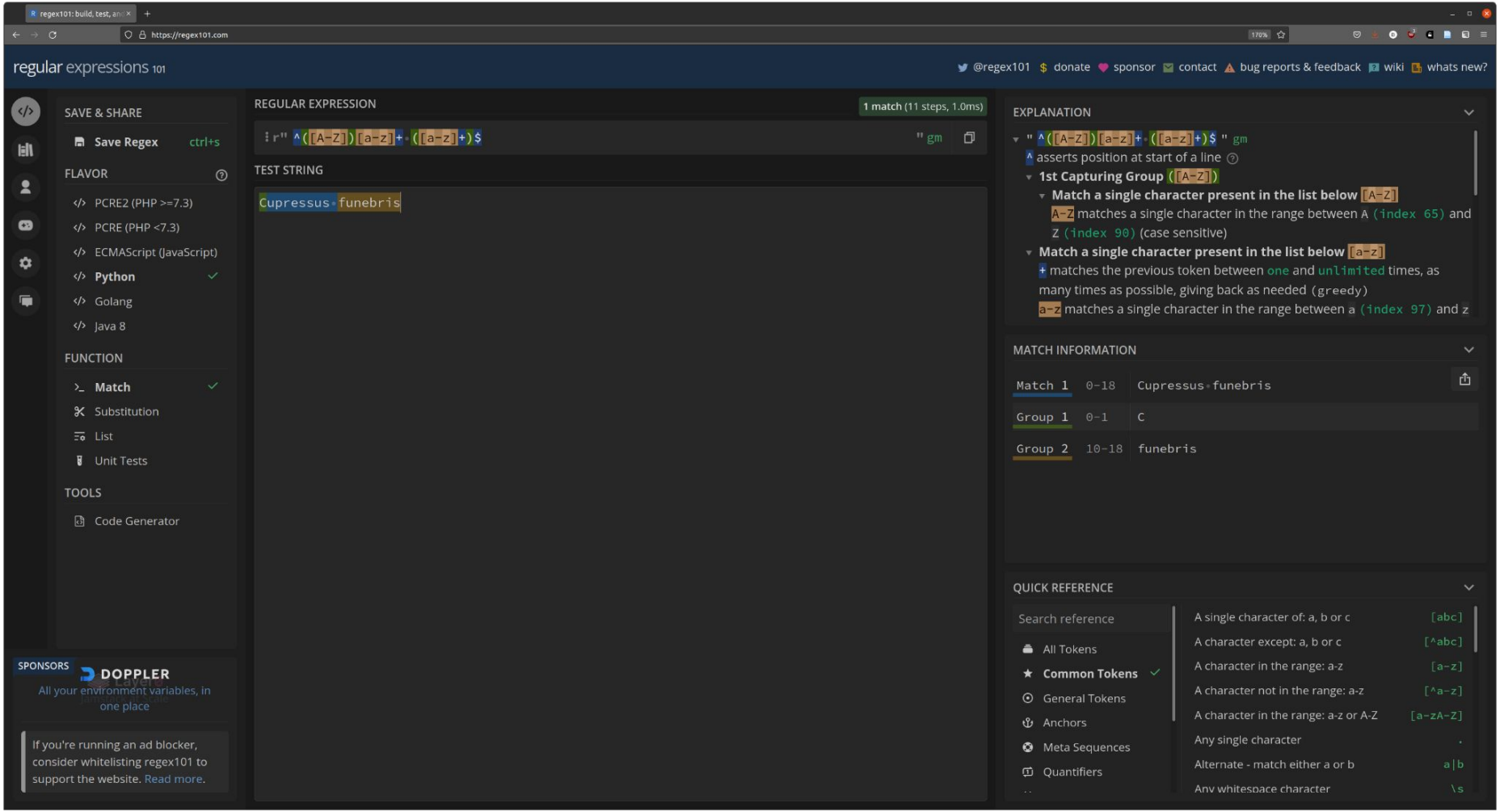
use built–in variable $1, $2, etc. to capture matches

built–in variables start with 1 (like in awk)

e.g. echo 'ABC' | perl -pe 's/(A.C)/found: $1/'

found: ABC

() capturing group; (?:) non–capturing group


regex101: build, test, and
https://regex101.com
regular expressions 101
@regex101 donate sponsor contact bug reports & feedback wiki whats new?
SAVE & SHARE
Save Regex ctrl+s
FLAVOR
PCRE2 (PHP >=7.3)
PCRE (PHP <7.3)
ECMAScript (JavaScript)
Python
Golang
Java 8
FUNCTION
Match
Substitution
List
Unit Tests
TOOLS
Code Generator
REGULAR EXPRESSION
1 match (11 steps, 1.0ms)
r" ^([A-Z])[a-z]+ ([a-z]+)$ " gm
TEST STRING
Cupressus funebris
EXPLANATION
" ^([A-Z])[a-z]+ ([a-z]+)$ " gm
^ asserts position at start of a line
1st Capturing Group ([A-Z])
Match a single character present in the list below [A-Z]
A-Z matches a single character in the range between A (index 65) and Z (index 90) (case sensitive)
Match a single character present in the list below [a-z]
+ matches the previous token between one and unlimited times, as many times as possible, giving back as needed (greedy)
a-z matches a single character in the range between a (index 97) and z
MATCH INFORMATION
Match 1 0-18 Cupressus funebris
Group 1 0-1 C
Group 2 10-18 funebris
QUICK REFERENCE
Search reference
All Tokens
Common Tokens
General Tokens
Anchors
Meta Sequences
Quantifiers
A single character of: a, b or c [abc]
A character except: a, b or c [^abc]
A character in the range: a-z [a-z]
A character not in the range: a-z [^a-z]
A character in the range: a-z or A-Z [a-zA-Z]
Any single character .
Alternate - match either a or b a|b
Any whitespace character \s
SPONSORS
DOPPLER
All your environment variables, in one place
If you're running an ad blocker, consider whitelisting regex101 to support the website. Read more.