bc - an arbitrary precision calculator language

Windows: http://gnuwin32.sourceforge.net/packages/bc.htm
macOS: should come built in
Linux: should come built in

Flags:
- `l` : uses mathlib libraries and makes more functions available
- `q` : quiet, doesn’t show headers when starting

Commands:
- `scale` : changes how many decimal places to use (for integer math, set to 0)
- `ibase` : this is the base numbering system for input
- `obase` : this is the base numbering system for output
- `last` : this returns the last outputted number

Example:

```
$ bc -q -l
/* -l command loads mathlib and sets scale=20 */
scale
20
3+4
7
4*5
20
/* a(x) is the arctan(x), we can use it to define pi */
pi=4*a(1)
radius=7
circumference=2*pi*radius
area=pi*radius^2
pi
3.14159265358979323844
radius
7
circumference
43.98229715025710533816
area
153.9380400258979323844
/* if we have three circles, how much total area is it */
last*3
461.81412007769960605068
/* if you use modulus, watch your scale */
scale
20
10%6
0.00000000000000000004
scale=0
10%6
4
```
Example:
convert a hexadecimal number into decimal and binary.
Note, hexadecimal characters in bc have to be capitalized

$ bc -q
ibase=16
6F1F767BF5E14A4DE9D5DF
134339344986286640331347423
obase=2
6F1F767BF5E14A4DE9D5DF
11011110001111110111110101111011101010100101001101110011101010111011111
Example:
Run our RSA algorithm and encrypt/decrypt the number 17. Our modulus is 55, our public exponent is 7 and our private exponent is 23. The modulo operator is "%", the "^" operator is used for exponentiation.

```bash
$ bc -q
17^7%55
8
8^23%55
17
```
openssl – certificate swiss army knife

Windows: [http://gnuwin32.sourceforge.net/packages/openssl.htm](http://gnuwin32.sourceforge.net/packages/openssl.htm)
macOS: [http://macappstore.org/openssl/](http://macappstore.org/openssl/) or google for other instructions
Linux: should be built in

openssl command [ command_opts ] [ command_args ]

Commands:
- x509 : give us information about a certificate file
- rsa : give us information about a key file
- genrsa : generate an RSA key
- s_client : connect to a host port and talk TLS/SSL. Supports plain SSL, or TLS for smtp, pop3, imap, ftp and ldap (requires patch)
- dgst : run a cryptographic digest like SHA256 or MD5
- aes-256-cbc : encrypt using AES

Example:

Create an RSA key

$ openssl genrsa -out demo_ca.key 2048
Generating RSA private key, 2048 bit long modulus
...............+++++
...................................+++++
e is 65537 (0x010001)
Example:

Create a Certificate for this key (aka, certificate signing request, CSR).

```
$ openssl req -key demo_ca.key -new -out demo_ca.csr
```

You are about to be asked to enter information that will be incorporated into your certificate request. What you are about to enter is what is called a Distinguished Name or a DN. There are quite a few fields but you can leave some blank. For some fields there will be a default value. If you enter '.', the field will be left blank.

```
Country Name (2 letter code) [XX]: US
State or Province Name (full name) []: NJ
Locality Name (eg, city) [Default City]: New Brunswick
Organization Name (eg, company) [Default Company Ltd]: Knights
Organizational Unit Name (eg, section) []: 
Common Name (eg, your name or your server's hostname) []: Knights Signing CA
Email Address []: 
```

Please enter the following 'extra' attributes to be sent with your certificate request
A challenge password []:
An optional company name []:

Brian Epstein <bepstein@ias.edu>
https://security.ias.edu
Example:

Self sign the demo_ca.csr file to make a self-signed certificate authority

```bash
$ openssl req -x509 -in demo_ca.csr -key demo_ca.key -out demo_ca.crt
```
Example:
Create a website certificate and sign it with the demo_ca.crt key

```bash
$ openssl genrsa -out www.knights.edu.key 2048
Generating RSA private key, 2048 bit long modulus
...............................................+++++
..............................................................+++++
e is 65537 (0x010001)

$ openssl req -key www.knights.edu.key -new -out www.knights.edu.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:NJ
Locality Name (eg, city) [Default City]:New Brunswick
Organization Name (eg, company) [Default Company Ltd]:Knights
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:www.knights.edu
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:

$ openssl ca -keyfile demo_ca.key -cert demo_ca.crt -in www.knights.edu.csr -out www.knights.edu.crt -config openssl.cnf -create_serial
Using configuration from openssl.cnf
Can't open /home/ep/certdemo/CA/index.txt.attr for reading, No such file or directory
139987170342720:error:02001002:system library:fopen:No such file or
directory:crypto/bio/bss_file.c:74:fopen('/home/ep/certdemo/CA/index.txt.attr','r')
Check that the request matches the signature
Signature ok
Certificate Details:
  Validity
    Not Before: Oct 29 19:16:42 2018 GMT
    Not After : Oct 29 19:16:42 2019 GMT
  Subject:
    countryName = US
    stateOrProvinceName = NJ
    organizationName = Knights
    commonName = www.knights.edu
  X509v3 extensions:
    X509v3 Basic Constraints:
      CA:FALSE
    Netscape Comment:
      OpenSSL Generated Certificate
    X509v3 Subject Key Identifier:
    X509v3 Authority Key Identifier:

Certificate is to be certified until Oct 29 19:16:42 2019 GMT (365 days)
Sign the certificate? [y/n]:y

Brian Epstein <bepstein@ias.edu>
https://security.ias.edu
1 out of 1 certificate requests certified, commit? [y/n] y
Write out database with 1 new entries
Data Base Updated
Example:
open an x509 certificate and list it's contents. If a file contains multiple certificates, only the first is shown

$ openssl x509 -in www.knights.edu.crt -text -noout
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number:
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: C = US, ST = NJ, L = New Brunswick, O = Knights, CN = Knights Signing CA
    Validity
    Not Before: Oct 29 19:16:42 2018 GMT
    Not After : Oct 29 19:16:42 2019 GMT
    Subject: C = US, ST = NJ, O = Knights, CN = www.knights.edu
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      Public-Key: (2048 bit)
      Modulus:
      8a:1c:85:83:85:46:1d:92:00:0f:87:f4:0e:1b:7f:
      8f:f1:44:63:5a:3c:89:bc:73:87:fc:8c:8a:3f:e8:
      83:6f
      Exponent: 65537 (0x10001)
    X509v3 extensions:
      X509v3 Basic Constraints:
      CA:FALSE
      Netscape Comment:
      OpenSSL Generated Certificate
    X509v3 Subject Key Identifier:
    Signature Algorithm: sha256WithRSAEncryption
    89:97:bb:72:ad:de:ab:2b:5b:0c:3f:01:ab:f7:3e:0d:5b:2c:86:
    e5:e9:52:09:02:70:cf:30:f6:6a:a0:5f:2a:8b:be:57:26:73:
    66:07:bb:03
Example:

generate and display a 174bit RSA key. This is an example of using both the "genrsa" and "rsa" commands together with a pipe on the command line.

```
$ openssl genrsa 174 | openssl rsa -text -noout
Generating RSA private key, 174 bit long modulus
..............................................................
e is 65537 (0x10001)
Private-Key: (174 bit)
    modulus:
        23:5b:cd:05:c6:77:33
    publicExponent: 65537 (0x10001)
    privateExponent:
        ef:08:bc:ce:93:92:b1
    prime1:
    prime2:
    exponent1:
    exponent2:
    coefficient:
```
Example:

multiple the above prime numbers together to verify the modulus. Notice the order of the “obase” and “ibase”. If you switch them, you are setting your output base to 0x16, or 22 in decimal. This also should be on one line.

```bash
  sed 's/://g' | \
  tr 'a-f' 'A-F' | \
  sed 's/^/obase=16;ibase=16;/' | \
  bc | \
  tr 'A-F' 'a-f' | \
  sed 's/\(.*\)/\1:/'
```
Example:
connect to a remote host and look at the certificates it has to offer. The
"showcerts" option will display all the certs offered by the server. These
can be examined with the x509 command. Also notice, once connected, we
can interact with this host as if we just used telnet to port 0. This is extremely
useful for troubleshooting what certificates a host offers, and how to interact
over an encrypted SSL tunnel.

$ openssl s_client -connect security.ias.edu:443 -showcerts
CONNECTED(00000003)
depth=2 C = US, ST = New Jersey, L = Jersey City, O = The USERTRUST Network, CN = USERTrust RSA
Certification Authority
verify return:1
depth=1 C = US, ST = MI, L = Ann Arbor, O = Internet2, OU = InCommon, CN = InCommon RSA Server CA
verify return:1
depth=0 C = US, postalCode = 08540, ST = New Jersey, L = Princeton, street = 1 Einstein Drive, O =
Institute for Advanced Study, CN = www.ias.edu
verify return:1

Certificate chain
0 s:/C=US/postalCode=08540/ST=New Jersey/L=Princeton/street=1 Einstein Drive/O=Institute for Advanced
Study/CN=www.ias.edu
i:/C=US/ST=MI/L=Ann Arbor/O=Internet2/OU=InCommon/CN=USERTrust RSA Certification Authority
-----BEGIN CERTIFICATE-----
MIIGnDCCBYSgAwIBAgIQKF1wLbhTHrcuwzkeNd5X+jANBkgkhkiGi9w0BAQsFADB2
M0swCQYDVQQGEwJVUzELMAkGA1UEBMCVMTkxTjZBVE5QTWRNJTNfMjM4MB4X
MBAGA1UdJQAwZS9XZ25JZMo5UTQ9d3M3Q25EeXNjaHZnYXhnZG46NTY2MDg4
S5d5b21tb24UNBFNiNcn21cBDA0eFw0NxNzhDVQOItmZcWAU0aWRTVGMdLjxHb
BBQgkIBWx1dGhZReUwUwIEQwDQYJKoZIhvcNAQEBBQADggEPADCCAQEEAB
Mz5UNTAiA1EGMsQuCQYDVQQGEwJVUzELMAkGA1UEBMCVMTkxTjZBVE5QTWRNJ
TnfMjM4MB4XMBAGA1UdJQAwZS9XZ25JZMo5UTQ9d3M3Q25EeXNjaHZnYXhnZG46
NTY2MDg4S5d5b21tb24UNBFNiNcn21cBDA0eFw0NxNzhDVQOItmZcWAU0aWRTV
GMdLjxHbBBQgkIBWx1dGhZReUwUwIEQwDQYJKoZIhvcNAQEBBQADggEPADCC
AQQwDQYJKoZIhvcNAQEBBQADggEPADCCAQEEAB
---
Certificate chain
0 s:/C=US/ST=MI/L=Ann Arbor/O=Internet2/OU=InCommon/CN=USERTrust RSA Certification Authority
i:/C=US/ST=New Jersey/L=Jersey City/O=The USERTRUST Network/CN=USERTrust RSA Certification Authority
-----BEGIN CERTIFICATE-----
MIIF+TCCA+GgAwIBAgIQRyDQ+oVGGn4XoWQCKYRjdDANBgkqhkiG9w0BAQsFADB2
M0swCQYDVQQGEwJVUzELMAkGA1UEBMCVMTkxTjZBVE5QTWRNJTNfMjM4MB4X
MBAGA1UdJQAwZS9XZ25JZMo5UTQ9d3M3Q25EeXNjaHZnYXhnZG46NTY2MDg4
S5d5b21tb24UNBFNiNcn21cBDA0eFw0NxNzhDVQOItmZcWAU0aWRTVGMdLjxHb
BBQgkIBWx1dGhZReUwUwIEQwDQYJKoZIhvcNAQEBBQADggEPADCCAQEEAB

Brian Epstein <bepstein@ias.edu>
-----END CERTIFICATE-----
---
Server certificate
subject=/C=US/postalCode=08540/ST=New Jersey/L=Princeton/street=1 Einstein Drive/O=Institute for Advanced Study/CN=www.ias.edu
issuer=/C=US/ST=MI/L=Ann Arbor/O=Internet2/OU=InCommon/CN=InCommon RSA Server CA
---
No client certificate CA names sent
Peer signing digest: SHA512
Server Temp Key: ECDH, P-256, 256 bits
---
SSL handshake has read 5145 bytes and written 380 bytes
Verification: OK
---
New, TLSv1.2, Cipher is ECDHE-RSA-AES128-GCM-SHA256
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
  Protocol : TLSv1.2
  Cipher   : ECDHE-RSA-AES128-GCM-SHA256
  Session-ID: C11FD3CF1391A650E920226192E5CDAC66E7BFE480680C4472363FC951D37E2
  Session-ID-ctx:
  Master-Key:
    C79EF1A7D02ED2CC4D2E89F948159D08AF873F9D0BD5FCFD8A601CE3A9EFE9D9703111646503FDDBFE7E9038F729A91
PSK identity: None
PSK identity hint: None
SRP username: None
Start Time: 1540846225
Timeout   : 7200 (sec)
Verify return code: 0 (ok)
Extended master secret: no
---
Example:

Calculate the SHA256 sum on a host that only has openssl and not the sha256sum tool.

```
$ sha256sum /etc/hosts
bash: sha256sum: command not found...
$ openssl dgst -sha256 /etc/hosts
SHA256(/etc/hosts)= 42c60aee9ac2254ea721673592386164914480669c06c2fad31123344fe71a7f
```
Example:
Quickly encrypt a config file to send over email to a vendor for troubleshooting purposes. I do this all the time, it isn’t too difficult to explain over the phone how to decrypt, and it gives you the option of protecting sensitive data over an insecure medium. You still have to tell them the password over the phone, though, which is better than sending cleartext.

$ openssl aes-256-cbc -in database_credentials.php -out database_credentials.php.enc
enter aes-256-cbc encryption password:
Verifying - enter aes-256-cbc encryption password:

$ file database_credentials.php
database_credentials.php: ASCII text
database_credentials.php.enc: data

$ more database_credentials.php
<?php
$dbuser = "dummy";
$dbpass = "drowssap";
$dbhost = "hopeyoudonthackme.com";
$dbname = "please";
?>

$ xxd database_credentials.php.enc
0000000: 5361 6c74 6564 5f5f 70d2 c0dc c413 bfbf Salted__p.......
0000010: a7e0 124e 6477 42e7 b553 17ff ee6c edb4 ...NdwB...l...
0000020: ab5b 15b2 ab0a 455c d2ef 0cb2 e87a 8350 .[....E
0000030: 2fe4 9b6a 8910 5e8a 2b56 56ce 8f5c c727 /..j..^VV..`
0000040: ebae 66b2 1218 f4fc 2c18 e375 f45d 4915 ..f.....u.
0000050: a756 1d1d bbdb 2a4ab 061f 844b 4f79 7752 .V........K0.wR
0000060: bdca c4a7 27f1 f965 0b19 337b 74cc beb3 ....'..e..3pt...
0000070: 25ad 5c94 bb9b 8581 b36e ffd1 6301 2b59 %.

$ base64 < database_credentials.php.enc | \ 
mail -s "Database credentials you asked for" support@example.com

Example:
Inspect an ssh RSA key for it's components

$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ep/.ssh/id_rsa): /home/ep/.ssh/rutgers
...
Your identification has been saved in /home/ep/.ssh/rutgers.
Your public key has been saved in /home/ep/.ssh/rutgers.pub.
...

$ openssl rsa -in .ssh/rutgers -text -noout
Private-Key: (2048 bit)
modulus:
...