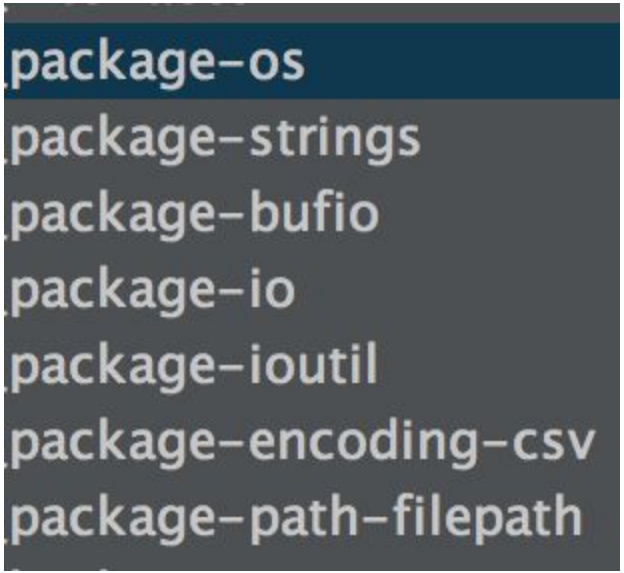
readers & writers

What do you know about readers and writers?

Are “readers” and “writers”
different from “reading” and “writing”?



What might you find in these packages related to reading and writing?


godoc.org/io
okmarks
type Reader
type Reader interface {
Read(p []byte) (n int, err error)
}


godoc.org/io
okmarks
type Writer
type Writer interface {
Write(p []byte) (n int, err error)
}

Reading A File

main.go

```go
package main

import (
    "log"
    "os"
    "io/ioutil"
    "fmt"
)

func main() {
    f, err := os.Open("hello.txt")
    if err != nil {
        log.Fatalln("my program broke")
    }
    defer f.Close()

    bs, err := ioutil.ReadAll(f)
    if err != nil {
        log.Fatalln("my program broke")
    }

    fmt.Println(bs)
    fmt.Println(string(bs))
}
```

Terminal

```
01_open-file $ go run main.go
[72 101 108 108 111]
Hello
01_open-file $
```

main.go

```go
package main

import (
    "log"
    "os"
    "io/ioutil"
    "fmt"
)

func main() {
    f, err := os.Open("hello.txt")
    if err != nil {
        log.Fatalln("my program broke")
    }
    defer f.Close()

    bs, err := ioutil.ReadAll(f)
    if err != nil {
        log.Fatalln("my program broke")
    }

    fmt.Println(bs)
    fmt.Println(string(bs))
}
```

Terminal

```
01_open-file $ go run main.go
[72 101 108 108 111]
Hello
01_open-file $
```

Reading A File

f
implements the
reader interface

ReadAll
takes a reader interface

Reader Interface

io.Reader

Remember what we learned about interfaces with our “vehicle interface” analogy

You could say
interfaces allow us to group things
by functionality; what they do; their methods

say I need to go from here to LA
I’ll ne **vehicle** to get there:
uck, bike, plane
all of thos sportation implement the “vehicle interface”
for what it m to be a vehicle

An interface is like
having more than one type:
“I’m a plane, and I’m a vehicle”
“I’m a truck, and I’m a vehicle”
“I’m a boat, and I’m a vehicle”

ight”
“a **vehicl**
over if you went throu car …
ferent modes of transportation
they all satisfy the criteria for what it

An interface is like
having more than one type:
“I’m a file, and I’m a reader interface”
“I’m a string, and I’m a reader interface”
“I’m a file, and I’m a writer interface”

if
then anything reader

ioutil.ReadAll takes a reader interface as an argument,
so it can take a file or a string:
func ReadAll(r io.Reader) ([]byte, error) { }

main.go

```go
package main

import (
    "log"
    "os"
    "io/ioutil"
    "fmt"
)

func main() {
    f, err := os.Open("hello.txt")
    if err != nil {
        log.Fatalln("my program broke")
    }
    defer f.Close()

    bs, err := ioutil.ReadAll(f)
    if err != nil {
        log.Fatalln("my program broke")
    }

    fmt.Println(bs)
    fmt.Println(string(bs))
}
```

Terminal

```
01_open-file $ go run main.go
[72 101 108 108 111]
Hello
01_open-file $
```

Reading A File

f
implements the
reader interface

ReadAll
takes a reader interface

an interface is a type

- f is type ***File** and type **reader interface**
- **ReadAll** takes type **reader interface** as an argument

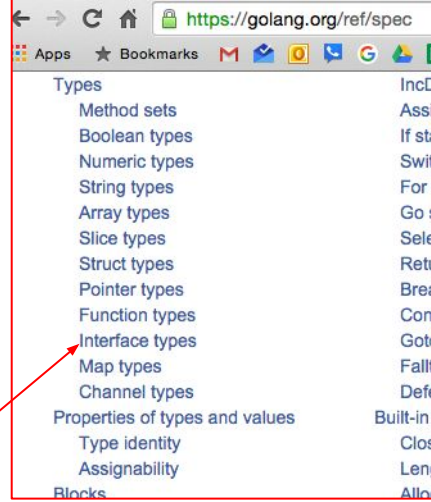
Reading A File

```go
package main

import (
    "log"
    "os"
    "io/ioutil"
    "fmt"
)

func main() {
    f, err := os.Open("hello.txt")
    if err != nil {
        log.Fatalln("my program broke")
    }
    defer f.Close()

    bs, err := ioutil.ReadAll(f)
    if err != nil {
        log.Fatalln("my program broke")
    }

    fmt.Println(bs)
    fmt.Println(string(bs))
}
```

```
Terminal
01_open-file $ go run main.go
[72 101 108 108 111]
Hello
01_open-file $
```

f
implements the
reader interface

ReadAll
takes a reader interface



an interface is a type

- f is type ***File** and type **reader interface**
- **ReadAll** takes type **reader interface** as an argument

```go
// Open opens the named file for reading.  If successful, methods on
// the returned file can be used for reading; the associated file
// descriptor has mode O_RDONLY.
// If there is an error, it will be of type *PathError.
func Open(name string) (file *File, err error) {
    return OpenFile(name, O_RDONLY, 0)
}
```

f has two types:
- *File
- reader interface

```go
// ReadAll reads from r until an error or EOF and returns the data it read.
// A successful call returns err == nil, not err == EOF. Because ReadAll is
// defined to read from src until EOF, it does not treat an EOF from Read
// as an error to be reported.
func ReadAll(r io.Reader) ([]byte, error) {
    return readAll(r, bytes.MinRead)
}
```

How does *File implement the reader interface?

main.go

```go
package main

import (
    "log"
    "os"
    "io/ioutil"
    "fmt"
)

func main() {
    f, err := os.Open("hello.txt")
    if err != nil {
        log.Fatalln("my program broke")
    }
    defer f.Close()

    bs, err := ioutil.ReadAll(f)
    if err != nil {
        log.Fatalln("my program broke")
    }

    fmt.Println(bs)
    fmt.Println(string(bs))
}
```

Terminal

```
01_open-file $ go run main.go
[72 101 108 108 111]
Hello
01_open-file $
```

godoc.org/os

type File

- func Create(name string) (*File, error)
- func NewFile(fd uintptr, name string) *File
- func Open(name string) (*File, error)
- func OpenFile(name string, flag int, perm FileMode) (*File, error)
- func Pipe() (r *File, w *File, err error)
- func (f *File) Chdir() error
- func (f *File) Chmod(mode FileMode) error
- func (f *File) Chown(uid, gid int) error
- func (f *File) Close() error
- func (f *File) Fd() uintptr
- func (f *File) Name() string
- func (f *File) Read(b []byte) (n int, err error)
- func (f *File) ...rror)
- func (f *File) ...
- func (f *File) ...err error)
- func (f *File) Seek(offset int64, whence int) (ret int64, err error)
- func (f *File) Stat() (FileInfo, error)
- func (f *File) Sync() error
- func (f *File) Truncate(size int64) error
- func (f *File) Write(b []byte) (n int, err error)
- func (f *File) WriteAt(b []byte, off int64) (n int, err error)
- func (f *File) WriteString(s string) (n int, err error)

methods of *File

```go
type Reader interface {
    Read(p []byte) (n int, err error)
}
```

*File implements the reader interface

Reader Interface

io.Reader

package io

```
import "io"
```

Package io provides basic interfaces to I/O primitives. Its primary job is to wrap existing implementations of such primitives, such as those in package os, into shared public interfaces that abstract the functionality, plus some other related primitives.

Because these interfaces and primitives wrap lower-level operations with various implementations, unless otherwise informed clients should not assume they are safe for parallel execution.

package io

```
import "io"
```

Package io provides basic interfaces to I/O primitives. Its primary job is to wrap existing implementations of such primitives, such as those in package os, into shared public interfaces that abstract the functionality, plus some other related primitives.

Because these interfaces and primitives wrap lower-level operations with various implementations, unless otherwise informed clients should not assume they are safe for parallel execution.

```
type Reader interface {
    Read(p []byte) (n int, err error)
}
```

package io

```
type Writer interface {
    Write(p []byte) (n int, err error)
}
```

package io

```
import "io"
```

Package io provides basic interfaces to I/O primitives. Its primary job is to wrap existing implementations of such primitives, such as those in package os into shared public interfaces that abstract the functionality, plus some other related primitives.

Because these interfaces and primitives wrap lower-level operations with various implementations, unless otherwise informed clients should not assume they are safe for parallel execution.

```
type Reader interface {
    Read(p []byte) (n int, err error)
}
```

package io

```
type Writer interface {
    Write(p []byte) (n int, err error)
}
```

godoc.org/os

type File

- func Create(name string) (*File, error)
- func NewFile(fd uintptr, name string) *File
- func Open(name string) (*File, error)
- func OpenFile(name string, flag int, perm FileMode) (*File, error)
- func Pipe() (r *File, w *File, err error)
- func (f *File) Chdir() error
- func (f *File) Chmod(mode FileMode) error
- func (f *File) Chown(uid, gid int) error
- func (f *File) Close() error
- func (f *File) Fd() uintptr
- func (f *File) Name() string
- func (f *File) Read(b []byte) (n int, err error)
- func (f *File) ReadAt(b []byte, off int64) (n int, err error)
- func (f *File) Readdir(n int) (fi []FileInfo, err error)
- func (f *File) Readdirnames(n int) (names []string, err error)
- func (f *File) Seek(offset int64, whence int) (ret int64, err error)
- func (f *File) Stat() (FileInfo, error)
- func (f *File) Sync() error
- func (f *File) Truncate(size int64) error
- func (f *File) Write(b []byte) (n int, err error)
- func (f *File) WriteAt(b []byte, off int64) (n int, err error)
- func (f *File) WriteString(s string) (n int, err error)

type FileInfo

type Reader

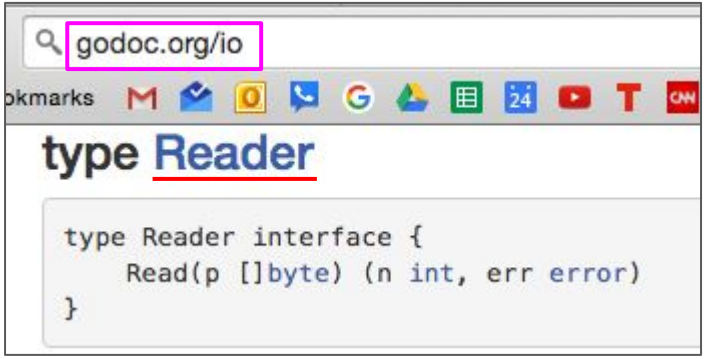
```
type Reader interface {
    Read(p []byte) (n int, err error)
}
```

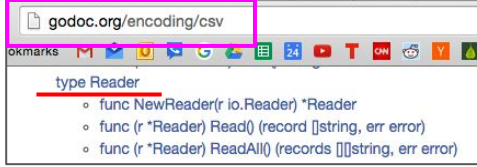
Any type with this method, implements reader interface

Reader is the interface that wraps the basic Read method.

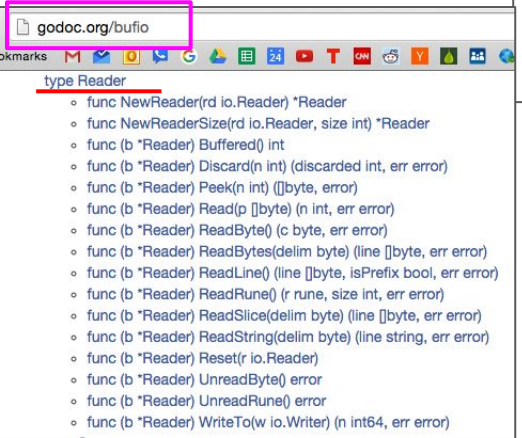
Read reads up to len(p) bytes into p. It returns the number of bytes read (0 <= n <= len(p)) and any error

Different packages can have funcs with the same name





Different packages can have funcs with the same name



What does it take for a type to implement the reader interface?

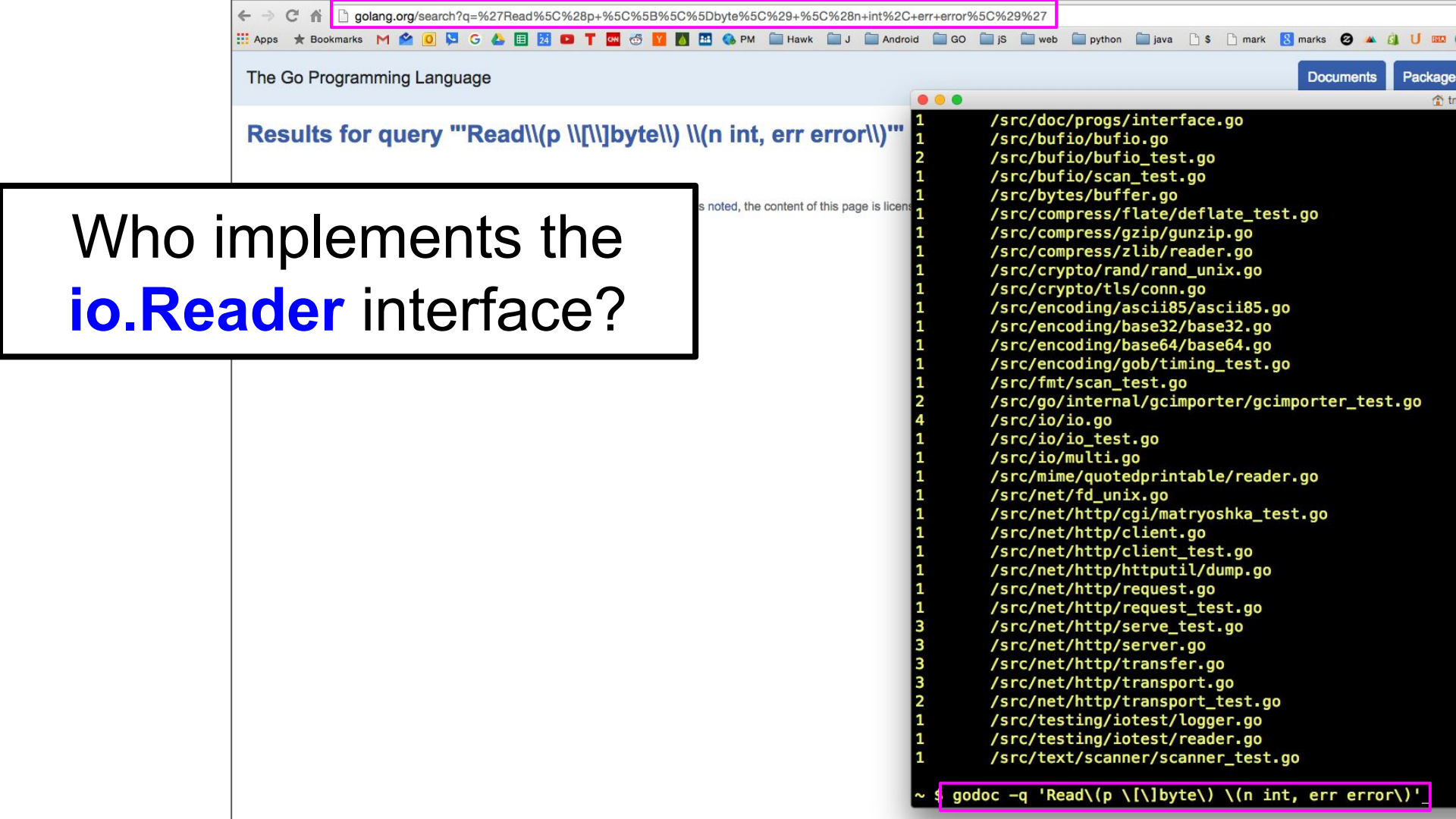
Reader Interface

io.Reader

What does it take for a type to implement the reader interface?

Reader Interface

io.Reader


golang.org/search?q=%27Read%5C%28p+%5C%5B%5C%5Dbyte%5C%29+%5C%28n+int%2C+err+error%5C%29%27
The Go Programming Language
Documents
Package
Results for query "'Read\\(p \\[\\]byte\\) \\(n int, err error\\)'"
Who implements the io.Reader interface?
1 /src/doc/progs/interface.go
1 /src/bufio/bufio.go
2 /src/bufio/bufio_test.go
1 /src/bufio/scan_test.go
1 /src/bytes/buffer.go
1 /src/compress/flate/deflate_test.go
1 /src/compress/gzip/gunzip.go
1 /src/compress/zlib/reader.go
1 /src/crypto/rand/rand_unix.go
1 /src/crypto/tls/conn.go
1 /src/encoding/ascii85/ascii85.go
1 /src/encoding/base32/base32.go
1 /src/encoding/base64/base64.go
1 /src/encoding/gob/timing_test.go
1 /src/fmt/scan_test.go
2 /src/go/internal/gcimporter/gcimporter_test.go
4 /src/io/io.go
1 /src/io/io_test.go
1 /src/io/multi.go
1 /src/mime/quotedprintable/reader.go
1 /src/net/fd_unix.go
1 /src/net/http/cgi/matryoshka_test.go
1 /src/net/http/client.go
1 /src/net/http/client_test.go
1 /src/net/http/httputil/dump.go
1 /src/net/http/request.go
1 /src/net/http/request_test.go
3 /src/net/http/serve_test.go
3 /src/net/http/server.go
3 /src/net/http/transfer.go
3 /src/net/http/transport.go
2 /src/net/http/transport_test.go
1 /src/testing/iotest/logger.go
1 /src/testing/iotest/reader.go
1 /src/text/scanner/scanner_test.go
~ godoc -q 'Read\(p \[\]byte\) \(n int, err error\)'

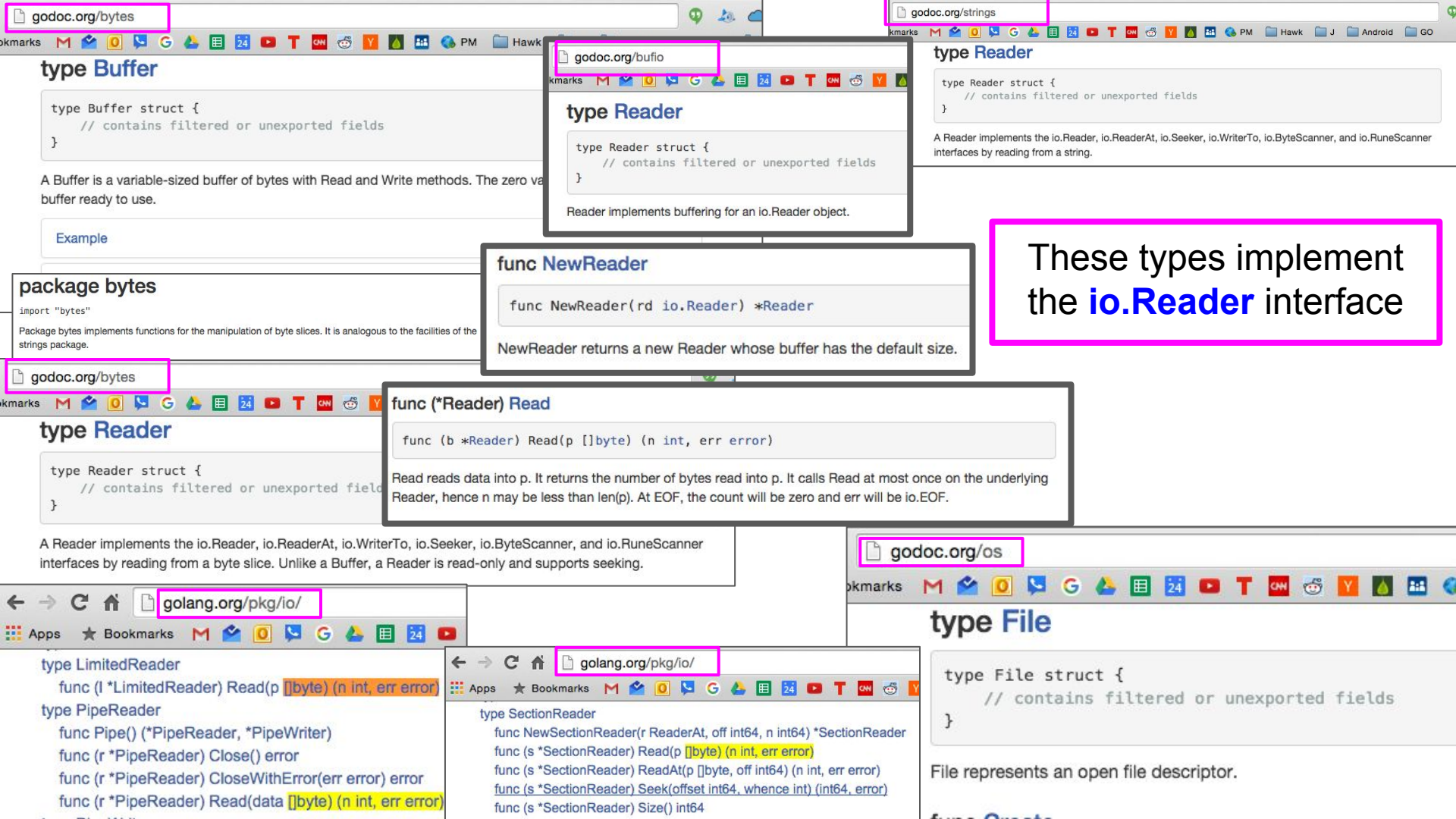
Who implements the **io.Reader** interface?

```
~ $ grep -r "Read(b \[\]byte) (n int, err error)" /usr/local/go/src
, err error) {
/usr/local/go/src/archive/tar/reader.go:func (rfr *regFileReader) Read(b []byte) (n int, err error)
/usr/local/go/src/archive/tar/reader.go:func (sfr *sparseFileReader) Read(b []byte) (n int, err error) {
/usr/local/go/src/archive/zip/reader.go:func (r *checksumReader) Read(b []byte) (n int, err error) {
/usr/local/go/src/bytes/reader.go:func (r *Reader) Read(b []byte) (n int, err error) {
/usr/local/go/src/compress/flate/deflate_test.go:func (r *sparseReader) Read(b []byte) (n int, err error) {
/usr/local/go/src/crypto/rand/rand.go:func Read(b []byte) (n int, err error) {
/usr/local/go/src/crypto/rand/rand_unix.go:func (r *devReader) Read(b []byte) (n int, err error) {
/usr/local/go/src/crypto/rand/rand_unix.go:func (r *reader) Read(b []byte) (n int, err error) {
/usr/local/go/src/crypto/rand/rand_windows.go:func (r *rngReader) Read(b []byte) (n int, err error) {
/usr/local/go/src/crypto/tls/conn.go:func (c *Conn) Read(b []byte) (n int, err error) {
/usr/local/go/src/crypto/tls/handshake_server_test.go:func (zeroSource) Read(b []byte) (n int, err error) {
/usr/local/go/src/crypto/tls/handshake_test.go:func (r *recordingConn) Read(b []byte) (n int, err error) {
/usr/local/go/src/encoding/gob/debug.go:func (p *peekReader) Read(b []byte) (n int, err error) {
/usr/local/go/src/fmt/scan.go:func (r *stringReader) Read(b []byte) (n int, err error) {
/usr/local/go/src/fmt/scan_test.go:func (ec *eofCounter) Read(b []byte) (n int, err error) {
/usr/local/go/src/fmt/scan_test.go:func (s *simpleReader) Read(b []byte) (n int, err error) {
/usr/local/go/src/image/jpeg/reader_test.go:func (r *eofReader) Read(b []byte) (n int, err error) {
/usr/local/go/src/mime/multipart/multipart_test.go:func (mr *maliciousReader) Read(b []byte) (n int, err error) {
/usr/local/go/src/net/fd_plan9.go:func (fd *netFD) Read(b []byte) (n int, err error) {
/usr/local/go/src/net/net.go:   Read(b []byte) (n int, err error)
/usr/local/go/src/net/textproto/reader.go:func (d *dotReader) Read(b []byte) (n int, err error) {
/usr/local/go/src/os/file.go:func (f *File) Read(b []byte) (n int, err error) {
/usr/local/go/src/strings/reader.go:func (r *Reader) Read(b []byte) (n int, err error) {
```

tm002 — bash — 172×45

bash

```
~ $ grep -r "Read(p \[\]byte) (n int, err error)" /usr/local/go/src
, err error) {
/usr/local/go/src/bufio/bufio_test.go:func (r *StringReader) Read(p []byte) (n int, err error) {
/usr/local/go/src/bufio/scan_test.go:func (sr *slowReader) Read(p []byte) (n int, err error) {
/usr/local/go/src/bytes/buffer.go:func (b *Buffer) Read(p []byte) (n int, err error) {
/usr/local/go/src/compress/flate/deflate_test.go:func (b *syncBuffer) Read(p []byte) (n int, err error) {
/usr/local/go/src/compress/gzip/gunzip.go:func (z *Reader) Read(p []byte) (n int, err error) {
/usr/local/go/src/compress/zlib/reader.go:func (z *reader) Read(p []byte) (n int, err error) {
/usr/local/go/src/crypto/tls/conn.go:func (b *block) Read(p []byte) (n int, err error) {
/usr/local/go/src/encoding/ascii85/ascii85.go:func (d *decoder) Read(p []byte) (n int, err error) {
/usr/local/go/src/encoding/base32/base32.go:func (d *decoder) Read(p []byte) (n int, err error) {
/usr/local/go/src/encoding/base64/base64.go:func (d *decoder) Read(p []byte) (n int, err error) {
/usr/local/go/src/encoding/gob/timing_test.go:func (b *benchmarkBuf) Read(p []byte) (n int, err error) {
/usr/local/go/src/fmt/scan_test.go:func (s *myStringReader) Read(p []byte) (n int, err error) {
/usr/local/go/src/io/io.go:     Read(p []byte) (n int, err error)
/usr/local/go/src/io/io.go:func (l *LimitedReader) Read(p []byte) (n int, err error) {
/usr/local/go/src/io/io.go:func (s *SectionReader) Read(p []byte) (n int, err error) {
/usr/local/go/src/io/io.go:func (t *teeReader) Read(p []byte) (n int, err error) {
/usr/local/go/src/io/io_test.go:func (r *dataAndErrorBuffer) Read(p []byte) (n int, err error) {
/usr/local/go/src/io/multi.go:func (mr *multiReader) Read(p []byte) (n int, err error) {
/usr/local/go/src/mime/multipart/quotedprintable.go:func (q *qpReader) Read(p []byte) (n int, err error) {
/usr/local/go/src/net/fd_unix.go:func (fd *netFD) Read(p []byte) (n int, err error) {
/usr/local/go/src/net/http/cgi/matryoshka_test.go:func (b neverEnding) Read(p []byte) (n int, err error) {
/usr/local/go/src/net/http/client.go:func (b *cancelTimerBody) Read(p []byte) (n int, err error) {
/usr/local/go/src/net/http/client_test.go:func (f eofReaderFunc) Read(p []byte) (n int, err error) {
/usr/local/go/src/net/http/httputil/dump.go:func (b neverEnding) Read(p []byte) (n int, err error) {
/usr/local/go/src/net/http/request.go:func (l *maxBytesReader) Read(p []byte) (n int, err error) {
/usr/local/go/src/net/http/serve_test.go:func (b neverEnding) Read(p []byte) (n int, err error) {
/usr/local/go/src/net/http/serve_test.go:func (cr countReader) Read(p []byte) (n int, err error) {
/usr/local/go/src/net/http/serve_test.go:func (r *repeatReader) Read(p []byte) (n int, err error) {
/usr/local/go/src/net/http/server.go:func (sr *liveSwitchReader) Read(p []byte) (n int, err error) {
/usr/local/go/src/net/http/server.go:func (ecr *expectContinueReader) Read(p []byte) (n int, err error) {
/usr/local/go/src/net/http/server.go:func (c *loggingConn) Read(p []byte) (n int, err error) {
/usr/local/go/src/net/http/transfer.go:func (r *errorReader) Read(p []byte) (n int, err error) {
/usr/local/go/src/net/http/transfer.go:func (b *body) Read(p []byte) (n int, err error) {
/usr/local/go/src/net/http/transfer.go:func (bl bodyLocked) Read(p []byte) (n int, err error) {
/usr/local/go/src/net/http/transport.go:func (es *bodyEOFSignal) Read(p []byte) (n int, err error) {
/usr/local/go/src/net/http/transport.go:func (gz *gzipReader) Read(p []byte) (n int, err error) {
/usr/local/go/src/net/http/transport.go:func (nr noteEOFReader) Read(p []byte) (n int, err error) {
/usr/local/go/src/net/http/transport_test.go:func (c byteFromChanReader) Read(p []byte) (n int, err error) {
/usr/local/go/src/net/mail/message.go:func (qd qDecoder) Read(p []byte) (n int, err error) {
/usr/local/go/src/net/timeout_test.go:func (b neverEnding) Read(p []byte) (n int, err error) {
/usr/local/go/src/testing/iotest/logger.go:func (l *readLogger) Read(p []byte) (n int, err error) {
/usr/local/go/src/testing/iotest/reader.go:func (r *dataErrReader) Read(p []byte) (n int, err error) {
/usr/local/go/src/text/scanner/scanner_test.go:func (r *StringReader) Read(p []byte) (n int, err error) {
```


godoc.org/bytes
type Buffer
type Buffer struct {
// contains filtered or unexported fields
}
A Buffer is a variable-sized buffer of bytes with Read and Write methods. The zero va
buffer ready to use.
Example
godoc.org/bufio
type Reader
type Reader struct {
// contains filtered or unexported fields
}
Reader implements buffering for an io.Reader object.
godoc.org/strings
type Reader
type Reader struct {
// contains filtered or unexported fields
}
A Reader implements the io.Reader, io.ReaderAt, io.Seeker, io.WriterTo, io.ByteScanner, and io.RuneScanner interfaces by reading from a string.
package bytes
import "bytes"
Package bytes implements functions for the manipulation of byte slices. It is analogous to the facilities of the strings package.
func NewReader
func NewReader(rd io.Reader) *Reader
NewReader returns a new Reader whose buffer has the default size.
These types implement the io.Reader interface
godoc.org/bytes
type Reader
type Reader struct {
// contains filtered or unexported field
}
A Reader implements the io.Reader, io.ReaderAt, io.WriterTo, io.Seeker, io.ByteScanner, and io.RuneScanner interfaces by reading from a byte slice. Unlike a Buffer, a Reader is read-only and supports seeking.
func (*Reader) Read
func (b *Reader) Read(p []byte) (n int, err error)
Read reads data into p. It returns the number of bytes read into p. It calls Read at most once on the underlying Reader, hence n may be less than len(p). At EOF, the count will be zero and err will be io.EOF.
golang.org/pkg/io/
type LimitedReader
func (l *LimitedReader) Read(p []byte) (n int, err error)
type PipeReader
func Pipe() (*PipeReader, *PipeWriter)
func (r *PipeReader) Close() error
func (r *PipeReader) CloseWithError(err error) error
func (r *PipeReader) Read(data []byte) (n int, err error)
golang.org/pkg/io/
type SectionReader
func NewSectionReader(r ReaderAt, off int64, n int64) *SectionReader
func (s *SectionReader) Read(p []byte) (n int, err error)
func (s *SectionReader) ReadAt(p []byte, off int64) (n int, err error)
func (s *SectionReader) Seek(offset int64, whence int) (int64, error)
func (s *SectionReader) Size() int64
godoc.org/os
type File
type File struct {
// contains filtered or unexported fields
}
File represents an open file descriptor.

https://golang.org/src/strings/reader.go#L37

```
func (r *Reader) Read(b []byte) (n int, err error) {
        if len(b) == 0 {
                return 0, nil
        }
        if r.i >= int64(len(r.s)) {
                return 0, io.EOF
        }
        r.prevRune = -1
        n = copy(b, r.s[r.i:])
        r.i += int64(n)
        return
}
```

All very different code ...

… yet all of this code begins and ends in the same way ...

```
func (f *File) Read(b []byte) (n int, err error) {
        if f == nil {
                return 0, ErrInvalid
        }
        n, e := f.read(b)
        if n < 0 {
                n = 0
        }
        if n == 0 && len(b) > 0 && e == nil {
                return 0, io.EOF
        }
        if e != nil {
                err = &PathError{"read", f.name, e}
        }
        return n, err
}
```

https://golang.org/src/bufio/bufio.go#L185

```
func (b *Reader) Read(p []byte) (n int, err error) {
        n = len(p)
        if n == 0 {
                return 0, b.readErr()
        }
        if b.r == b.w {
                if b.err != nil {
                        return 0, b.readErr()
                }
                if len(p) >= len(b.buf) {
```

… take a slice of bytes and read it, then return the number of bytes read and error status ...

https://golang.org/src/bytes/reader.go#L38

```
func (r *Reader) Read(b []byte) (n int, err error) {
        if len(b) == 0 {
                return 0, nil
        }
        if r.i >= int64(len(r.s)) {
                return 0, io.EOF
        }
        r.prevRune = -1
        n = copy(b, r.s[r.i:])
        r.i += int64(n)
        return
}
```

… the **interface** is this commonality …
… there is a common **interface** …

… any func that knows how to use this read method ... can receive any of these types which have this read method …

… we create an **interface** type, which all of these other types can match if they have the common **interface,** in this case, the read method …

this is how interfaces work … an interface has been created ...

https://golang.org/src/bytes/buffer.go#L255

```
nc (b *Buffer) Read(p []byte) (n int, err error) {
        b.lastRead = opInvalid
        if b.off >= len(b.buf) {
                // Buffer is empty, reset to recover space.
                b.Truncate(0)
                if len(p) == 0 {
                        return
                }
                return 0, io.EOF
        }
        n = copy(p, b.buf[b.off:])
        b.off += n
        if n > 0 {
                b.lastRead = opRead
        }
        return
}
```

https://golang.org/src/strings/reader.go#L37

```
func (r *Reader) Read(b []byte) (n int, err error) {
        if len(b) == 0 {
                return 0, nil
        }
        if r.i >= int64(len(r.s)) {
                return 0, io.EOF
        }
        r.prevRune = -1
        n = copy(b, r.s[r.i:])
        r.i += int64(n)
        return
}
```

All very different code ...

… yet all of this code begins and ends in the same way ...

https://golang.org/src/bufio/bufio.go#L185

```
func (b *Reader) Read(p []byte) (n int, err error) {
        n = len(p)
        if n == 0 {
                return 0, b.readErr()
        }
        if b.r == b.w {
                if b.err != nil {
                        return 0, b.readErr()
                }
                if len(p) >= len(b.buf) {
```

```
func (f *File) Read(b []byte) (n int, err error) {
        if f == nil {
                return 0, ErrInvalid
        }
        n, e := f.read(b)
        if n < 0 {
                n = 0
        }
        if n == 0 && len(b) > 0 && e == nil {
                return 0, io.EOF
        }
        if e != nil {
                err = &PathError{"read", f.name, e}
        }
        return n, err
}
```

… take a slice of bytes and read it, then return the number of bytes read and error status ...

https://golang.org/src/bytes/buffer.go#L255

```
nc (b *Buffer) Read(p []byte) (n int, err error) {
        b.lastRead = opInvalid
        if b.off >= len(b.buf) {
```

https://golang.org/src/bytes/reader.go#L38

```
func (r *Reader) Read(b []byte) (n int, err error) {
        if len(b) == 0 {
                return 0, nil
        }
        if r.i >= int64(len(r.s)) {
                return 0, io.EOF
        }
        r.prevRune = -1
        n = copy(b, r.s[r.i:])
        r.i += int64(n)
        return
}
```

… the **interface** is this commonality …
… there is a common **interface** …

… any func that knows how to use this read method ... can receive any of these types which have this read method …

… we create an **interface** type, which all of these other types can match if they have the common **interface,** in this case, the read method …

this is how interfaces work … an interface has been created ...

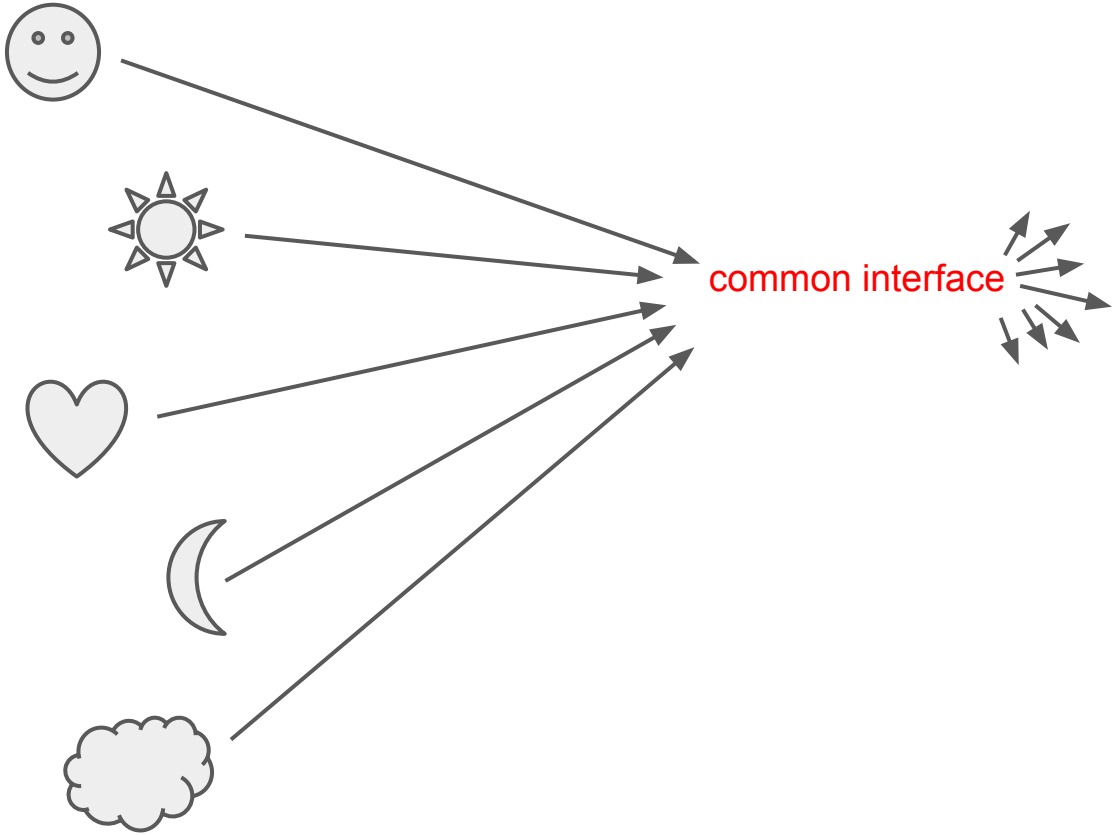
godoc.org/io

type Reader

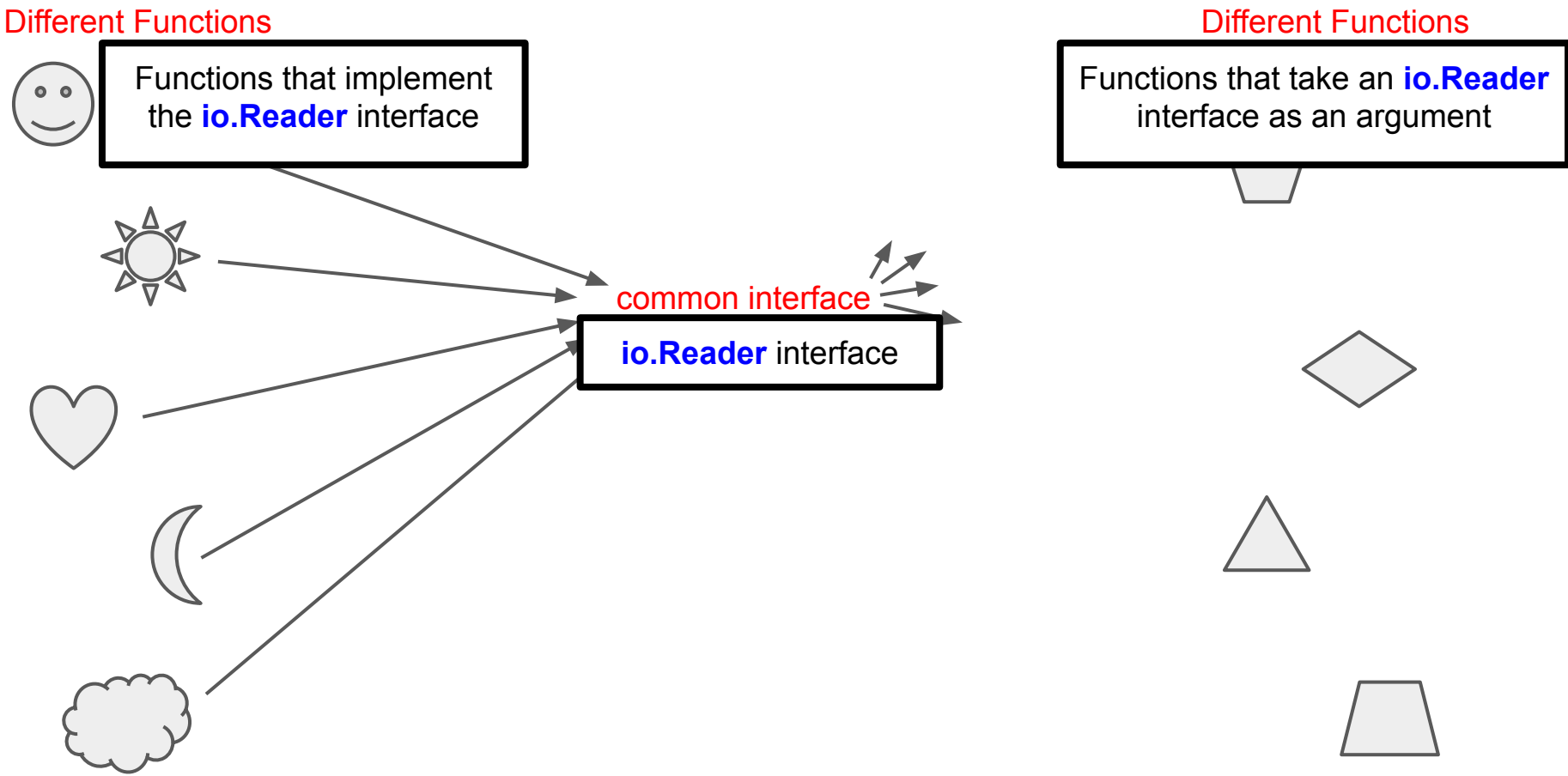
```
type Reader interface {
    Read(p []byte) (n int, err error)
}
```

Different Functions

Different Functions



any of the functions on the left
can be matched with any of the functions on the right
because all of the functions share a common, agreed upon way of interfacing with each other



any of the functions on the left
can be matched with any of the functions on the right
because all of the functions share a common, agreed upon way of interfacing with each other

Different Functions

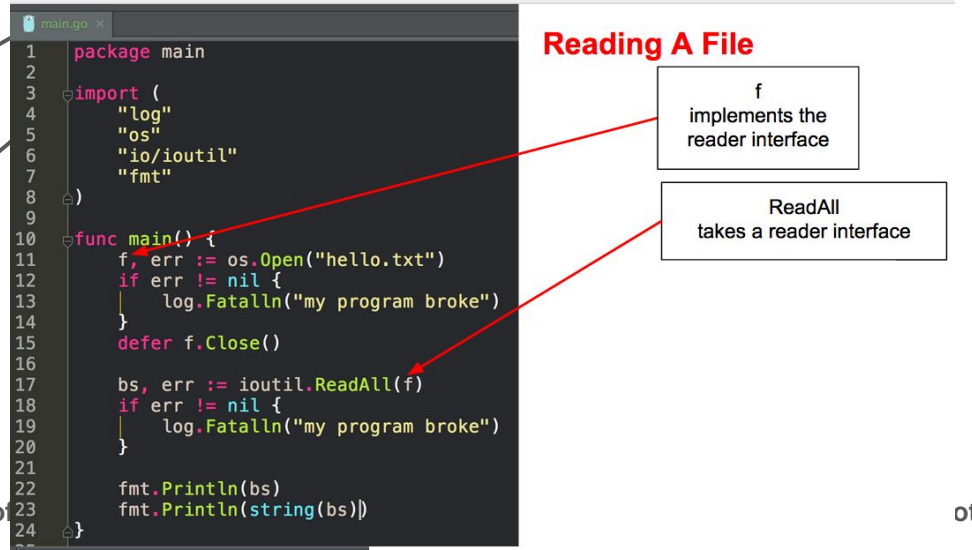
Functions that implement the **io.Reader** interface

Different Functions

Functions that take an **io.Reader** interface as an argument

common interface

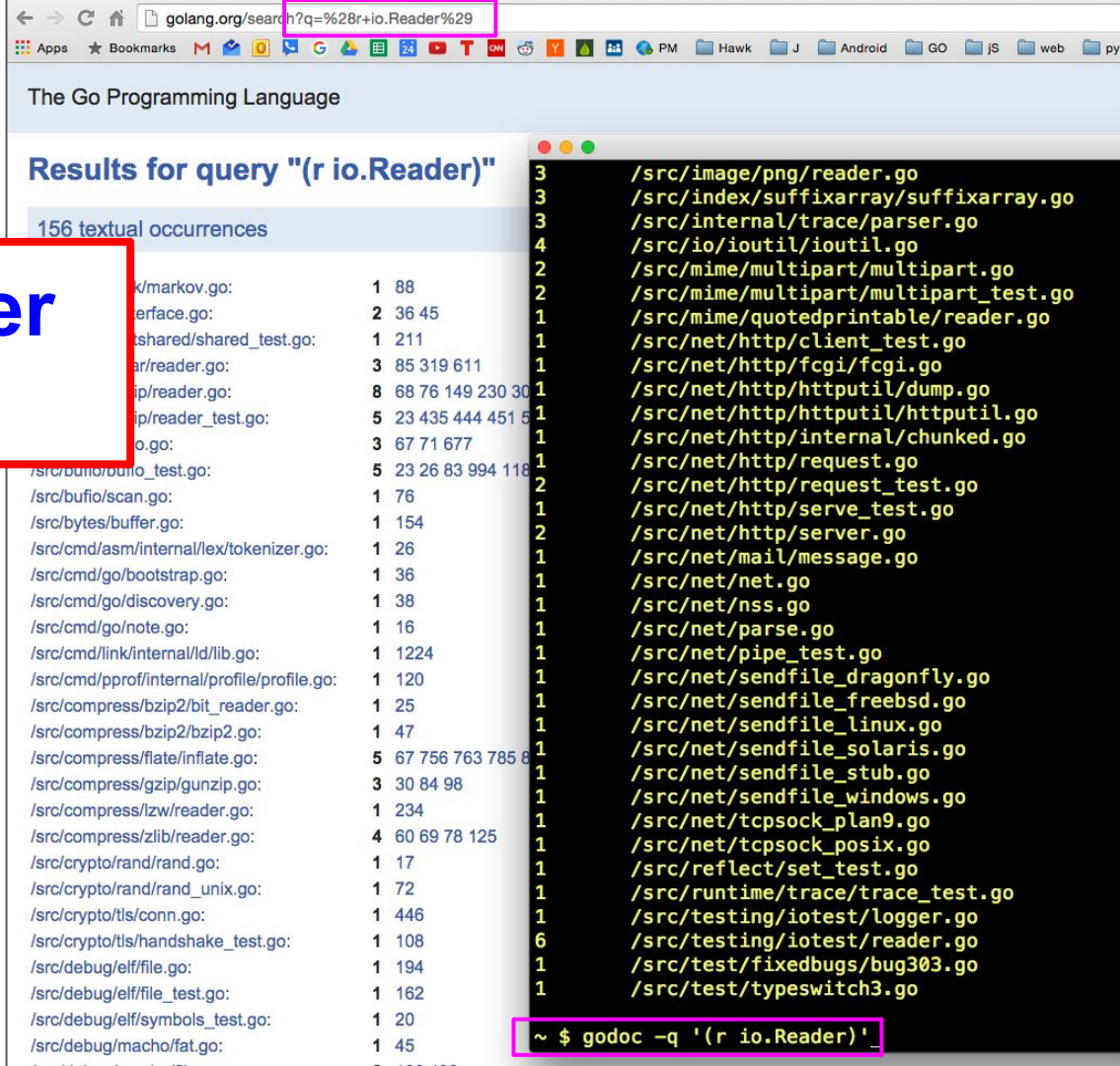
io.Reader interface

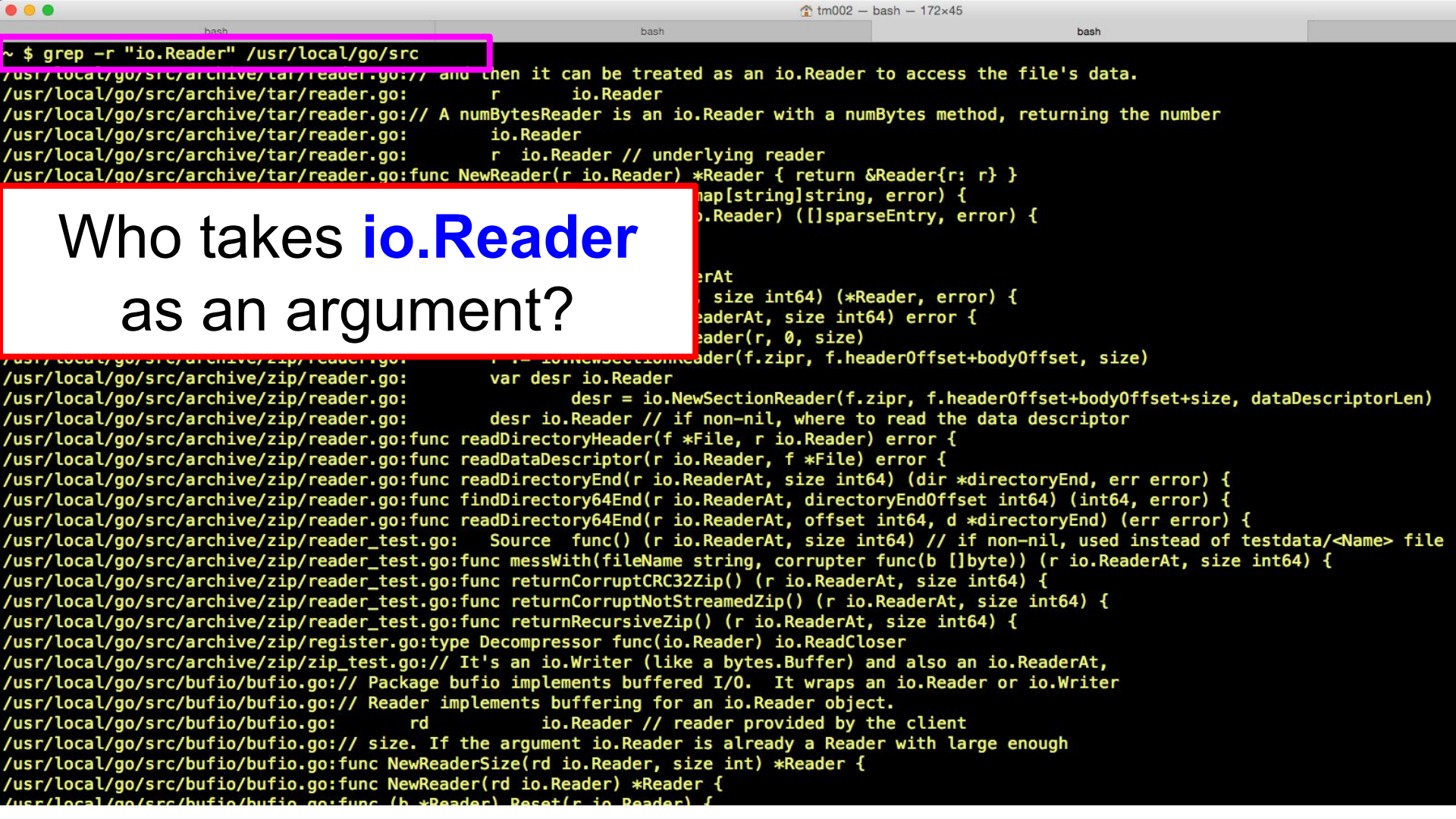


because all of

other

Who takes **io.Reader** as an argument?




tm002 — bash — 172×45
bash
bash
bash
~ $ grep -r "io.Reader" /usr/local/go/src
/usr/local/go/src/archive/tar/reader.go:// and then it can be treated as an io.Reader to access the file's data.
/usr/local/go/src/archive/tar/reader.go:	r	io.Reader
/usr/local/go/src/archive/tar/reader.go:// A numBytesReader is an io.Reader with a numBytes method, returning the number
/usr/local/go/src/archive/tar/reader.go:	io.Reader
/usr/local/go/src/archive/tar/reader.go:	r io.Reader // underlying reader
/usr/local/go/src/archive/tar/reader.go:func NewReader(r io.Reader) *Reader { return &Reader{r: r} }
map[string]string, error) {
.Reader) ([]sparseEntry, error) {
erAt
size int64) (*Reader, error) {
eaderAt, size int64) error {
eader(r, 0, size)
ader(f.zipr, f.headerOffset+bodyOffset, size)
/usr/local/go/src/archive/zip/reader.go:	var desr io.Reader
/usr/local/go/src/archive/zip/reader.go:		desr = io.NewSectionReader(f.zipr, f.headerOffset+bodyOffset+size, dataDescriptorLen)
/usr/local/go/src/archive/zip/reader.go:	desr io.Reader // if non-nil, where to read the data descriptor
/usr/local/go/src/archive/zip/reader.go:func readDirectoryHeader(f *File, r io.Reader) error {
/usr/local/go/src/archive/zip/reader.go:func readDataDescriptor(r io.Reader, f *File) error {
/usr/local/go/src/archive/zip/reader.go:func readDirectoryEnd(r io.ReaderAt, size int64) (dir *directoryEnd, err error) {
/usr/local/go/src/archive/zip/reader.go:func findDirectory64End(r io.ReaderAt, directoryEndOffset int64) (int64, error) {
/usr/local/go/src/archive/zip/reader.go:func readDirectory64End(r io.ReaderAt, offset int64, d *directoryEnd) (err error) {
/usr/local/go/src/archive/zip/reader_test.go:	Source func() (r io.ReaderAt, size int64) // if non-nil, used instead of testdata/<Name> file
/usr/local/go/src/archive/zip/reader_test.go:func messWith(fileName string, corrupter func(b []byte)) (r io.ReaderAt, size int64) {
/usr/local/go/src/archive/zip/reader_test.go:func returnCorruptCRC32Zip() (r io.ReaderAt, size int64) {
/usr/local/go/src/archive/zip/reader_test.go:func returnCorruptNotStreamedZip() (r io.ReaderAt, size int64) {
/usr/local/go/src/archive/zip/reader_test.go:func returnRecursiveZip() (r io.ReaderAt, size int64) {
/usr/local/go/src/archive/zip/register.go:type Decompressor func(io.Reader) io.ReadCloser
/usr/local/go/src/archive/zip/zip_test.go:// It's an io.Writer (like a bytes.Buffer) and also an io.ReaderAt,
/usr/local/go/src/bufio/bufio.go:// Package bufio implements buffered I/O. It wraps an io.Reader or io.Writer
/usr/local/go/src/bufio/bufio.go:// Reader implements buffering for an io.Reader object.
/usr/local/go/src/bufio/bufio.go:	rd		io.Reader // reader provided by the client
/usr/local/go/src/bufio/bufio.go:// size. If the argument io.Reader is already a Reader with large enough
/usr/local/go/src/bufio/bufio.go:func NewReaderSize(rd io.Reader, size int) *Reader {
/usr/local/go/src/bufio/bufio.go:func NewReader(rd io.Reader) *Reader {
Who takes io.Reader
as an argument?

```
// Fscan scans text read from r, storing successive space-separated
// values into successive arguments.  Newlines count as space.  It
// returns the number of items successfully scanned.  If that is less
// than the number of arguments, err will report why.
func Fscan(r io.Reader, a ...interface{}) (n int, err error) {
        s, old := newScanState(r, true, false)
        n, err = s.doScan(a)
        s.free(old)
        return
}

// Fscanln is similar to Fscan, but stops scanning at a
// after the final item there must be a newline or EOF.
func Fscanln(r io.Reader, a ...interface{}) (n int, err
        s, old := newScanState(r, false, true)
        n, err = s.doScan(a)
        s.free(old)
        return
}

// Fscanf scans text read from r, storing successi
// values into successive arguments as determined
// returns the number of items successfully parse
// Newlines in the input must match newlines in th
func Fscanf(r io.Reader, format string, a ...inter
        s, old := newScanState(r, false, false)
        n, err = s.doScanf(format, a)
        s.free(old)
        return
}
```

```
// NewScanner returns a new Scanner to read from r.
// The split function defaults to ScanLines.
func NewScanner(r io.Reader) *Scanner {
        return &Scanner{
                r:            r,
```

```
// Reset discards any buffered data, resets all sta
// the buffered reader to read from r.
func (b *Reader) Reset(r io.Reader) {
        b.reset(b.buf, r)
}
```

```
// readAll reads from r until an error or EOF and returns the data it read
// from the internal buffer allocated with a specified capacity.
func readAll(r io.Reader, capacity int64) (b []byte, err error) {
        buf := bytes.NewBuffer(make([]byte, 0, capacity))
        // If the buffer overflows, we will get bytes.ErrTooLarge.
        // Return that as an error. Any other panic remains.
        defer func() {
```

```
// ReadAll reads from r until an error or EOF and returns the data it read.
// A successful call returns err == nil, not err == EOF. Because ReadAll is
// defined to read from src until EOF, it does not treat an EOF from Read
// as an error to be reported.
func ReadAll(r io.Reader) ([]byte, error) {
        return readAll(r, bytes.MinRead)
}
```

golang.org/pkg/net/http/#Request

Apps ★ Bookmarks

```
type Request
    func NewRequest(method, urlStr string, body io.Reader) (*Request, error)
    func ReadRequest(b *bufio.Reader) (req *Request, err error)
    func (r *Request) AddCookie(c *Cookie)
    func (r *Request) BasicAuth() (username, password string, ok bool)
    func (r *Request) Cookie(name string) (*Cookie, error)
    func (r *Request) Cookies() []*Cookie
    func (r *Request) FormFile(key string) (multipart.File, *multipart.FileHeader, error)
    func (r *Request) FormValue(key string) string
    func (r *Request) MultipartReader() (*multipart.Reader, error)
    func (r *Request) ParseForm() error
    func (r *Request) ParseMultipartForm(maxMemory int64) error
    func (r *Request) PostFormValue(key string) string
    func (r *Request) ProtoAtLeast(major, minor int) bool
    func (r *Request) Referer() string
    func (r *Request) SetBasicAuth(username, password string)
    func (r *Request) UserAgent() string
    func (r *Request) Write(w io.Writer) error
    func (r *Request) WriteProxy(w io.Writer) error
type Response
    func Get(url string) (resp *Response, err error)
    func Head(url string) (resp *Response, err error)
    func Post(url string, bodyType string, body io.Reader) (resp *Response, err error)
    func PostForm(url string, data url.Values) (resp *Response, err error)
```

```
// ReadFrom reads data from r until EOF and appends it to the buffer, growing
// the buffer as needed. The return value n is the number of bytes read. Any
// error except io.EOF encountered during the read is also returned. If the
// buffer becomes too large, ReadFrom will panic with ErrTooLarge.
func (b *Buffer) ReadFrom(r io.Reader) (n int64, err error) {
        b.lastRead = opInvalid
        // If buffer is empty, reset to recover spac
        if b.off >= len(b.buf) {
```

```
func (b *Reader) reset(buf []byte, r io.Reader) {
        *b = Reader{
                buf:          buf,
                r,
                tByte:       -1,
                tRuneSize: -1,
}
```

```
// DecodeAll reads a GIF image from r and returns the sequential frames
// and timing information.
func DecodeAll(r io.Reader) (*GIF, error) {
        var d decoder
        if err := d.decode(r, false); err != nil {
                return nil, err
```

```
func (b *Writer) ReadFrom(r io.Reader) (n int64, err error) {
        if b.Buffered() == 0 {
                if w, ok := b.wr.(io.ReaderFrom); ok {
                        return w.ReadFrom(r)
                }
        }
        var m int
```

```
// Decode decodes an image that has been encoded in a registered format.
// The string returned is the format name used during format registration.
// Format registration is typically done by an init function in the codec-
// specific package.
func Decode(r io.Reader) (Image, string, error) {
        rr := asReader(r)
```

```go
func main() {
    f, err := os.Open("hello.txt")
    if err != nil {
        log.Fatalln("my program broke")
    }
    defer f.Close()

    bs, err := ioutil.ReadAll(f)
    if err != nil {
        log.Fatalln("my program broke")
    }

    fmt.Println(bs)
    fmt.Println(string(bs))
}
```

```go
func ReadAll(r io.Reader) ([]byte, error) {
    return readAll(r, bytes.MinRead)
}
```

```go
type Reader interface {
    Read(p []byte) (n int, err error)
}
```

```go
func readAll(r io.Reader, capacity int64) (b []byte, err error) {
    buf := bytes.NewBuffer(make([]byte, 0, capacity))
    // If the buffer overflows, we will get bytes.ErrTooLarge.
    // Return that as an error. Any other panic remains.
    defer func() {
        e := recover()
        if e == nil {
            return
        }
        if panicErr, ok := e.(error); ok && panicErr == bytes.ErrTooLarge {
            err = panicErr
        } else {
            panic(e)
        }
    }()
    _, err = buf.ReadFrom(r)
    return buf.Bytes(), err
}
```

```go
func (b *Buffer) ReadFrom(r io.Reader) (n int64, err error) {
    b.lastRead = opInvalid
    // If buffer is empty, reset to recover space.
    if b.off >= len(b.buf) {
        b.Truncate(0)
    }
    for {
        if free := cap(b.buf) - len(b.buf); free < MinRead {
            // not enough space at end
            newBuf := b.buf
            if b.off+free < MinRead {
                // not enough space using beginning of buffer;
                // double buffer capacity
                newBuf = makeSlice(2*cap(b.buf) + MinRead)
            }
            copy(newBuf, b.buf[b.off:])
            b.buf = newBuf[:len(b.buf)-b.off]
            b.off = 0
        }
        m, e := r.Read(b.buf[len(b.buf):cap(b.buf)])
        b.buf = b.buf[0 : len(b.buf)+m]
        n += int64(m)
        if e == io.EOF {
            break
        }
        if e != nil {
            return n, e
        }
    }
    return n, nil // err is EOF, so return nil explicitly
}
```

Eventually it all gets down to that read method being used

grep

globally search a **r**egular **e**xpression and **p**rint)

```
~ $ man grep_
```

tm002 — less — 172×37

bash | bash | bash | less

```
GREP(1)                  BSD General Commands Manual                  GREP(1)

NAME
     grep, egrep, fgrep, zgrep, zegrep, zfgrep -- file pattern searcher

SYNOPSIS
     grep [-abcdDEFGHhIiJLlmnOopqRSsUVvwxZ] [-A num] [-B num] [-C[num]] [-e pattern] [-f file] [--binary-files=value] [--color[=when]] [--colour[=when]]
          [--context[=num]] [--label] [--line-buffered] [--null] [pattern] [file ...]

DESCRIPTION
     The grep utility searches any given input files, selecting lines that match one or more patterns.  By default, a pattern matches an input line if the
     regular expression (RE) in the pattern matches the input line without its trailing newline.  An empty expression matches every line.  Each input line
     that matches at least one of the patterns is written to the standard output.

     grep is used for simple patterns and basic regular expressions (BREs); egrep can handle exte
     more information on regular expressions.  fgrep is quicker than both grep and egrep, but can
     regular expressions).  Patterns may consist of one or more lines, allowing any of the patter

     zgrep, zegrep, and zfgrep act like grep, egrep, and fgrep, respectively, but accept input fi
     sion utilities.

     The following options are available:

     -A num, --after-context=num
             Print num lines of trailing context after each match.  See also the -B and -C option

     -a, --text
             Treat all files as ASCII text.  Normally grep will simply print ``Binary file ... ma
             option forces grep to output lines matching the specified pattern.

     -B num, --before-context=num
             Print num lines of leading context before each match.  See also the -A and -C option

     -b, --byte-offset
             The offset in bytes of a matched pattern is displayed in front of the respective mat
```

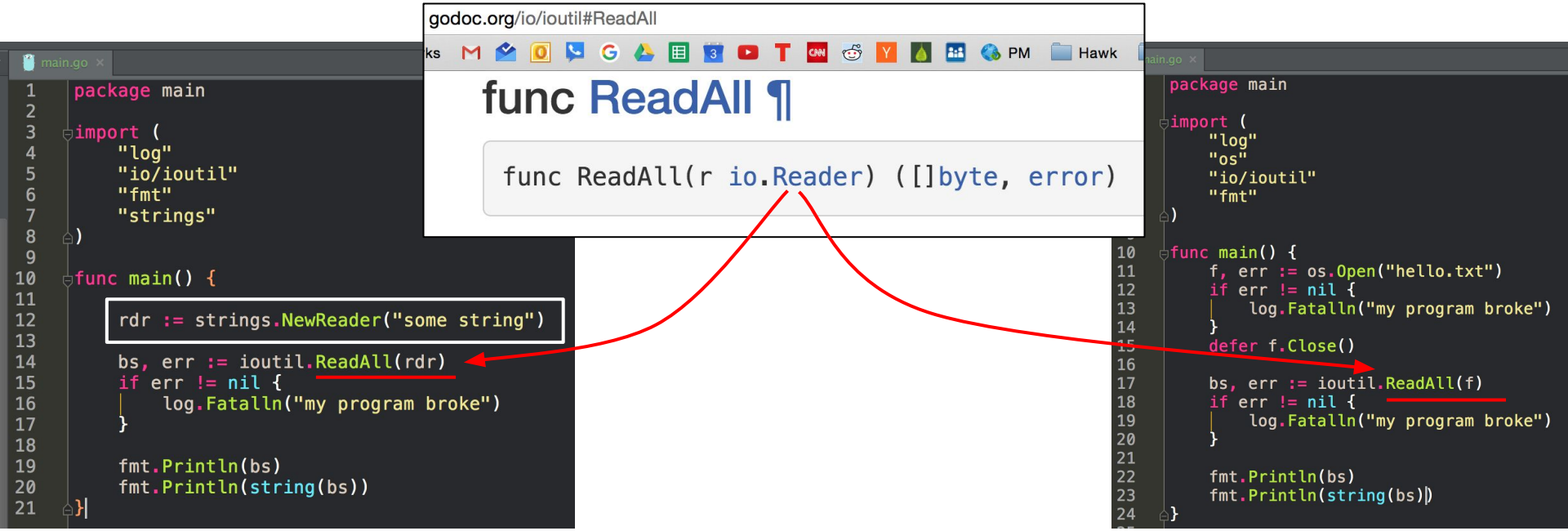
grep

From Wikipedia, the free encyclopedia

grep is a command-line utility for searching plain-text data sets for lines matching a regular expression. Grep was originally developed for the Unix operating system, but is available today for all Unix-like systems. Its name comes from the ed command *g/re/p* (*globally search a regular expression* and *print*), which has the same effect: doing a global search with the regular expression and printing all matching lines.[3][4]

strings.NewReader

This illustrates the flexibility of interfaces



func NewReader

```
func NewReader(s string) *Reader
```

NewReader returns a new Reader reading from s. It is similar to bytes.NewBufferString but more efficient and read-only.

func Open

```
func Open(name string) (*File, error)
```

Open opens the named file for reading. If successful, methods on the returned file can be used for reading; the associated file descriptor has mode O_RDONLY. If there is an error, it will be of type *PathError.

godoc.org/os

type File

- func Create(name string) (*File, error)
- func NewFile(fd uintptr, name string) *File
- func Open(name string) (*File, error)
- func OpenFile(name string, flag int, perm FileMode) (*File, error)
- func Pipe() (r *File, w *File, err error)
- func (f *File) Chdir() error
- func (f *File) Chmod(mode FileMode) error
- func (f *File) Chown(uid, gid int) error
- func (f *File) Close() error
- func (f *File) Fd() uintptr
- func (f *File) Name() string
- func (f *File) Read(b []byte) (n int, err error)
- func (f *File) ReadAt(b []byte, off int64) (n int, err error)
- func (f *File) Readdir(n int) (fi []FileInfo, err error)
- func (f *File) Readdirnames(n int) (names []string, err error)
- func (f *File) Seek(offset int64, whence int) (ret int64, err error)
- func (f *File) Stat() (FileInfo, error)
- func (f *File) Sync() error
- func (f *File) Truncate(size int64) error
- func (f *File) Write(b []byte) (n int, err error)
- func (f *File) WriteAt(b []byte, off int64) (n int, err error)
- func (f *File) WriteString(s string) (n int, err error)

methods

godoc.org/strings

type Reader

- func NewReader(s string) *Reader
- func (r *Reader) Len() int
- func (r *Reader) Read(b []byte) (n int, err error)
- func (r *Reader) ReadAt(b []byte, off int64) (n int, err error)
- func (r *Reader) ReadByte() (b byte, err error)
- func (r *Reader) ReadRune() (ch rune, size int, err error)
- func (r *Reader) Seek(offset int64, whence int) (int64, error)
- func (r *Reader) Size() int64
- func (r *Reader) UnreadByte() error
- func (r *Reader) UnreadRune() error
- func (r *Reader) WriteTo(w io.Writer) (n int64, err error)

Readers

- *os.File
  - **Read**
    - reads up to len(b) bytes from the File
    - func (f *File) Read(**b []byte**) (n int, err error)
  - Readdirnames
    - reads and returns a slice of names from the directory f
    - func (f *File) Readdirnames(n int) (names []string, err error)
- io/ioutil
  - **ReadAll**
    - reads from r until an error or EOF and returns the data it read
    - func ReadAll(**r io.Reader**) ([]byte, error)
  - Readdir
    - reads the contents of the directory associated with file and returns a slice of up to n FileInfo values
    - func (f *File) Readdir(n int) (fi []FileInfo, err error)
  - Readdirnames
    - reads and returns a slice of names from the directory f.
    - func (f *File) Readdirnames(n int) (names []string, err error)

```
func main() {
    src, err := os.Open("src.txt")
    if err != nil {
        panic(err)
    }
    defer src.Close()

    dst, err := os.Create("dst.txt")
    if err != nil {
        panic(err)
    }
    defer dst.Close()

    bs := make([]byte, 5)
    src.Read(bs)
    dst.Write(bs)
}
```

```
func main() {
    f, err := os.Open(os.Args[1])
    if err != nil {
        log.Fatalln("my program broke"
    }
    defer f.Close()

    bs, err := ioutil.ReadAll(f)
    if err != nil {
        log.Fatalln("my program broke
    }

    str := string(bs)
    fmt.Println(str)
}
```

Readers

- *strings.Reader
  - **Read**
    - reads up to len(b) bytes from the File
    - func (r *Reader) Read(b []byte) (n int, err error)
- *bufio.Reader
  - **Read**
    - Read reads data into p
    - func (b *Reader) Read(p []byte) (n int, err error)
- io
  - Copy
    - Copy copies from src to dst until either EOF is reached on src or an error occurs
    - func Copy(dst Writer, src Reader) (written int64, err error)
- *csv.Reader
  - Read
    - Read reads one record from r. The record is a slice of strings with each string representing one field.
    - func (r *Reader) Read() (record []string, err error)

```
func main() {

    dst, err := os.Create(os.Args[1])
    if err != nil {
        log.Fatalf("error creating destination file:%v ", err)
    }
    defer dst.Close()

    rdr := strings.NewReader("hello world")

    io.Copy(dst, rdr)
}
```

main.go

```go
package main

import (
    "log"
    "io/ioutil"
    "fmt"
    "strings"
)

func main() {

    rdr := strings.NewReader("some string")

    bs, err := ioutil.ReadAll(rdr)
    if err != nil {
        log.Fatalln("my program broke")
    }

    fmt.Println(bs)
    fmt.Println(string(bs))
}
```

Just FYI

We create a NewReader then pass it into ReadAll

Terminal

```
03_strings-newReader $ go run main
[115 111 109 101 32 115 116 114 10
some string
03_strings-newReader $
```

godoc.org/io/ioutil#ReadAll

func ReadAll ¶

```
func ReadAll(r io.Reader) ([]byte, error)
```

ReadAll takes an Reader interface

```
type Reader interface {
    Read(p []byte) (n int, err error)
}
```

a pointer to a strings.Reader implements io.Reader interface

type Reader

- func NewReader(s string) *Reader
- func (r *Reader) Len() int
- func (r *Reader) Read(b []byte) (n int, err error)
- func (r *Reader) ReadAt(b []byte, off int64) (n int, err error)

NewReader returns a pointer to a strings.Reader

func NewReader

```
func NewReader(s string) *Reader
```

NewReader returns a new Reader reading from s. It is similar to bytes.NewBufferString but more efficient and read-only.

Writer Interface

io.writer

```go
func main() {

    dst, err := os.Create(os.Args[1])
    if err != nil {
        log.Fatalf("error creating destination file:%v ", err)
    }
    defer dst.Close()

    dst.Write([]byte("Hello World"))
}
```

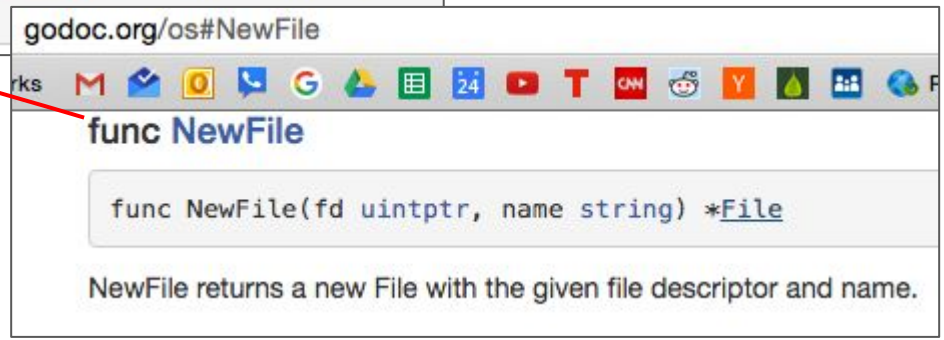
```go
func main() {

    dst, err := os.Create("hello.txt")
    if err != nil {
        log.Fatalln("error creating destination file: ", err.Error())
    }
    defer dst.Close()

    bs := []byte("Put some phrase here.")

    _, err = dst.Write(bs)
    if err != nil {
        log.Fatalln("error writing to file: ", err.Error())
    }
}
```

```go
func main() {

    f, err := os.Open(os.Args[1:])
    if err != nil {
        log.Fatalln("my program broke: ", err.Error())
    }
    defer f.Close()

    io.Copy(os.Stdout, f)
}
```

```go
func main() {

    err := ioutil.WriteFile("hello.txt", []byte("Hello world"), 0777)
    if err != nil {
        panic("something went wrong")
    }
}
```

```go
func main() {

    f, err := os.Open(os.Args[1:])
    if err != nil {
        log.Fatalln("my program broke: ", err.Error())
    }
    defer f.Close()

    io.Copy(os.Stdout, f)
}
```

```go
var (
    Stdin  = NewFile(uintptr(syscall.Stdin), "/dev/stdin")
    Stdout = NewFile(uintptr(syscall.Stdout), "/dev/stdout")
    Stderr = NewFile(uintptr(syscall.Stderr), "/dev/stderr")
)
```



Does a *File implement the writer interface?

```
type Writer interface {
    Write(p []byte) (n int, err error)
}
```

godoc.org/os

type File

- func Create(name string) (*File, error)
- func NewFile(fd uintptr, name string) *File
- func Open(name string) (*File, error)
- func OpenFile(name string, flag int, perm FileMode) (*File, error)
- func Pipe() (r *File, w *File, err error)
- func (f *File) Chdir() error
- func (f *File) Chmod(mode FileMode) error
- func (f *File) Chown(uid, gid int) error
- func (f *File) Close() error
- func (f *File) Fd() uintptr
- func (f *File) Name() string
- func (f *File) Read(b []byte) (n int, err error)
- func (f *File) ReadAt(b []byte, off int64) (n int, err error)
- func (f *File) Readdir(n int) (fi []FileInfo, err error)
- func (f *File) Readdirnames(n int) (names []string, err error)
- func (f *File) Seek(offset int64, whence int) (ret int64, err error)
- func (f *File) Stat() (FileInfo, error)
- func (f *File) Sync() error
- func (f *File) Truncate(size int64) error
- func (f *File) Write(b []byte) (n int, err error)
- func (f *File) WriteAt(b []byte, off int64) (n int, err error)
- func (f *File) WriteString(s string) (n int, err error)

Does a *File implement the writer interface?

Yes.
We want to be able to read and write to a file

```
type Writer interface {
    Write(p []byte) (n int, err error)
}
```

godoc.org/os

type File

- func Create(name string) (*File, error)
- func NewFile(fd uintptr, name string) *File
- func Open(name string) (*File, error)
- func OpenFile(name string, flag int, perm FileMode) (*File, error)
- func Pipe() (r *File, w *File, err error)
- func (f *File) Chdir() error
- func (f *File) Chmod(mode FileMode) error
- func (f *File) Chown(uid, gid int) error
- func (f *File) Close() error
- func (f *File) Fd() uintptr
- func (f *File) Name() string
- func (f *File) Read(b []byte) (n int, err error)
- func (f *File) ReadAt(b []byte, off int64) (n int, err error)
- func (f *File) Readdir(n int) (fi []FileInfo, err error)
- func (f *File) Readdirnames(n int) (names []string, err error)
- func (f *File) Seek(offset int64, whence int) (ret int64, err error)
- func (f *File) Stat() (FileInfo, error)
- func (f *File) Sync() error
- func (f *File) Truncate(size int64) error
- func (f *File) Write(b []byte) (n int, err error)
- func (f *File) WriteAt(b []byte, off int64) (n int, err error)
- func (f *File) WriteString(s string) (n int, err error)

How we use the **reader** interface

func NewReader

```
func NewReader(s string) *Reader
```

NewReader returns a new Reader reading from s. It is similar to bytes.NewBufferString but more efficient and read-only.

func Open

```
func Open(name string) (*File, error)
```

Open opens the named file for reading. If successful, methods on the returned file can be used for reading; the associated file descriptor has mode O_RDONLY. If there is an error, it will be of type *PathError.

***Reader implemented the reader interface**

main.go

```
1  package main
2
3  import (
4      "log"
5      "io/ioutil"
6      "fmt"
7      "strings"
8  )
9
10 func main() {
11
12     rdr := strings.NewReader("some string")
13
14     bs, err := ioutil.ReadAll(rdr)
15     if err != nil {
16         log.Fatalln("my program broke")
17     }
18
19     fmt.Println(bs)
20     fmt.Println(string(bs))
21 }
```

godoc.org/io/ioutil#ReadAll

ks PM Hawk

func ReadAll ¶

```
func ReadAll(r io.Reader) ([]byte, error)
```

What implements the writer interface?

***File implemented the reader interface**

main.go

```
package main

import (
    "log"
    "os"
    "io/ioutil"
    "fmt"
)

10 func main() {
11     f, err := os.Open("hello.txt")
12     if err != nil {
13         log.Fatalln("my program broke")
14     }
15     defer f.Close()
16
17     bs, err := ioutil.ReadAll(f)
18     if err != nil {
19         log.Fatalln("my program broke")
20     }
21
22     fmt.Println(bs)
23     fmt.Println(string(bs))
24 }
```

How we use the **writer** interface

func NewReader

```
func NewReader(s string) *Reader
```

NewReader returns a new Reader reading from s. It is similar to bytes.NewBufferString but more efficient and read-only.

func Create

```
func Create(name string) (*File, error)
```

Create creates the named file with mode 0666 (before umask), truncating it if it already exists. If successful, methods on the returned File can be used for I/O; the associated file descriptor has mode O_RDWR. If there is an error, it will be of type *PathError.

***Reader implemented the reader interface**

main.go

```go
package main

import (
    "log"
    "io/ioutil"
    "fmt"
    "strings"
)

func main() {

    rdr := strings.NewReader("some string")

    bs, err := ioutil.ReadAll(rdr)
    if err != nil {
        log.Fatalln("my program broke")
    }

    fmt.Println(bs)
    fmt.Println(string(bs))
}
```

godoc.org/io/ioutil#ReadAll

ks PM Hawk

func ReadAll ¶

```
func ReadAll(r io.Reader) ([]byte, error)
```

What implements the writer interface?

***File implements the writer interface**

main.go

```go
package main

import (
    "fmt"
    "log"
    "os"
)

func main() {
    str := "Here is a phrase."
    bs := []byte(str)
    fmt.Println(str)
    fmt.Println(bs)

    f, err := os.Create("hello.txt")
    if err != nil {
        log.Fatalln("my program broke")
    }
    defer f.Close()

    n, err := f.Write(bs)
    if err != nil {
        log.Fatalln("my program broke")
    }
    fmt.Println(n)
}
```

Are there any funcs in package ioutil
which a type could use if it is implementing the writer interface?

package ioutil

```
import "io/ioutil"
```

Package ioutil implements some I/O utility functions.

Index

Are there any funcs in package ioutil
which a type could use if it is implementing the writer interface?

No.

package ioutil

```
import "io/ioutil"
```

Package ioutil implements some I/O utility functions.

Index

Variables
func NopCloser(r io.Reader) io.ReadCloser
func ReadAll(r io.Reader) ([]byte, error)
func ReadDir(dirname string) ([]os.FileInfo, error)
func ReadFile(filename string) ([]byte, error)
func TempDir(dir, prefix string) (name string, err error)
func TempFile(dir, prefix string) (f *os.File, err error)
func WriteFile(filename string, data []byte, perm os.FileMode) error

Readers

- *os.File
  - **Write**
    - Write writes len(b) bytes to the File.
    - func (f *File) Write(b []byte) (n int, err error)
- io/ioutil
  - **ReadAll**
    - reads from r until an error or EOF and returns the data it read
    - func ReadAll(**r io.Reader**) ([]byte, error)
  - Readdir
    - reads the contents of the directory associated with file and returns a slice of up to n FileInfo values
    - func (f *File) Readdir(n int) (fi []FileInfo, err error)
  - Readdirnames
    - reads and returns a slice of names from the directory f.
    - func (f *File) Readdirnames(n int) (names []string, err error)

```go
func main() {
    src, err := os.Open("src.txt")
    if err != nil {
        panic(err)
    }
    defer src.Close()

    dst, err := os.Create("dst.txt")
    if err != nil {
        panic(err)
    }
    defer dst.Close()

    bs := make([]byte, 5)
    src.Read(bs)
    dst.Write(bs)
}
```

```go
func main() {
    f, err := os.Open(os.Args[1])
    if err != nil {
        log.Fatalln("my program broke"
    }
    defer f.Close()

    bs, err := ioutil.ReadAll(f)
    if err != nil {
        log.Fatalln("my program broke
    }

    str := string(bs)
    fmt.Println(str)
}
```

primitives

package io

```
import "io"
```

Package io provides basic interfaces to I/O primitives. Its primary job is to wrap existing implementations of such primitives, such as those in package os into shared public interfaces that abstract the functionality, plus some other related primitives.

Because these interfaces and primitives wrap lower-level operations with various implementations, unless otherwise informed clients should not assume they are safe for parallel execution.

```
type Reader interface {
    Read(p []byte) (n int, err error)
}
```

package io

```
type Writer interface {
    Write(p []byte) (n int, err error)
}
```

What are “**I/O primitives**”?

package io

```
import "io"
```

Package io provides basic interfaces to I/O primitives. Its primary job is to wrap existing implementations of such primitives, such as those in package os into shared public interfaces that abstract the functionality, plus some other related primitives.

Because these interfaces and primitives wrap lower-level operations with various implementations, unless otherwise informed clients should not assume they are safe for parallel execution.

```
type Reader interface {
    Read(p []byte) (n int, err error)
}
```

package io

```
type Writer interface {
    Write(p []byte) (n int, err error)
}
```

What are “**I/O primitives**”?

- read
- write

package io

```
import "io"
```

Package io provides basic interfaces to I/O primitives. Its primary job is to wrap existing implementations of such primitives, such as those in package os, into shared public interfaces that abstract the functionality, plus some other related primitives.

Because these interfaces and primitives wrap lower-level operations with various implementations, unless otherwise informed clients should not assume they are safe for parallel execution.

```
type Reader interface {
    Read(p []byte) (n int, err error)
}
```

package io

```
type Writer interface {
    Write(p []byte) (n int, err error)
}
```

What are "**I/O primitives**"?

- read
- write

godoc.org/os

type File

- func Create(name string) (*File, error)
- func NewFile(fd uintptr, name string) *File
- func Open(name string) (*File, error)
- func OpenFile(name string, flag int, perm FileMode) (*File, error)
- func Pipe() (r *File, w *File, err error)
- func (f *File) Chdir() error
- func (f *File) Chmod(mode FileMode) error
- func (f *File) Chown(uid, gid int) error
- func (f *File) Close() error
- func (f *File) Fd() uintptr
- func (f *File) Name() string
- func (f *File) Read(b []byte) (n int, err error)
- func (f *File) ReadAt(b []byte, off int64) (n int, err error)
- func (f *File) Readdir(n int) (fi []FileInfo, err error)
- func (f *File) Readdirnames(n int) (names []string, err error)
- func (f *File) Seek(offset int64, whence int) (ret int64, err error)
- func (f *File) Stat() (FileInfo, error)
- func (f *File) Sync() error
- func (f *File) Truncate(size int64) error
- func (f *File) Write(b []byte) (n int, err error)
- func (f *File) WriteAt(b []byte, off int64) (n int, err error)
- func (f *File) WriteString(s string) (n int, err error)

type FileInfo

programmers.stackexchange.com/questions/139747/what-is-meant-by-a-primitive-data-type

It kind of depends on the language.

For example, in languages like C and C++, you have a number of built-in scalar types - `int`, `float`, `double`, `char`, etc. These are "primitive" in the sense that they cannot be decomposed into simpler components. From these basic types you can define new types - pointer types, array types, struct types, union types, etc.

package io

```
import "io"
```

Package io provides basic interfaces to I/O primitives. Its primary job is to wrap existing implementations of such primitives, such as those in package os into shared public interfaces that abstract the functionality, plus some other related primitives.

Because these interfaces and primitives wrap lower-level operations with various implementations, unless otherwise informed clients should not assume they are safe for parallel execution.

```
type Reader interface {
    Read(p []byte) (n int, err error)
}
```

package io

```
type Writer interface {
    Write(p []byte) (n int, err error)
}
```

What are "**I/O primitives**"?

- read
- write

godoc.org/os

type File

- func Create(name string) (*File, error)
- func NewFile(fd uintptr, name string) *File
- func Open(name string) (*File, error)
- func OpenFile(name string, flag int, perm FileMode) (*File, error)
- func Pipe() (r *File, w *File, err error)
- func (f *File) Chdir() error
- func (f *File) Chmod(mode FileMode) error
- func (f *File) Chown(uid, gid int) error
- func (f *File) Close() error
- func (f *File) Fd() uintptr
- func (f *File) Name() string
- func (f *File) Read(b []byte) (n int, err error)
- func (f *File) ReadAt(b []byte, off int64) (n int, err error)
- func (f *File) Readdir(n int) (fi []FileInfo, err error)
- func (f *File) Readdirnames(n int) (names []string, err error)
- func (f *File) Seek(offset int64, whence int) (ret int64, err error)
- func (f *File) Stat() (FileInfo, error)
- func (f *File) Sync() error
- func (f *File) Truncate(size int64) error
- func (f *File) Write(b []byte) (n int, err error)
- func (f *File) WriteAt(b []byte, off int64) (n int, err error)
- func (f *File) WriteString(s string) (n int, err error)

In computer science, **primitive data type** is either of the following:[citation needed]

- a *basic type* is a data type [1] provided by a programming language as a basic building block. Most languages allow more complicated *composite types* to be recursively constructed starting from basic types.
- a *built-in type* is a data type for which the programming language provides built-in support.

Primitive types are the most basic **data types** available within the Java language; these include boolean , byte , char , short , int , long , float and double . These **types** serve as the building blocks of **data** manipulation in Java.

Primitive Types - Wikibooks, open books for an open world
https://en.wikibooks.org/wiki/Java.../**Primitive_Types** Wikibooks

programmers.stackexchange.com/questions/139747/what-is-meant-by-a-primitive-data-type

It kind of depends on the language.

For example, in languages like C and C++, you have a number of built-in scalar types - `int` , `float` , `double` , `char` , etc. These are "primitive" in the sense that they cannot be decomposed into simpler components. From these basic types you can define new types - pointer types, array types, struct types, union types, etc.

unix file permissions

unix file system permissions

```
permission to:  owner      group      other
                /---\      /---\      /---\
octal:            6          6          6
binary:         1 1 0      1 1 0      1 1 0
what to permit: r w x      r w x      r w x

binary         - 1: enabled, 0: disabled

what to permit - r: read, w: write, x: execute

permission to  - owner: the user that create the file/folder
                 group: the users from group that owner is member
                 other: all other users
```

In Unix-like operating systems, **chmod** is the command and system call which may change the access permissions to file system objects (files and directories). It may also alter special mode flags. The request is filtered by the umask. The name is an abbreviation of change mode.

chmod - Wikipedia, the free encyclopedia
https://en.wikipedia.org/wiki/**Chmod** Wikipedia