atag - tag extender for Acme

(version 0.2.1)

Alexander Sychev (santucco@gmail.com)

1. Introduction. This is an implementation of `atag` command for `Acme`. It adds specified commands to a tag of every `Acme`'s window or only in windows, matched by a regular expression.

2. Implementation.

```
// This file is part of atag
//
// Copyright (c) 2020 Alexander Sychev. All rights reserved.
//
// Redistribution and use in source and binary forms, with or without
// modification, are permitted provided that the following conditions are
// met:
//
// * Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.
// * Redistributions in binary form must reproduce the above
// copyright notice, this list of conditions and the following disclaimer
// in the documentation and/or other materials provided with the
// distribution.
// * The name of author may not be used to endorse or promote products derived from
// this software without specific prior written permission.
//
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
// "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
// LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
// A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
// OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
// SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
// LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
// DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
// THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
// (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
// OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
package main
import(
  ⟨Imports 3⟩
)
var(
  ⟨Global variables 6⟩
)
```

3.

⟨Imports 3⟩ ≡

```
"fmt"
"os"
```

See also sections 5 and 8.

This code is used in section 2.

4. At first, if no commands are specified, let's print the usage info and exit. Then an enumeration of opened windows is processed in a separated goroutine. Then pooling of `Acme`'s log is started. Start of the enumeration is syncronized with the start of pulling `Acme`'s log.

```
func main(){
  if len(os.Args) ≡ 1 {
    fmt.Fprintf(os.Stderr, "Tag␣extender\nExtends␣tags␣of␣Acme␣with␣specified␣commands\n")
    fmt.Fprintf(os.Stderr, "Usage:␣%s␣[<regular␣expression>:]<commands>␣...\nwhere:\n",
        os.Args[0])
    fmt.Fprintf(os.Stderr,
        "\t<regular␣expression>␣-␣a␣regular␣expression␣applied␣to␣window's␣name\n")
    fmt.Fprintf(os.Stderr,
        "\t<commands>␣-␣a␣list␣of␣commands␣is␣added␣in␣every␣Acme's␣window\n")
    fmt.Fprintf(os.Stderr,
        "\t\t\tor␣in␣windows␣matched␣by␣a␣specified␣<regular␣expression>\n")
    return
  }
  ⟨ Parsing of a command line 7 ⟩
  sync := make(chan bool)
  ⟨ Enumerate the opened windows 10 ⟩
  ⟨ Start polling of Acme's log 9 ⟩
}
```

5.

```
⟨ Imports 3 ⟩ +≡
  "strings"
  "regexp"
```

6.

```
⟨ Global variables 6 ⟩ ≡
  common []string
  rgx map[*regexp.Regexp][]string = make(map[*regexp.Regexp][]string)
```

This code is used in section 2.

7.

```
⟨ Parsing of a command line 7 ⟩ ≡
  for _, v := range os.Args[1:] {
    v = strings.Trim(v, "\"'")
    f := strings.Split(v, ":")
    if len(f) ≡ 1 {
      common = append(common, v)
    } else if r, err := regexp.Compile(f[0]); err ≠ nil {
      fmt.Fprintf(os.Stderr, "cannot␣compile␣regexp␣%q:␣%s\n", f[0], err)
    } else {
      rgx[r] = strings.Fields(f[1])
    }
  }
```

This code is used in section 4.

8.

```
⟨ Imports 3 ⟩ +≡
  "github.com/santucco/goacme"
```

9.

```
⟨ Start polling of Acme's log 9 ⟩ ≡
  log, err := goacme.OpenLog()
  if err ≠ nil {
    fmt.Fprint(os.Stderr, err)
    return
  }
  defer log.Close()
  close(sync)
  for ev, err := log.Read(); err ≡ nil; ev, err = log.Read() {
    if ev.Type ≡ goacme.NewWin {
      id := ev.Id
      name := ev.Name
      ⟨ Write specified commands to a tag of the new window with id after pipe simbol 11 ⟩
    }
  }
```

This code is used in section 4.

10.

```
⟨ Enumerate the opened windows 10 ⟩ ≡
  go func(){
    ← sync
    ids, err := goacme.WindowsInfo()
    if err ≠ nil {
      fmt.Fprintf(os.Stderr, "cannot␣get␣a␣list␣of␣the␣opened␣windows␣of␣Acme:␣%v\n", err)
      return
    }
    for _, v := range ids {
      id := v.Id
      name := ""
      if len(v.Tag)⟩0 {
        name = v.Tag[0]
      }
      ⟨ Write specified commands to a tag of the new window with id after pipe simbol 11 ⟩
    }
  }()
```

This code is used in section 4.

11.

```
⟨ Write specified commands to a tag of the new window with id after pipe simbol 11 ⟩ ≡
  var tag []string
  for r, v := range rgx {
    if r.Match([]byte(name)) {
      tag = append(tag, v...)
    }
  }
  tag = append(tag, common...)
  if err := writeTag(id, tag); err ≠ nil {
    fmt.Fprint(os.Stderr, err)
  }
```

This code is used in sections 9 and 10.

12. Let's describe a writing of tag like a function

```
  func writeTag(id int, list []string) error{
    ⟨ Check if list is empty 13 ⟩
    ⟨ Open a window w by id 14 ⟩
    ⟨ Read the tag into s 15 ⟩
    ⟨ Remove the tag content before the pipe symbol 16 ⟩
    ⟨ Compose a new tag 17 ⟩
    ⟨ Clear the tag and write the new tag 18 ⟩
    return nil
  }
```

13.

```
⟨ Check if list is empty 13 ⟩ ≡
  if len(list) ≡ 0 {
    return nil
  }
```

This code is used in section 12.

14.

```
⟨ Open a window w by id 14 ⟩ ≡
  w, err := goacme.Open(id)
  if err ≠ nil {
    return fmt.Errorf("cannot␣open␣a␣window␣with␣id␣%d:␣%s\n", id, err)
  }
  defer w.Close()
```

This code is used in section 12.

15.

⟨ Read the tag into s 15 ⟩ ≡

```
f, err := w.File("tag")
if err ≠ nil {
    return fmt.Errorf("cannot get tag file of the window with id %d: %s\n", id, err)
}
var b [200]byte
n, err := f.Read(b[:])
if err ≠ nil {
    return fmt.Errorf("cannot read the tag of the window with id %d: %s\n", id, err)
}
s := string(b[:n])
```

This code is used in section 12.

16.

⟨ Remove the tag content before the pipe symbol 16 ⟩ ≡

```
if n = strings.Index(s, "|"); n ≡ −1 {
    n = 0
} else {
    n++
}
s = s[n:]
```

This code is used in section 12.

17. We remove duplicates from added command

⟨ Compose a new tag 17 ⟩ ≡

```
{
    f := strings.Fields(s)
    var l []string
    loop:
    for _, v := range list {
        for _, v2 := range f {
            if v ≡ v2 {
                continue loop
            }
        }
        l = append(l, v)
    }
    l = append(l, f...)
    s = " " + strings.Join(l, " ")
}
```

This code is used in section 12.

18.

⟨ Clear the tag and write the new tag 18 ⟩ ≡

```
if err := w.WriteCtl("cleartag"); err ≠ nil {
    return fmt.Errorf("cannot clear the tag of the window with id %d: %s\n", id, err)
}
if _, err := f.Write([]byte(s)); err ≠ nil {
    return fmt.Errorf("cannot write the tag of the window with id %d: %s\n", id, err)
}
```

This code is used in section 12.

Args: 4, 7. *Close*: 9, 14.

⟨ Check if *list* is empty 13 ⟩ Used in section 12.
⟨ Clear the tag and write the new tag 18 ⟩ Used in section 12.
⟨ Compose a new tag 17 ⟩ Used in section 12.
⟨ Enumerate the opened windows 10 ⟩ Used in section 4.
⟨ Global variables 6 ⟩ Used in section 2.
⟨ Imports 3, 5, 8 ⟩ Used in section 2.
⟨ Open a window *w* by *id* 14 ⟩ Used in section 12.
⟨ Parsing of a command line 7 ⟩ Used in section 4.
⟨ Read the tag into *s* 15 ⟩ Used in section 12.
⟨ Remove the tag content before the pipe symbol 16 ⟩ Used in section 12.
⟨ Start polling of `Acme`'s log 9 ⟩ Used in section 4.
⟨ Write specified commands to a tag of the new window with *id* after pipe simbol 11 ⟩ Used in sections 9 and 10.

atag - tag extender for Acme

(version 0.2.1)