

SimpleHTTPServer (Python)

(Superare Samba tramite modulo Python con o senza Upload)

Augusto Scatolini (webmaster@comunecampagnano.it) (a.scatolini@linux4campagnano.net)
Miniguia n. 131
Ver. 1.0 Giugno 2011



Avete necessità di condividere dei dati (dalla vostra Linux Box) sulla LAN con altri computer e non avete il tempo e la voglia di mettere mano a configurazioni complesse?

Quello di cui avete bisogno è un piccolo server, SimpleHTTPServer, da avviare dalla riga di comando.

SimpleHTTPServer è disponibile in tutti sistemi che hanno installata la versione Python 2.5 o superiore.

Per esempio, le distribuzioni GNU/Linux più note dispongono della versione Python aggiornata e con essa questo modulo che funge da HTTP server.

<http://www.pianeta.com/map/index.php/38342/python-condividere-dati-nella-lan-con-simplehttpserver-in-semplicit>

Nella directory `/usr/lib/pythonx.y/` di un sistema Unix-like come GNU/Linux Ubuntu c'è un modulo (uno script di due paginette di codice) che si chiama SimpleHTTPServer.py, quello che segue è il codice:

```

"""Simple HTTP Server.
This module builds on BaseHTTPServer by implementing the standard GET
and HEAD requests in a fairly straightforward manner.
"""
version = "0.6"
__all__ = ["SimpleHTTPRequestHandler"]
import os
import posixpath
import BaseHTTPServer
import urllib
import cgi
import shutil
import mimetypes
try:
    from cStringIO import StringIO
except ImportError:
    from StringIO import StringIO
class SimpleHTTPRequestHandler(BaseHTTPServer.BaseHTTPRequestHandler):
    """Simple HTTP request handler with GET and HEAD commands.
    This serves files from the current directory and any of its
    subdirectories. The MIME type for files is determined by
    calling the .guess_type() method.
    The GET and HEAD requests are identical except that the HEAD
    request omits the actual contents of the file.
    """
    server_version = "SimpleHTTP/" + version
    def do_GET(self):
        """Serve a GET request."""
        f = self.send_head()
        if f:
            self.copyfile(f, self.wfile)
            f.close()
    def do_HEAD(self):
        """Serve a HEAD request."""
        f = self.send_head()
        if f:
            f.close()
    def send_head(self):
        """Common code for GET and HEAD commands.
        This sends the response code and MIME headers.
        Return value is either a file object (which has to be copied
        to the outputfile by the caller unless the command was HEAD),
        and must be closed by the caller under all circumstances), or
        None, in which case the caller has nothing further to do.
        """
        path = self.translate_path(self.path)
        f = None
        if os.path.isdir(path):
            if not self.path.endswith('/'):
                # redirect browser - doing basically what apache does
                self.send_response(301)
                self.send_header("Location", self.path + "/")
                self.end_headers()
                return None
            for index in "index.html", "index.htm":
                index = os.path.join(path, index)
                if os.path.exists(index):
                    path = index
                    break
            else:
                return self.list_directory(path)
        ctype = self.guess_type(path)
        try:
            # Always read in binary mode. Opening files in text mode may cause
            # newline translations, making the actual size of the content
            # transmitted *less* than the content-length!
            f = open(path, 'rb')
        except IOError:
            self.send_error(404, "File not found")
            return None
        self.send_response(200)
        self.send_header("Content-type", ctype)
        fs = os.fstat(f.fileno())
        self.send_header("Content-Length", str(fs[6]))
        self.send_header("Last-Modified", self.date_time_string(fs.st_mtime))
        self.end_headers()
        return f
    def list_directory(self, path):
        """Helper to produce a directory listing (absent index.html).
        Return value is either a file object, or None (indicating an
        error). In either case, the headers are sent, making the
        interface the same as for send_head().
        """
        try:
            list = os.listdir(path)
        except os.error:
            self.send_error(404, "No permission to list directory")
            return None
        list.sort(key=lambda a: a.lower())
        f = StringIO()

```

```

displaypath = cgi.escape(urllib.unquote(self.path))
f.write("<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">")
f.write("<html>\n<title>Directory listing for %s</title>\n" % displaypath)
f.write("<body>\n<h2>Directory listing for %s</h2>\n" % displaypath)
f.write("<hr>\n<ul>\n")
for name in list:
    fullname = os.path.join(path, name)
    displayname = linkname = name
    # Append / for directories or @ for symbolic links
    if os.path.isdir(fullname):
        displayname = name + "/"
        linkname = name + "/"
    if os.path.islink(fullname):
        displayname = name + "@"
    # Note: a link to a directory displays with @ and links with /
    f.write("<li><a href=\"%s\">%s</a>\n"
           % (urllib.quote(linkname), cgi.escape(displayname)))
f.write("</ul>\n<hr>\n</body>\n</html>\n")
length = f.tell()
f.seek(0)
self.send_response(200)
self.send_header("Content-type", "text/html")
self.send_header("Content-Length", str(length))
self.end_headers()
return f

def translate_path(self, path):
    """Translate a /-separated PATH to the local filename syntax.
    Components that mean special things to the local file system
    (e.g. drive or directory names) are ignored. (XXX They should
    probably be diagnosed.)
    """
    # abandon query parameters
    path = path.split('?', 1)[0]
    path = path.split('#', 1)[0]
    path = posixpath.normpath(urllib.unquote(path))
    words = path.split('/')
    words = filter(None, words)
    path = os.getcwd()
    for word in words:
        drive, word = os.path.splitdrive(word)
        head, word = os.path.split(word)
        if word in (os.curdir, os.pardir): continue
        path = os.path.join(path, word)
    return path

def copyfile(self, source, outfile):
    """Copy all data between two file objects.
    The SOURCE argument is a file object open for reading
    (or anything with a read() method) and the DESTINATION
    argument is a file object open for writing (or
    anything with a write() method).
    The only reason for overriding this would be to change
    the block size or perhaps to replace newlines by CRLF
    -- note however that this the default server uses this
    to copy binary data as well.
    """
    shutil.copyfileobj(source, outfile)

def guess_type(self, path):
    """Guess the type of a file.
    Argument is a PATH (a filename).
    Return value is a string of the form type/subtype,
    usable for a MIME Content-type header.
    The default implementation looks the file's extension
    up in the table self.extensions_map, using application/octet-stream
    as a default; however it would be permissible (if
    slow) to look inside the data to make a better guess.
    """
    base, ext = posixpath.splitext(path)
    if ext in self.extensions_map:
        return self.extensions_map[ext]
    ext = ext.lower()
    if ext in self.extensions_map:
        return self.extensions_map[ext]
    else:
        return self.extensions_map[""]

if not mimetypes.inited:
    mimetypes.init() # try to read system mime.types
extensions_map = mimetypes.types_map.copy()
extensions_map.update({
    '.': 'application/octet-stream', # Default
    '.py': 'text/plain',
    '.c': 'text/plain',
    '.h': 'text/plain',
})

def test(handlerclass = SimpleHTTPRequestHandler,
         serverclass = BaseHTTPServer.HTTPServer):
    BaseHTTPServer.test(handlerclass, serverclass)

if __name__ == '__main__':
    test()

```

Questo modulo che si avvia con il comando (da terminale) `python -m SimpleHTTPServer` ha la facoltà di avviare a sua volta un piccolo server HTTP che ascolta sulla porta 8000.

Un'altra particolarità è dovuta al fatto che l'interprete Python e quindi il comando `python` può essere lanciato da qualunque posizione (nel File System), da qualunque directory.

Ne consegue che la radice del server HTTP cambia a seconda della posizione dalla quale si avvia il modulo e quindi il server.

Per esempio se apro il terminale, la posizione di default è la mia directory home. Questo è facilmente verificabile tramite il comando `pwd`

Se avvio il server da questa posizione con il comando `python -m SimpleHTTPServer` la radice del server HTTP corrisponde alla mia directory home

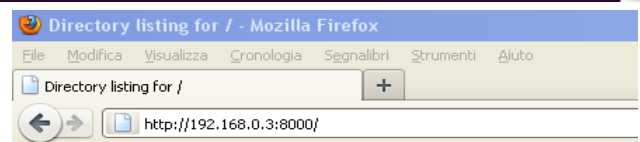
Se adesso da una macchina Win XP della stessa LAN (avente per esempio indirizzo IP = 192.168.0.99) apro un Browser all'indirizzo <http://192.168.0.3:8000> (questo è l'indirizzo della macchina Ubuntu sulla LAN, il terminale mi segnala che la macchina 192.168.0.99 ha chiesto la lista del contenuto della radice del server.

E in effetti questo è quello che vede la macchina Win XP 192.168.0.99

```
augusto@augusto-desktop: ~
File Modifica Visualizza Terminale Aiuto
augusto@augusto-desktop:~$ pwd
/home/augusto
augusto@augusto-desktop:~$
```

```
augusto@augusto-desktop: ~
File Modifica Visualizza Terminale Aiuto
augusto@augusto-desktop:~$ pwd
/home/augusto
augusto@augusto-desktop:~$ python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
```

```
augusto@augusto-desktop: ~
File Modifica Visualizza Terminale Aiuto
augusto@augusto-desktop:~$ pwd
/home/augusto
augusto@augusto-desktop:~$ python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
192.168.0.99 - - [12/Jun/2011 17:23:56] "GET / HTTP/1.1" 200 -
```



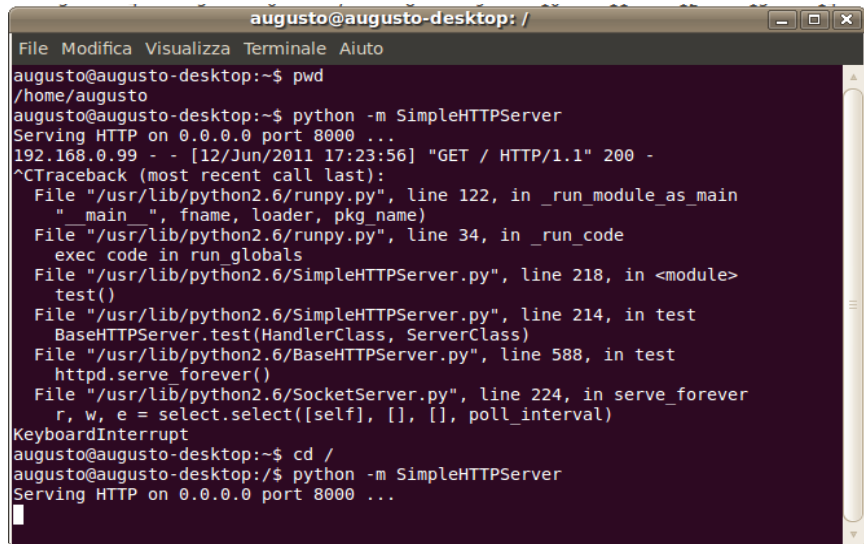
Directory listing for /

- [.AbiSuite/](#)
- [.adobe/](#)
- [.aMule/](#)
- [.archimedes/](#)
- [.autoscan-network/](#)
- [.avidemux/](#)
- [.bash_history](#)
- [.bash_logout](#)
- [.bashrc](#)
- [.BillardGL.conf.v7](#)
- [.bitrock/](#)
- [.bluefish/](#)
- [.bricscad/](#)
- [.bricsys/](#)
- [.bristol/](#)
- [.cache/](#)
- [.clicompanion2](#)
- [.compiz/](#)
- [.config/](#)
- [.DBDesigner4/](#)
- [.dbeaver/](#)
- [.dbus/](#)
- [.dike/](#)
- [.dmrc](#)
- [.dvdcss/](#)

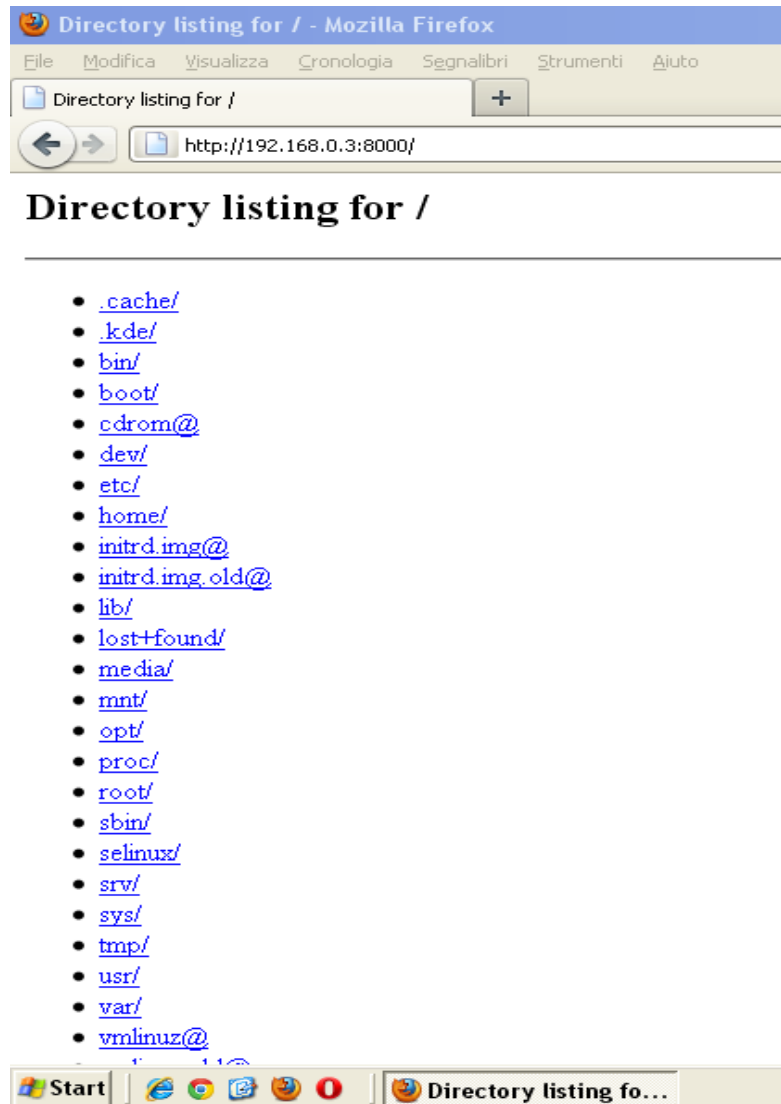


Ma, con questa logica, se tramite il terminale, (dopo aver chiuso il server HTTP con il comando **Ctrl+C**) mi sposto sulla radice del File System con il comando **cd /** e ri-avvio il server HTTP con il comando **python -m SimpleHTTPServer**

La macchina WinXP dovrebbe vedere l'elenco dei file e delle directory della radice della macchina GNU/Linux Ubuntu



```
augusto@augusto-desktop: /
File Modifica Visualizza Terminale Aiuto
augusto@augusto-desktop:~$ pwd
/home/augusto
augusto@augusto-desktop:~$ python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
192.168.0.99 - - [12/Jun/2011 17:23:56] "GET / HTTP/1.1" 200 -
^C
Traceback (most recent call last):
  File "/usr/lib/python2.6/runpy.py", line 122, in _run_module_as_main
    "_main_", fname, loader, pkg_name)
  File "/usr/lib/python2.6/runpy.py", line 34, in _run_code
    exec code in run_globals
  File "/usr/lib/python2.6/SimpleHTTPServer.py", line 218, in <module>
    test()
  File "/usr/lib/python2.6/SimpleHTTPServer.py", line 214, in test
    BaseHTTPServer.test(HandlerClass, ServerClass)
  File "/usr/lib/python2.6/BaseHTTPServer.py", line 588, in test
    httpd.serve_forever()
  File "/usr/lib/python2.6/SocketServer.py", line 224, in serve_forever
    r, w, e = select.select([self], [], [], poll_interval)
KeyboardInterrupt
augusto@augusto-desktop:~$ cd /
augusto@augusto-desktop:/$ python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
```



INFATTI →

Quindi lanciando questo comando avviamo un server HTTP la cui radice corrisponde alla directory dalla quale lanciamo il comando e il contenuto di quella directory sarà visibile e disponibile per il download da tutte le macchine della stessa rete LAN.

E non c'è un modo per poter fare l'upload oltre al download su quel server (directory)?

Grazie all'autore "bones7456" che dal sito <http://li2z.cn> sembrerebbe cinese (inutile aprirlo, il sito è DOWN) e alle sue 296 righe di codice scritto in Python basato sul codice SimpleHTTPServer.py ora abbiamo a disposizione il modulo `SimpleHTTPServerWithUpload.py`

quello che segue è il codice:

```
#!/usr/bin/env python

"""Simple HTTP Server With Upload.

This module builds on BaseHTTPServer by implementing the standard GET
and HEAD requests in a fairly straightforward manner.

"""

__version__ = "0.1"
__all__ = ["SimpleHTTPRequestHandler"]
__author__ = "bones7456"
__home_page__ = "http://li2z.cn/"

import os
import posixpath
import BaseHTTPServer
import urllib
import cgi
import shutil
import mimetypes
import re
try:
    from cStringIO import StringIO
except ImportError:
    from StringIO import StringIO

class SimpleHTTPRequestHandler(BaseHTTPServer.BaseHTTPRequestHandler):

    """Simple HTTP request handler with GET/HEAD/POST commands.

    This serves files from the current directory and any of its
    subdirectories. The MIME type for files is determined by
    calling the .guess_type() method. And can receive file uploaded
    by client.

    The GET/HEAD/POST requests are identical except that the HEAD
    request omits the actual contents of the file.

    """

    server_version = "SimpleHTTPWithUpload/" + __version__

    def do_GET(self):
        """Serve a GET request."""
        f = self.send_head()
        if f:
            self.copyfile(f, self.wfile)
            f.close()

    def do_HEAD(self):
        """Serve a HEAD request."""
        f = self.send_head()
        if f:
            f.close()

    def do_POST(self):
```

```

"""Serve a POST request."""
r, info = self.deal_post_data()
print r, info, "by: ", self.client_address
f = StringIO()
f.write("<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">")
f.write("<html>\n<title>Upload Result Page</title>\n")
f.write("<body>\n<h2>Upload Result Page</h2>\n")
f.write("<hr>\n")
if r:
    f.write("<strong>Success:</strong>")
else:
    f.write("<strong>Failed:</strong>")
f.write(info)
f.write("<br><a href='%s'>back</a>" % self.headers['referer'])
f.write("<hr><small>Powered By: bones7456, check new version at ")
f.write("<a href='http://li2z.cn/?s=SimpleHTTPServerWithUpload'>")
f.write("here</a>.</small></body>\n</html>\n")
length = f.tell()
f.seek(0)
self.send_response(200)
self.send_header("Content-type", "text/html")
self.send_header("Content-Length", str(length))
self.end_headers()
if f:
    self.copyfile(f, self.wfile)
    f.close()

def deal_post_data(self):
    boundary = self.headers.plisttext.split("=")[1]
    remainbytes = int(self.headers['content-length'])
    line = self.rfile.readline()
    remainbytes -= len(line)
    if not boundary in line:
        return (False, "Content NOT begin with boundary")
    line = self.rfile.readline()
    remainbytes -= len(line)
    fn = re.findall(r'Content-Disposition.*name="file"; filename="(.*)"', line)
    if not fn:
        return (False, "Can't find out file name...")
    path = self.translate_path(self.path)
    fn = os.path.join(path, fn[0])
    while os.path.exists(fn):
        fn += " "
    line = self.rfile.readline()
    remainbytes -= len(line)
    line = self.rfile.readline()
    remainbytes -= len(line)
    try:
        out = open(fn, 'wb')
    except IOError:
        return (False, "Can't create file to write, do you have permission to write?")

    preline = self.rfile.readline()
    remainbytes -= len(preline)
    while remainbytes > 0:
        line = self.rfile.readline()
        remainbytes -= len(line)
        if boundary in line:
            preline = preline[0:-1]
            if preline.endswith('\r'):
                preline = preline[0:-1]
            out.write(preline)
            out.close()
            return (True, "File '%s' upload success!" % fn)
        else:
            out.write(preline)
            preline = line
    return (False, "Unexpect Ends of data.")

```

```

def send_head(self):
    """Common code for GET and HEAD commands.

```

This sends the response code and MIME headers.

Return value is either a file object (which has to be copied to the outputfile by the caller unless the command was HEAD, and must be closed by the caller under all circumstances), or None, in which case the caller has nothing further to do.

```

"""
path = self.translate_path(self.path)
f = None
if os.path.isdir(path):
    if not self.path.endswith('/'):
        # redirect browser - doing basically what apache does
        self.send_response(301)
        self.send_header("Location", self.path + "/")
        self.end_headers()
        return None
    for index in "index.html", "index.htm":
        index = os.path.join(path, index)
        if os.path.exists(index):
            path = index
            break
    else:
        return self.list_directory(path)
ctype = self.guess_type(path)
try:
    # Always read in binary mode. Opening files in text mode may cause
    # newline translations, making the actual size of the content
    # transmitted *less* than the content-length!
    f = open(path, 'rb')
except IOError:
    self.send_error(404, "File not found")
    return None
self.send_response(200)
self.send_header("Content-type", ctype)
fs = os.fstat(f.fileno())
self.send_header("Content-Length", str(fs[6]))
self.send_header("Last-Modified", self.date_time_string(fs.st_mtime))
self.end_headers()
return f

```

```

def list_directory(self, path):
    """Helper to produce a directory listing (absent index.html).

```

Return value is either a file object, or None (indicating an error). In either case, the headers are sent, making the interface the same as for `send_head()`.

```

"""
try:
    list = os.listdir(path)
except os.error:
    self.send_error(404, "No permission to list directory")
    return None
list.sort(key=lambda a: a.lower())
f = StringIO()
displaypath = cgi.escape(urllib.unquote(self.path))
f.write("<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">")
f.write("<html>\n<title>Directory listing for %s</title>\n" % displaypath)
f.write("<body>\n<h2>Directory listing for %s</h2>\n" % displaypath)
f.write("<hr>\n")
f.write("<form ENCTYPE=\"multipart/form-data\" method=\"post\">")
f.write("<input name=\"file\" type=\"file\"/>")
f.write("<input type=\"submit\" value=\"upload\"/></form>\n")
f.write("<hr>\n<ul>\n")
for name in list:
    fullname = os.path.join(path, name)
    displayname = linkname = name
    # Append / for directories or @ for symbolic links
    if os.path.isdir(fullname):
        displayname = name + "/"
        linkname = name + "/"
    if os.path.islink(fullname):
        displayname = name + "@"
        # Note: a link to a directory displays with @ and links with /
    f.write("<li><a href=\"%s\">%s</a>\n"
           % (urllib.quote(linkname), cgi.escape(displayname)))
f.write("</ul>\n<hr>\n</body>\n</html>\n")
length = f.tell()
f.seek(0)
self.send_response(200)
self.send_header("Content-type", "text/html")
self.send_header("Content-Length", str(length))
self.end_headers()
return f

```

```
def translate_path(self, path):
    """Translate a /-separated PATH to the local filename syntax.
```

Components that mean special things to the local file system (e.g. drive or directory names) are ignored. (XXX They should probably be diagnosed.)

```
"""
# abandon query parameters
path = path.split('?')[0]
path = path.split('#')[0]
path = posixpath.normpath(urllib.unquote(path))
words = path.split('/')
words = filter(None, words)
path = os.getcwd()
for word in words:
    drive, word = os.path.splitdrive(word)
    head, word = os.path.split(word)
    if word in (os.curdir, os.pardir): continue
    path = os.path.join(path, word)
return path
```

```
def copyfile(self, source, outputfile):
    """Copy all data between two file objects.
```

The SOURCE argument is a file object open for reading (or anything with a read() method) and the DESTINATION argument is a file object open for writing (or anything with a write() method).

The only reason for overriding this would be to change the block size or perhaps to replace newlines by CRLF -- note however that this the default server uses this to copy binary data as well.

```
"""
shutil.copyfileobj(source, outputfile)
```

```
def guess_type(self, path):
    """Guess the type of a file.
```

Argument is a PATH (a filename).

Return value is a string of the form type/subtype, usable for a MIME Content-type header.

The default implementation looks the file's extension up in the table self.extensions_map, using application/octet-stream as a default; however it would be permissible (if slow) to look inside the data to make a better guess.

```
"""
base, ext = posixpath.splitext(path)
if ext in self.extensions_map:
    return self.extensions_map[ext]
ext = ext.lower()
if ext in self.extensions_map:
    return self.extensions_map[ext]
else:
    return self.extensions_map[""]
```

```
if not mimetypes.inited:
    mimetypes.init() # try to read system mime.types
extensions_map = mimetypes.types_map.copy()
extensions_map.update({
    '.': 'application/octet-stream', # Default
    '.py': 'text/plain',
    '.c': 'text/plain',
    '.h': 'text/plain',
})
```

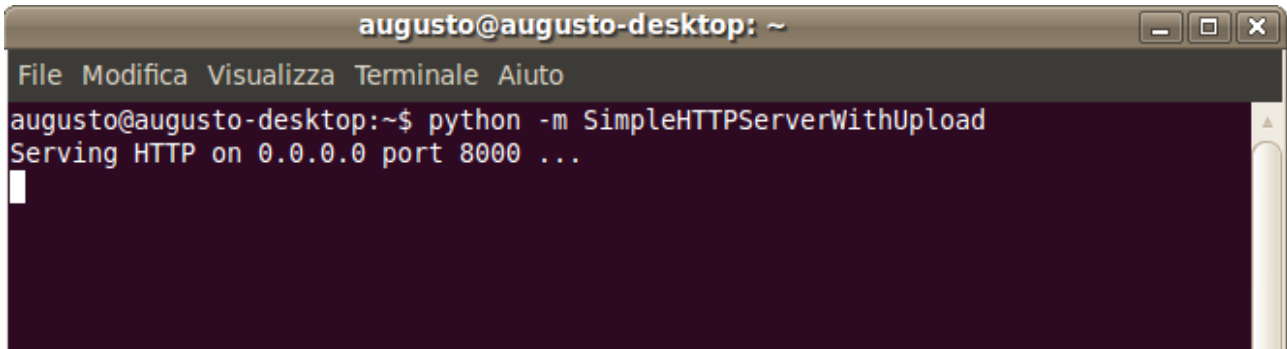
```
def test(HandlerClass = SimpleHTTPRequestHandler,
        ServerClass = BaseHTTPServer.HTTPServer):
    BaseHTTPServer.test(HandlerClass, ServerClass)
```

```
if __name__ == '__main__':
    test()
```

Quindi è sufficiente aprire un editor, incollare il codice (precedentemente copiato), salvare il file come `SimpleHTTPServerWithUpload.py` e collocarlo nella stessa directory dove è già presente il file `SimpleHTTPServer.py`

La modalità e la sintassi di utilizzo del modulo è identica a quella che si utilizza per il codice originale:

```
python -m SimpleHTTPServerWithUpload
```



```
augusto@augusto-desktop: ~
File Modifica Visualizza Terminale Aiuto
augusto@augusto-desktop:~$ python -m SimpleHTTPServerWithUpload
Serving HTTP on 0.0.0.0 port 8000 ...
```

Questa volta la macchina Win XP vedrà:



Niente male vero!

Ma come si può inserire un file come `SimpleHTTPServerWithUpload.py` in una directory nella quale l'utente normale non ha diritti di scrittura??

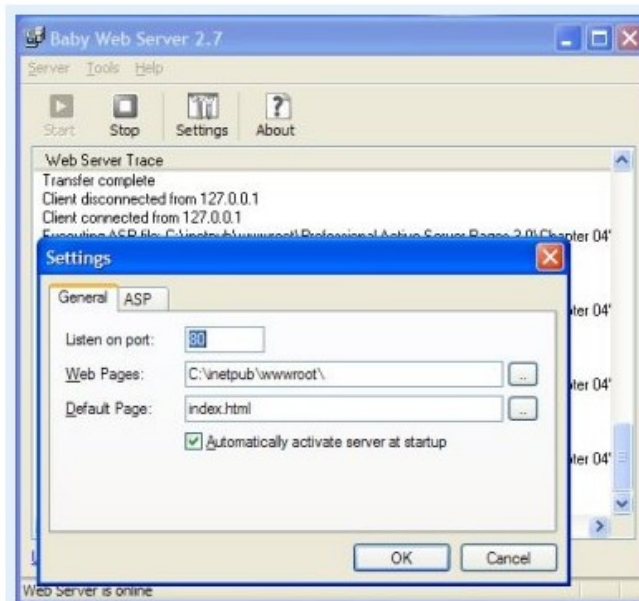
E' sufficiente lanciare nautilus (o un altro file manager) da terminale antepoendo il comando `sudo`.

E viceversa, ovvero un semplice server HTTP su Windows per accedervi dalla rete LAN?

Fortunatamente possiamo usare un piccolo web server portatile di soli 225 KB come **Baby Web Server**, gratuito, credo open Source, unico limite il numero massimo di connessioni simultanee che è pari a 5

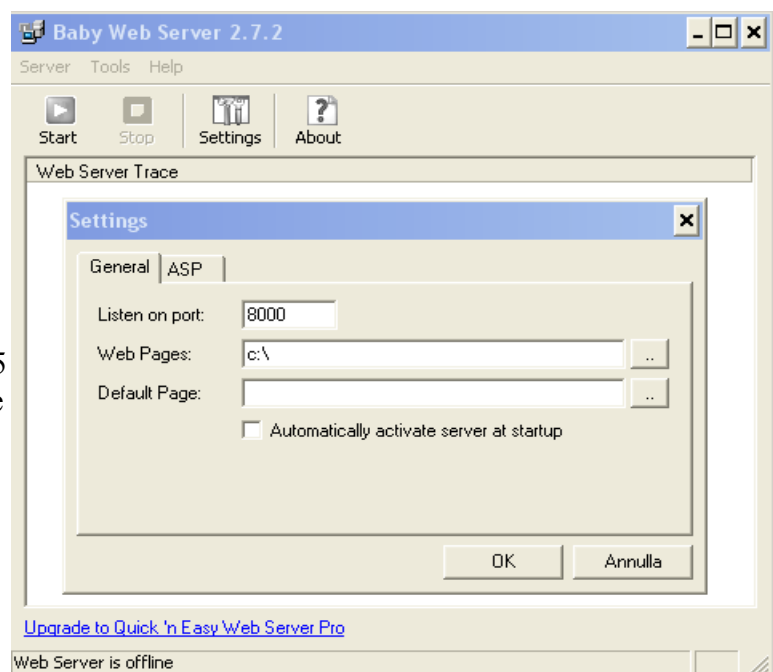


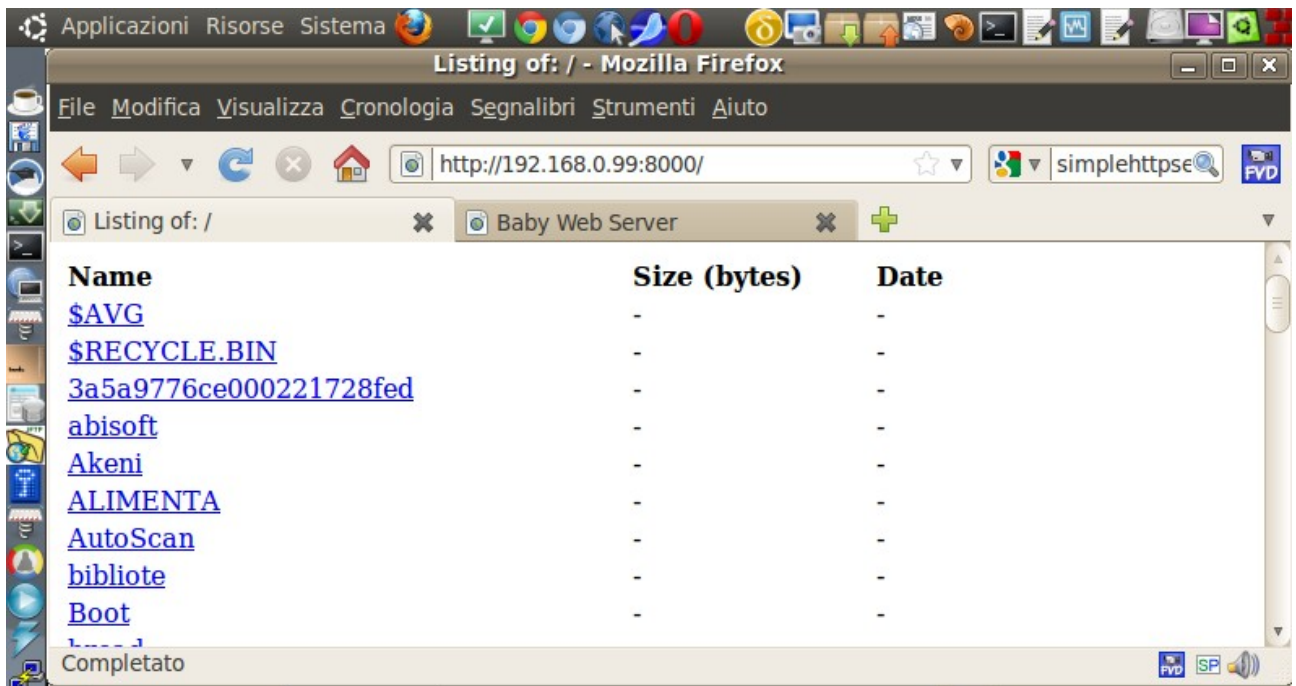
- ASP2VB Converter
- Baby FTP Server
- Baby POP3 Server
- Baby Web Server
- Cookie Viewer
- DBF Explorer
- FTP Wanderer 3.0
- Install Builder
- MDB 2 XML
- Multi Ping
- Pablo Commander
- POP3 Preview
- Popup Killer
- Quick 'n Easy FTP Server
- Quick 'n Easy Mail Server



Baby ASP Web Server Version 2.7.2

Una volta scaricato e lanciato l'eseguibile e settato il percorso della radice del sito e la porta (per comodità la settiamo su 8000) e avviato, tutte le macchine della stessa rete LAN (max 5 contemporaneamente) potranno vedere e scaricare il contenuto del sito.

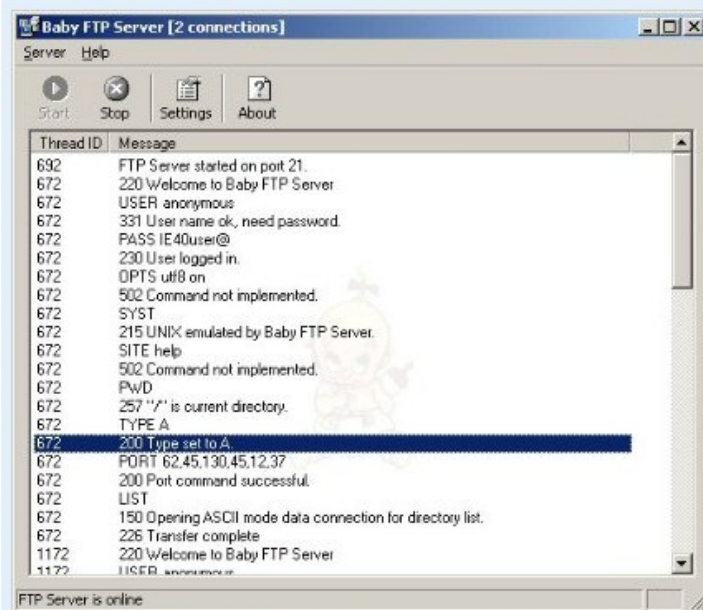




E non c'è un modo per poter fare l'upload oltre al download su quel server (directory)?
 In questo caso è opportuno scaricare un piccolo (114 KB) portatile e gratuito server FTP dallo stesso sito **Baby FTP Server**

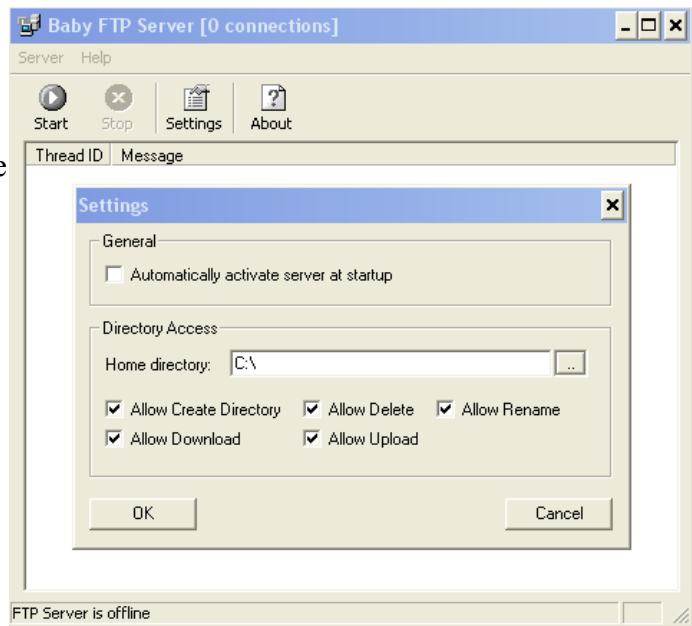


- ASP2VB Converter
- Baby FTP Server
- Baby POP3 Server
- Baby Web Server
- Cookie Viewer
- DBF Explorer
- FTP Wanderer 3.0
- Install Builder
- MDB 2 XML
- Multi Ping
- Pablo Commander
- POP3 Preview
- Popup Killer
- Quick'n Easy FTP Server
- Quick'n Easy Mail Server

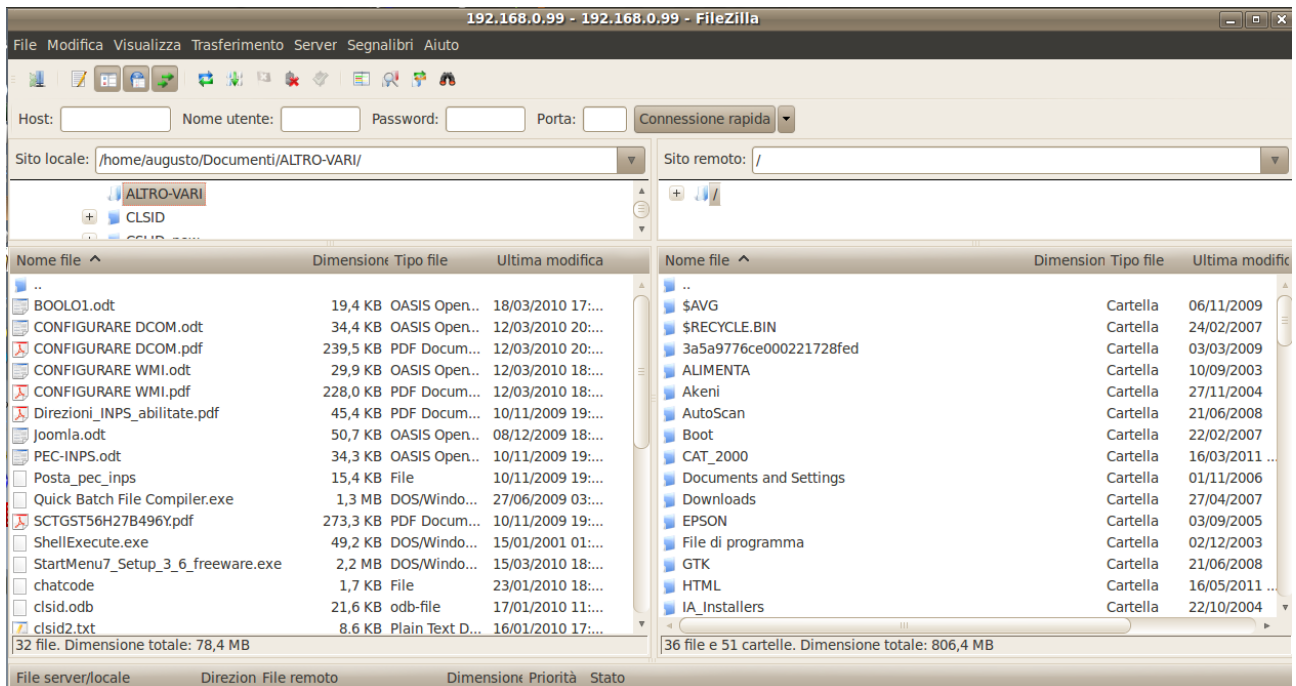


Baby FTP Server Version 1.24

Una volta scaricato e lanciato il file eseguibile e settato pochi parametri come la possibilità di fare l'upload, di creare directory, di cancellare file, di rinominare, ecc il server FTP sarà disponibile sulla classica porta 21



Da Ubuntu è comodo lanciare il client FTP FileZilla



Credo che sia sufficiente!

FINE

Questo documento è rilasciato con licenza Copyleft
(tutti i rovesci sono riservati)
altre miniguide su

<http://www.comunecampagnano.it/gnu/miniguide.htm>
oppure direttamente su <http://miniguide.tk>

sito consigliato: <http://www.linux4campagnano.net>