Installare moduli Python

Versione 2.3.4

Greg Ward

17 gennaio 2005

Python Software Foundation
Email: distutils-sig@python.org
Traduzione presso
http://www.zonapython.it
Email: zap@zonapython.it

Sommario

Questo documento descrive l'utility delle distribuzioni Python ("Distutils") dal punto di vista dell'utente finale, descrivendo come estendere le funzionalità di un'installazione Python standard per la compilazione e l'installazione di moduli ed estensioni Python fornite da terze parti.

Traduzione in italiano a cura di **Paolo Caldana** *verbal@teppisti.it – 20 gennaio 2005*.

Indice

Indice

1 Introduzione

Sebbene l'ampia libreria standard di Python copra molte necessità di programmazione, arriva sempre il momento in cui avrete bisogno di aggiungere nuove funzionalità alla vostra installazione Python sotto forma di moduli di terze parti. Questo può essere necessario per supportare il vostro personale stile di programmazione, o per supportare un'applicazione che volete usare e che sembra essere scritta in Python. In passato, c'è stato un supporto minimo per aggiungere moduli di terze parti ad un'installazione Python preesistente. Con l'introduzione dell'utility delle distribuzioni Python (in breve Distutils) in Python 2.0, questo è cambiato.

Questo documento punta principalmente alle persone che hanno bisogno di installare moduli Python di terze parti: utenti finali ed amministratori di sistema che hanno bisogno soltanto di fare girare alcune applicazioni Python e programmatori Python che vogliono aggiungere alcune raffinatezze alle loro toolbox. Non avete bisogno di conoscere Python per leggere questo documento; ci sarà qualche rapida occhiata sull'uso del modo interattivo di Python per esplorare la vostra installazione, tutto qui. Se state cercando informazioni su come distribuire i vostri moduli Python personali in modo che gli altri possano usarli, vedete il manuale *Distribuire moduli Python*.

1.1 Il caso: semplice installazione

Nel migliore dei casi, qualcuno avrà preparato una versione speciale del modulo in una distribuzione che volete installare, compilandolo specificatamente per la vostra piattaforma e sarà installabile come qualsiasi altro software disponibile. Per esempio, lo sviluppatore di moduli potrebbe realizzare un'installatore eseguibile per utenti Windows, un pacchetto RPM, per utenti di sistemi Linux basati sugli RPM (Red Hat, SuSe, Mandrake e molti altri), un pacchetto Debian per utenti di sistemi basati su Debian e così via.

In quel caso, vorrete scaricate l'installatore appropriato per la vostra piattaforma e farci la cosa più ovvia: eseguirlo se è un eseguibile, `rpm -install` se è un RPM, etc.. Non avrete bisogno di eseguire Python o uno script di setup, non avrete bisogno di compilare niente—non avrete neanche bisogno di leggere alcuna istruzione (sebbene sia sempre una cosa giusta da fare).

Sicuramente le cose non saranno sempre così semplici. Potreste essere interessati ad un modulo di una distribuzione che non ha un installer semplice da usare per la vostra piattaforma. In quel caso, dovrete iniziare con il sorgente della distribuzione rilasciato dall'autore/manutentore del modulo. Installare da un sorgente non è troppo difficile, purché i moduli siano pacchettizzati nel modo convenzionale. Questo documento tratta la compilazione e l'installazione dei moduli partendo da un sorgente standard di una distribuzione.

1.2 Il nuovo standard: Distutils

Se scaricate un modulo sorgente di una distribuzione, potrete sapere molto velocemente se è stato pacchettizzato e distribuito nel modo standard, per esempio usando Distutils. Per prima cosa osservate come sia il nome della distribuzione che il numero di versione verranno evidenziati nel nome dell'archivio scaricato, per esempio 'foo-1.0.tar.gz' o 'widget-0.9.7.zip'. Successivamente l'archivio si scompatterà all'interno di una directory chiamata in modo simile: 'foo-1.0' o 'widget-0.9.7'. In aggiunta, la distribuzione conterrà uno script di setup 'setup.py' ed un file chiamato 'README.txt' o solamente 'README', che dovrebbe spiegare come avviene la compilazione e l'installazione del modulo della distribuzione ed infine le modalità per eseguirlo

```
python setup.py install
```

Se avete fatto tutte queste cose, allora sarete pronti a conoscere come costruire ed installare il modulo che avete appena scaricato: eseguite il comando appena menzionato. A meno che non abbiate la necessità di installare qualcosa in un modo non convenzionale o non vogliate personalizzare il processo di compilazione, non avrete

realmente bisogno di questo manuale. Ovvero, il suddetto comando è tutto quello di cui avete bisogno e potreste anche non finire la lettura di questo manuale.

2 Installazione e compilazione standard

Come descritto nella sezione 1.2, la compilazione e l'installazione di un modulo per la vostra distribuzione, usando Distutils, è composta di solito da un semplice comando:

```
python setup.py install
```

Su UNIX, dovrete eseguire questo comando da una shell; su Windows, dovrete aprire una finestra di comando ("DOS box") e farlo lì; su Mac OS, le cose sono un po' più complicate (vedete più avanti).

2.1 Differenze tra le varie piattaforme

Dovreste sempre eseguire il comando setup dalla directory radice della vostra distribuzione, per esempio la sottodirectory di livello superiore dove il modulo sorgente della distribuzione vi è stato scompattato dentro. Per esempio, se avete appena scaricato un modulo sorgente per la vostra distribuzione 'foo-1.0.tar.gz' all'interno di un sistema UNIX, la cosa giusta da fare è:

```
gunzip -c foo-1.0.tar.gz | tar xf - # spacchetta nella directory
cd foo-1.0
python setup.py install
```

Su Windows, avrete probabilmente scaricato il file 'foo-1.0.zip'. Se avete scaricato il file archivio in 'C:\Temp', allora questo si spacchetterà in 'C:\Temp\foo-1.0'; potete usare un programma grafico per file compressi (come Winzip) o un tool da riga di comando (come **unzip** o **pkunzip**) per scompattare l'archivio. Quindi, aprite un terminale ("DOS box"), ed eseguite:

```
cd c:\Temp\foo-1.0
python setup.py install
```

Su Mac OS 9, fate doppio click sullo script 'setup.py'. Verrà aperta una finestra di dialogo da dove potrete selezionare il comando `install`, quindi selezionate il bottone `run` che lancerà l'installazione della vostra distribuzione. La finestra di dialogo viene costruita dinamicamente, quindi vengono elencati tutti i comandi e le opzioni per la vostra distribuzione.

2.2 Separare il lavoro da svolgere

Eseguendo `setup.py install` compilerete ed installerete tutti i moduli in una sola volta. Se preferite lavorare in modo incrementale—utile specialmente se volete personalizzare il processo di compilazione, o se le cose sono andate male—potete usare lo script setup per eseguire un passaggio alla volta. Questo aiuta particolarmente quando la compilazione e l'installazione viene fatta da utenti diversi—per esempio, potreste voler costruire un modulo per la vostra distribuzione e darlo ad un amministratore di sistema per installarlo (o farlo da soli, con i privilegi di super-utente). Potreste costruire tutto in una sola volta e quindi installare tutto successivamente, invocando lo script di installazione una seconda volta:

```
python setup.py build
python setup.py install
```

Se quindi vi comporterete come abbiamo appena visto, verrete avvisati che eseguendo il comando `install`, prima verrà eseguito il comando `build` che, in questo caso, avviserà velocemente che non c'è niente da fare finché il contenuto della directory 'build' non verrà aggiornato.

Potreste non avere bisogno di questa funzionalità, che comporta il dover interrompere spesso le proprie attività, se solitamente tutto quello che fate è installare moduli scaricati dalla rete, ma tutto questo è utile per molte altre sperimentazioni. Se volete distribuire i vostri moduli Python personali ed estensioni, potrete eseguire su di loro molti singoli comandi Distutils.

2.3 Come compilare il lavoro

Come descritto in precedenza, il comando `build` si incarica di mettere i file da installare dentro una *build directory*. Per definizione, questa è sotto la directory radice della distribuzione 'build'; se siete interessati eccessivamente alla velocità, o se volete prendere l'albero sorgente originario, potete cambiare la directory di compilazione con l'opzione **--build-base**.

Per esempio:

```
python setup.py build --build-base=/tmp/pybuild/foo-1.0
```

(O potete rendere questo permanente con una direttiva nel vostro sistema o nel file di configurazione personale Distutils; vedete la sezione 5.) Normalmente, questo non è necessario.

Lo schema predefinito per l'albero di compilazione è come il seguente :

```
--- build/ --- lib/
o
--- build/ --- lib.<plat>/
               temp.<plat>/
```

dove `<plat>` amplia una rapida descrizione della piattaforma OS/hardware corrente e della versione di Python. La prima forma, con una sola directory 'lib', viene usata per "distribuzioni pure dei moduli"—che sono distribuzioni di moduli che includono solo moduli Python puri. Se una distribuzione di moduli contiene qualsiasi estensione (moduli scritti in C/C++), allora viene usata la seconda forma, con due directory `<plat>`. In quel caso, la directory 'temp.*plat*' contiene i file temporanei generati dal processo compile/link che non sono stati veramente installati. In uno o nell'altro caso, la directory 'lib' (o 'lib.*plat*') contiene tutti i moduli Python (Python puro ed estensioni) che verranno installati.

In futuro, molte directory verranno aggiunte per manipolare gli script Python, la documentazione, i binari eseguibili e qualsiasi cosa sia necessaria per svolgere l'installazione dei moduli Python e delle applicazioni.

2.4 Come installare il lavoro

Dopo l'esecuzione del comando `build` (sia che lo eseguiate esplicitamente o che il comando `install` lo faccia per voi), il lavoro del comando `install` è relativamente semplice: tutto quello che deve fare è copiare tutto sotto 'build/lib' (o 'build/lib.*plat*') nella directory che avete scelto per l'installazione.

Se non scegliete una directory di installazione—per esempio, se eseguite soltanto `setup.py install`—allora il comando `install` installa nell'indirizzo convenzionale per i moduli di terze parti di Python. Questo indirizzo varia a seconda della piattaforma e da come compilerete autonomamente i moduli. Su UNIX e Mac OS, questo dipende anche dalla struttura del modulo, se ciò che stiamo installando è Python puro o contiene estensioni ("non-pure"):

Piattaforma	Indirizzo per l'installazione predefinita	Valore predefinito	Note
UNIX (puro)	*prefix*/lib/python2.0/site-packages	/usr/local/lib/python2.0/site-packages	(1)
UNIX (non puro)	*exec-prefix*/lib/python2.0/site-packages	/usr/local/lib/python2.0/site-packages	(1)
Windows	*prefix*	C:\Python	(2)
Mac OS (puro)	*prefix*:Lib:site-packages	Python:Lib:site-packages	
Mac OS (non puro)	*prefix*:Lib:site-packages	Python:Lib:site-packages	

Note:

(1) La maggior parte delle distribuzioni Linux include Python come una parte standard del sistema, così *prefix* ed *exec-prefix* sono solitamente entrambi in '/usr' su Linux. Se compilerete da soli Python su Linux (o su un sistema UNIX-like), i predefiniti *prefix* e *exec-prefix* sono '/usr/local'.

(2) La directory d'installazione predefinita in Windows era 'C:\Program Files\Python' per Python 1.6a1, 1.5.2, e precedenti.

prefix ed *exec-prefix* rimangono per le directory in cui viene installato Python e rappresentano il luogo dove trova le sue librerie al momento dell'esecuzione. Sono sempre le stesse sotto Windows e Mac OS e spesso sotto UNIX. Potete trovare dove sono nella vostra installazione Python, sia *prefix* che *exec-prefix*, eseguendo Python in modalità interattiva e digitando pochi semplici comandi. Sotto UNIXdigitate semplicemente `python` al prompt della shell. Sotto Windows, scegliete Avvio > Programmi > Python 2.1 > Python (riga di comando). Sotto Mac OS 9, avvio 'PythonInterpreter'. Una volta che l'interprete è avviato, digitate il codice Python al prompt. Per esempio, sul mio sistema Linux, io digito le tre istruzioni mostrate sotto, e prendo l'output come mostrato, per estrarre i miei *prefix* e *exec-prefix*:

```
Python 2.3.4 (#1, May 29 2004, 17:05:23)
[GCC 3.3.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> sys.prefix
'/usr'
>>> sys.exec_prefix
'/usr'
```

Se non volete installare i moduli nella posizione predefinita, o se non avete i permessi per scriverci, allora avete bisogno di leggere dell'installazione alternativa nella sezione 3. Se volete personalizzare la vostra directory di installazione in modo più marcato, vedete la sezione 4 nell'installazione personalizzata.

3 Alternative per l'installazione

Spesso, questo è necessario, o desiderabile, per installare moduli in una posizione differente dalla posizione predefinita per i moduli di terze parti di Python. Per esempio, su un sistema UNIX potreste non avere i permessi per scrivere nella directory predefinita per i moduli di terze parti. O vorreste provare un modulo prima di includerlo in una parte predefinita della vostra distribuzione Python locale. Questo è soprattutto vero quando si aggiorna una distribuzione già esistente: vorrete essere sicuri che la vostra base esistente di script lavori con la nuova versione prima dell'aggiornamento attuale.

Il comando Distutils `install` è progettato per eseguire un'installazione dei moduli di una distribuzione in una posizione alternativa in modo semplice ed indolore. L'idea principalmente è quella di fornire una directory di base per l'installazione ed il comando `install` sceglie una serie di directory (chiamamdo uno *schema d'installazione*) sotto questa directory di base, nella quale installare i file. I dettagli differiscono a seconda delle piattaforme, così da leggere comunque le sezioni che voi utilizzate.

3.1 Installazione alternativa: UNIX (lo schema casalingo)

Sotto UNIX, ci sono due modi per effettuare un'installazione alternativa. Lo "schema prefissato" è simile a come le installazioni alternative lavorano sotto Windows e Mac OS, ma non è necessariamente il modo più efficace per

mantenere una libreria personale Python. Perciò documenteremo prima il modo più conveniente e utile per uno "schema casalingo".

L'idea dietro lo "schema casalingo" è costruire e tenere i moduli Python personali al sicuro, probabilmente sotto la vostra directory home. L'installazione di un nuovo modulo di una distribuzione è semplice, come:

```
python setup.py install --home=<dir>
```

dove potete fornire ogni directory a piacimento per l'opzione **--home**. I pigri possono digitare solo una tilde (~); il comando `install` lo espanderà alla vostra directory home:

```
python setup.py install --home=~
```

L'opzione **--home** definisce la directory base dell'installazione. I file verranno installati nelle seguenti directory sotto l'installazione di base come segue:

Tipo di file	**Directory d'installazione**	**Opzioni di sovrascrittura**
distribuzione di moduli puri	*home*/lib/python	**--install-purelib**
distribuzione di moduli non puri	*home*/lib/python	**--install-platlib**
scripts	*home*/bin	**--install-scripts**
data	*home*/share	**--install-data**

3.2 Installazione alternativa: UNIX (lo schema prefissato)

Lo "schema prefix" è utile quando volete usare un'installazione Python per eseguire il build/install (per esempio per avviare lo script di setup), ma volete installare i moduli in una directory di moduli di terze parti di una differente installazione Python (o qualcosa che assomigli ad una differente installazione Python). Se questo suona come un dettaglio molto insignificante è perché lo "schema casalingo" è già stato descritto. Pertanto, ci sono almeno due casi conosciuti dove lo schema prefissato sarebbe utile.

Per prima cosa considerate che molte distribuzioni Linux mettono Python in '/usr', invece che nel più tradizionale '/usr/local'. Questo è assolutamente appropriato, poiché in quei casi Python è parte "del sistema" piuttosto che un programma aggiuntivo locale. Pertanto, se state installando i moduli Python da sorgente, probabilmente vorrete che finiscano in '/usr/local/lib/python2.*X*', piuttosto che in '/usr/lib/python2.*X*'. Questo può essere fatto con

```
/usr/bin/python setup.py install --prefix=/usr/local
```

Un'altra possibilità è un filesystem di rete, dove il nome usato per scrivere nella directory remota è differente dal nome usato per leggerla: per esempio, l'interprete Python accedendo come '/usr/local/bin/python' potrebbe ricercare i moduli in '/usr/local/lib/python2.*X*', ma questi moduli dovrebbero anche essere installati, dichiarando, '/mnt/@*server*/export/lib/python2.*X*'. Questo potrebbe essere fatto con

```
/usr/local/bin/python setup.py install --prefix=/mnt/@server/export
```

In entrambi i casi, l'opzione **--prefix** definisce l'installazione base e l'opzione **--exec-prefix** definisce l'installazione base per i file della piattaforma specifica. Allo stato attuale questo significa solo distribuzioni di moduli non puri, ma potrebbe essere esteso alle librerie C, ai binari eseguibili, etc. etc.. Se **--exec-prefix** non viene fornito, viene predefinito **--prefix**. I file vengono installati così:

Tipo di file	**Directory d'installazione**	**Opzioni di sovrascrittura**
distribuzione di moduli puri	*prefix*/lib/python2.*X*/site-packages	**--install-purelib**
distribuzione di moduli non puri	*exec-prefix*/lib/python2.*X*/site-packages	**--install-platlib**
scripts	*prefix*/bin	**--install-scripts**
data	*prefix*/share	**--install-data**

Non viene richiesto che **--prefix** o **--exec-prefix** puntino ad un'installazione Python alternativa; se le directory menzionate non esistono ancora, verranno create al momento dell'installazione.

Incidentalmente, la reale ragione per cui lo schema prefix è importante è che l'installazione standard UNIX usa lo schema prefix, ma con **--prefix** ed **--exec-prefix** forniti da Python stesso come `sys.prefix` e `sys.exec_prefix`. Così, potreste pensare di non usare mai lo schema prefix, ma in realtà, ogni volta che eseguirete `python setup.py install` senza nessun'altra opzione, ne farete uso.

Notate che installando le estensioni in un'installazione alternativa di Python non ci saranno effetti su come queste estensioni sono state compilate: in particolare, i file d'intestazione di Python ('Python.h' e compagnia), installati con l'interprete usato per eseguire lo script di setup, verranno usati nella compilazione delle estensioni. È vostra responsabilità assicurarvi che l'interprete usato per eseguire l'installazione delle estensioni in questo modo sia compatibile con l'interprete usato per compilarle. Il modo migliore per assicurarvene è controllare che i due interpreti siano della stessa versione di Python (possibilmente di differenti compilazioni o magari copie della stessa compilazione). Naturalmente, se i vostri **--prefix** e **--exec-prefix** non puntano parimenti ad un'installazione alternativa di Python, questo è irrilevante.

3.3 Installazione alternativa: Windows

Poiché Windows non ha nessuna concezione della directory home degli utenti e poiché l'installazione standard di Python sotto Windows è più semplice che sotto UNIX, non ci sono differenze tra le opzioni **--prefix** ed **--home**. Usate soltanto l'opzione **--prefix** per specificare una directory base. Per esempio:

```
python setup.py install --prefix="\Temp\Python"
```

per installare i moduli nella directory '\Temp\Python' del disco corrente.

L'installazione viene definita dall'opzione **--prefix**; l'opzione **--exec-prefix** non è supportata sotto Windows. I file vengono installati così:

Tipo di file	**Directory d'installazione**	**Opzioni di sovrascrittura**
distribuzione di moduli puri	*prefix*	**--install-purelib**
distribuzione di moduli non puri	*prefix*	**--install-platlib**
scripts	*prefix*\Scripts	**--install-scripts**
data	*prefix*\Data	**--install-data**

3.4 Installazione alternativa: Mac OS 9

Come Windows, Mac OS non ha nozione delle home directory (o degli utenti), ha una semplice installazione standard di Python. Quindi è necessaria solamente un'opzione **--prefix**. Questa definisce la base dell'installazione ed i file vengono installati al di sotto di essa, nel seguente modo:

Tipo di file	**Directory d'installazione**	**Opzioni di sovrascrittura**
distribuzione di moduli puri	*prefix*:Lib:site-packages	**--install-purelib**
distribuzione di moduli non puri	*prefix*:Lib:site-packages	**--install-platlib**
scripts	*prefix*:Scripts	**--install-scripts**
data	*prefix*:Data	**--install-data**

Vedete la sezione 2.1 per informazioni su come fornire ulteriori argomenti da riga di comando allo script di setup con MacPython.

4 Installazione personalizzata

Qualche volta, lo schema di installazione alternativo descritto nella sezione 3 non fa quello che volete. Potreste volere modificare solo una o due directory mentre tenete tutto sotto la stessa directory di base, o vorreste ridefinire completamente lo schema di installazione. In ogni caso, otterrete una *schema d'installazione personalizzato*.

Avrete probabilmente visto la colonna delle "opzioni di sovrascrittura" nelle tabelle che descrivono lo schema di installazione alternativo sopraindicato. Queste opzioni definiscono il vostro schema d'installazione personalizzato. Queste opzioni di sovrascrittura possono essere relative, assolute o esplicitamente definite nei termini di una delle directory di base dell'installazione. (Ci sono due directory di base d'installazione e normalmente sono le stesse—differiscono nell'uso dello "schema prefissato" UNIX e quando fornite opzioni differenti per **--prefix** ed **--exec-prefix**.)

Per esempio, volete installare un modulo nella vostra directory home sotto UNIX—ma volete che gli script vadano in '~/scripts' piuttosto che in '~/bin'. Come potreste aspettarvi, potete sovrascrivere questa directory mediante l'opzione **--install-scripts**; in questo caso , ha più senso fornire un percorso relativo, che verrà interpretato come relativo alla directory di base dell'installazione (la vostra home directory, in questo caso):

```
python setup.py install --home=~ --install-scripts=scripts
```

Un altro esempio UNIX: supponiamo che la vostra installazione di Python sia stata compilata ed installata con il prefisso '/usr/local/python', così uno script di un'installazione standard verrà caricato in '/usr/local/python/bin'. Se invece lo volete in '/usr/local/bin', dovrete fornire questa directory assoluta per l'opzione **--install-scripts**:

```
python setup.py install --install-scripts=/usr/local/bin
```

Quanto descritto esegue un'installazione usando lo "schema prefissato", dove il valore per prefix è ovunque il vostro interprete Python sia installato con — '/usr/local/python' in questo caso.

Se avete Python su Windows, potreste volere conservare i moduli di terze parti nella sottodirectory *prefix*, invece che nella predefinita *prefix*. Questo è semplice quasi come la personalizzazione dello script di installazione nella directory—dovete solamente ricordare che ci sono due tipi di moduli che vi interessano, moduli puri e moduli non puri (per esempio moduli provenienti da una distribuzione non pura). Per esempio:

```
python setup.py install --install-purelib=Site --install-platlib=Site
```

Le directory d'installazione specificate sono relative al *prefix*. All'occorrenza, potete anche assicurarvi che queste directory siano nel percorso di ricerca dei moduli di Python, inserendo un file '.pth' in *prefix*. Vedete la sezione 4.1 per capire come modificare il percorso di ricerca di Python.

Se volete definire un intero schema di installazione, dovete soltanto fornire tutte le opzioni delle directory di installazione. Il metodo raccomandato per farlo è fornire percorsi relativi; per esempio, se volete mantenere tutti i moduli relativi a Python sotto 'python' nella vostra home e volete una directory separata per ogni piattaforma che usate, sempre nella vostra home directory, potreste definire il seguente schema di installazione:

```
python setup.py install --home=~ \
                        --install-purelib=python/lib \
                        --install-platlib=python/lib.$PLAT \
                        --install-scripts=python/scripts
                        --install-data=python/data
```

o, in modo equivalente,

```
python setup.py install --home=~/python \
                        --install-purelib=lib \
                        --install-platlib='lib.$PLAT' \
                        --install-scripts=scripts
                        --install-data=data
```

`$PLAT` non è (necessariamente) una variabile di sviluppo—verrà espansa da Distutils come se analizzasse le opzioni della vostra riga di comando, come è stato fatto quando sono stati analizzati i vostri file di configurazione.

Ovviamente, specificare l'intero schema di installazione ogni volta che installate un nuovo modulo di una distribuzione diventerebbe molto tedioso. In tal caso, potete mettere queste opzioni nel file di configurazione di Distutils (vedete la sezione 5):

```
[install]
install-base=$HOME
install-purelib=python/lib
install-platlib=python/lib.$PLAT
install-scripts=python/scripts
install-data=python/data
```

o, in modo equivalente,

```
[install]
install-base=$HOME/python
install-purelib=lib
install-platlib=lib.$PLAT
install-scripts=scripts
install-data=data
```

Notate che vi sono cose *non* equivalenti, nel caso in cui forniate una differente directory di installazione di base quando eseguite lo script di setup. Per esempio,

```
python setup.py --install-base=/tmp
```

nel primo caso installerà i moduli puri in */tmp/python/lib* ed in */tmp/lib* nel secondo. (Per il secondo caso, probabilmente vorrete vorrete fornire una base di installazione in '/tmp/python'.)

Avrete probabilmente notato l'uso di `$HOME` e `$PLAT` nell'esempio di configurazione del file in ingresso. Quelle sono le variabili di configurazione di Distutils, che assomigliano molto alle variabili di ambiente. Di fatto, potete usare le variabili di ambiente nei file di configurazione su piattaforme che supportano questa notazione, ma Distutils definisce anche qualche ulteriore variabile che potrebbe non essere presente nel vostro ambiente, come `$PLAT`. (All'occorrenza, su sistemi che non hanno variabili di ambiente, come Mac OS 9, le variabili di configurazione fornite da Distutils sono le uniche che potete usare.) Vedete la sezione 5 per i dettagli.

4.1 Modificare il percorso di ricerca di Python

Quando l'interprete Python esegue un'istruzione `import`, ricerca sia codice Python che moduli di estensione lungo il percorso di ricerca. Un valore predefinito per il percorso viene configurato nel binario Python quando l'interprete viene compilato. Potete determinare il percorso per importare il modulo `sys` e stampare il valore di `sys.path`.

```
$ python
Python 2.2 (#11, Oct  3 2002, 13:31:27)
[GCC 2.96 20000731 (Red Hat Linux 7.3 2.96-112)] on linux2
Type ‘‘help’’, ‘‘copyright’’, ‘‘credits’’ or ‘‘license’’ for more information.
>>> import sys
>>> sys.path
[’’, ’/usr/local/lib/python2.3’, ’/usr/local/lib/python2.3/plat-linux2’,
 ’/usr/local/lib/python2.3/lib-tk’, ’/usr/local/lib/python2.3/lib-dynload’,
 ’/usr/local/lib/python2.3/site-packages’]
>>>
```

La stringa null in `sys.path` rappresenta la directory di lavoro corrente.

La convenzione prevista per i pacchetti installati localmente è di metterli nella directory ‘.../site-packages/’, ma potreste volere installare i moduli Python all’interno di qualche directory arbitraria. Per esempio il vostro sito potrebbe avere una convenzione per prendere tutto il software correlato al server web sotto ‘/www’. I moduli Python aggiuntivi potrebbero quindi stare in ‘/www/python’ e in previsione di effettuarne una successiva importazione, questa directory dovrebbe essere aggiunta al `sys.path`. Esistono vari modi per aggiungere directory al percorso di ricerca di Python.

Un sistema conveniente è quello di aggiungere una directory al file di configurazione, che sia già presente nel percorso di ricerca di Python, di solito nella directory ‘.../site-packages/’. I file di configurazione del percorso hanno l’estensione ‘.pth’ ed ogni riga deve contenere un singolo percorso che verrà accodato a `sys.path`. (Perché il nuovo percorso sia accodato a `sys.path`, i moduli nelle directory aggiunte non dovranno sovrascrivere i moduli standard. Questo significa che non potete utilizzare questo meccanismo per installare nuove versioni di moduli standard.)

I percorsi possono essere assoluti o relativi, in questo caso sono relativi alla directory contenente il file ‘.pth’. Ogni directory aggiunta al percorso di ricerca verrà scansionata in cerca di file ‘.pth’. Vedete *la documentazione per il modulo `site`* per maggiori informazioni.

Un modo meno conveniente è quello di editare il file ‘site.py’ della libreria standard di Python e modificare il `sys.path`. Il ‘site.py’ viene automaticamente importato quando viene eseguito l’interprete Python, a meno che non sia presente l’opzione **-S** per sopprimere questo comportamento. In questo modo potete semplicemente editare il file ‘site.py’ ed aggiungere due righe.

```
import sys
sys.path.append(’/www/python/’)
```

Tuttavia, se reinstallate la stessa versione di Python (forse quando fate un upgrade da 2.2 alla 2.2.2 per esempio) il file ‘site.py’ verrà sovrascritto dalla versione corrente. Dovete ricordarvi quello che è stato modificato e salvarvi una copia prima di effettuare l’installazione.

Ci sono due variabili d’ambiente che possono modificare `sys.path`. Tramite PYTHONHOME è possibile impostare un valore alternativo per il valore prefix dell’installazione Python. Per esempio, se PYTHONHOME viene impostato a ‘`/www/python`’, il percorso di ricerca verrà impostato a `[’’, ’/www/python/lib/python2.2/’’, ’/www/python/lib/python2.3/plat-linux2’, ...]`.

La variabile PYTHONPATH può essere impostata ad una lista di percorsi che verranno aggiunti all’inizio di `sys.path`. Per esempio, se PYTHONPATH viene impostato a ‘`/www/python:/opt/py`’, il percorso di ricerca inizierà con `[’/www/python’, ’/opt/py’]`. (Notate che le directory devono esistere, nell’ordine con le quali vengono aggiunte a `sys.path`; il modulo `site` rimuove i percorsi che non esistono.)

Infine, `sys.path` è solamente una semplice lista Python, così ogni applicazione Python può modificarla aggiungendo o rimuovendo le sue voci.

5 Il file di configurazione di Distutils

Come descritto in precedenza, potete usare i file di configurazione di Distutils per registrare preferenze personali o indirizzi per ogni opzione di Distutils. Così, ogni opzione ed ogni comando possono essere conservati in uno, due

o tre file di configurazione (a seconda della vostra piattaforma) che potranno essere consultati prima che venga analizzata la riga di comando. Questo significa che quei file di configurazione annulleranno i valori predefiniti e la riga di comando sovrascriverà i file di configurazione. Inoltre, se si applicano file di configurazione multipli, i valori presenti nei file "iniziali" verranno annullati da quelli "successivi".

5.1 Indirizzi e nomi dei file di configurazione

I nomi e la posizione dei file di configurazione varieranno leggermente nelle diverse piattaforme. Su UNIX, i tre file di configurazione (nell'ordine in cui vengono elaborati) sono:

Tipo di file	Indirizzo e nome del file	Note
sistema	*prefix*/lib/python*ver*/distutils/distutils.cfg	(1)
personali	$HOME/.pydistutils.cfg	(2)
locali	setup.cfg	(3)

Su Windows, i file di configurazione sono:

Tipo di file	Indirizzo e nome del file	Note
sistema	*prefix*\Lib\distutils\distutils.cfg	(4)
personali	%HOME%\pydistutils.cfg	(5)
locali	setup.cfg	(3)

E su Mac OS:

Tipo di file	Indirizzo e nome del file	Note
sistema	*prefix*:Lib:distutils:distutils.cfg	(6)
personali	N/A	
locali	setup.cfg	(3)

Note:

(1) Solo per dirlo, i file di configurazione di Distutils risiedono nella medesima directory di dove Distutils stesso è stato installato; è così in Python 1.6 e seguenti, su UNIX. Per Python 1.5.2, le Distutils vengono normalmente installate in '*prefix*/lib/python1.5/site-packages/distutils', così i file di configurazione del sistema dovrebbero essere messe lì, in Python 1.5.2.

(2) Su UNIX, se la variabile di ambiente HOME non viene definita, la directory home dell'utente verrà determinata dalla la funzione `getpwuid()`, dal modulo standard pwd.

(3) Per esempio, nella directory corrente (solitamente la posizione dello script di setup).

(4) (Vedete anche le note (1).) In Python 1.6 e successivi, il valore predefinito per il prefix d'installazione di Python è 'C:\Python', così i file di configurazione normalmente si trovano in 'C:\Python\Lib\distutils\distutils.cfg'. In Python 1.5.2, il valore predefinito per il prefix era 'C:\Program Files\Python' e le Distutils non erano parte della libreria standard—così il file di configurazione in un'installazione standard di Python sotto Windows dovrebbe essere 'C:\Program Files\Python\distutils\distutils.cfg'.

(5) Su Windows, se la variabile di ambiente HOME non viene definita, non verrà trovato o usato nessun file di configurazione. (In altre parole, le Distutils non si apetteranno la vostra directory home in Windows).

(6)

(Vedete anche note (1) e (4).) Il valore predefinito per il prefix d'installazione è solo 'Python:', così sotto Python 1.6 e successivi è normalmente 'Python:Lib:distutils:distutils.cfg'.

5.2 Sintassi dei file di configurazione

I file di configurazione di Distutils hanno tutti la medesima sintassi. I file di configurazione vengono riuniti all'interno di sezioni. C'è una sezione per ogni comando Distutils, più una sezione `global` per opzioni globali che interessano ogni comando. Ogni sezione consiste di un'opzione per riga, specificata come `option=value`.

Per esempio, il seguente è un file di configurazione completo che forza tutti i comandi ad un'esecuzione non prolissa in modo predefinito:

```
[global]
verbose=0
```

Se questo viene installato come il file di configurazione del sistema, interesserà tutti i processi di tutti i moduli della distribuzione Python per ogni utente del sistema. Se viene installato come un vostro personale file di configurazione (nei sistemi che lo supportano), interesserà solo il modulo della distribuzione elaborato per voi. Se viene usato come il 'setup.cfg' per un particolare modulo della distribuzione, interesserà solo quella distribuzione.

Potete sovrascrivere la directory predefinita "build base" ed eseguire i comandi `build*` forzando sempre la ricostruzione di tutti i file nel seguente modo:

```
[build]
build-base=blib
force=1
```

Che corrisponde agli argomenti da riga di comando

```
python setup.py build --build-base=blib --force
```

ad eccezione di quelli che comprendono il comando `build` da riga di comando, ovvero, quel particolare comando verrà comunque eseguito. L'inclusione di un particolare comando nel file di configurazione non ha implicazioni; significa soltanto che se il comando viene eseguito, l'opzione nel file di configurazione verrà applicata. (O se altri comandi che traggono valori da esso vengono eseguiti, useranno i valori del file di configurazione.)

Potete trovare un elenco completo delle opzioni per ogni comando usando l'opzione **--help**, per esempio:

```
python setup.py build --help
```

e troverete l'elenco completo delle opzioni globali da usare.

L'**--help** senza un comando:

```
python setup.py --help
```

Vedete anche la sezione "Riferimenti" del manuale "Distribuire moduli Python"

6 Compilare le estensioni: trucchi e suggerimenti

Quando possibile, Distutils tenta di usare le informazioni di configurazione rese disponibili dall'interprete Python usato per eseguire lo script 'setup.py'. Per esempio, le stesse opzioni del compilatore e del linker usate per compilare Python verranno usate anche per compilare le estensioni. Di solito questo lavora bene, ma in situazioni complicate può essere inappropriato. Questa sezione discute su come annullare il comportamento predefinito di Distutils.

6.1 Modificare le opzioni di compilazione/linker

Compilando un'estensione Python scritta in C o C++ verranno richieste specifiche e personalizzate opzioni per il compilatore ed il linker, per usare una particolare libreria o produrre un tipo speciale di codice oggetto. Questo è vero, specialmente se l'estensione non è stata testata sulla vostra piattaforma, o se state provando a compilare Python per più piattaforme.

Nella maggior parte dei casi, l'autore dell'estensione potrebbe avere previsto che la compilazione dell'estensione potrebbe essere complicata ed allora fornirà un apposito file 'Setup' da editare. Questo verrà fatto solo se il modulo distribuito contiene molti moduli di estensione separati, o se vengono richieste spesso impostazioni elaborate per le opzioni del compilatore in uso.

Un file 'Setup', se presente, viene analizzato per accettare un elenco delle estensioni da costruire. Ogni riga in un file 'Setup' descrive un singolo modulo. Le righe hanno la seguente struttura:

module `...` [*sourcefile* `...`] [*cpparg* `...`] [*library* `...`]

Esaminerà ogni riga del campo alla volta.

- *module* è il nome del modulo di estensione che deve essere costruito e dovrebbe essere un identificatore Python valido. Non potete cambiare solo questo per rinominare un modulo (sarebbe necessario editare il codice sorgente), così dovrebbe essere lasciato stare.
- *sourcefile* è del tutto simile ad un file di codice sorgente, giudicando dal nome del file. I nomi dei file che finiscono in '.c' si assume che siano scritti in C, i nomi dei file che finiscono in '.C', '.cc' e '.c++' si assumono essere in C++ ed i nomi dei file che finiscono in '.mm' o '.m' si assume che siano scritti in Objective C.
- *cpparg* è un argomento per il preprocessore C e rappresenta tutto ciò che inizia con **-I**, **-D**, **-U** o **-C**.
- *library* rappresenta tutto ciò che finisce con '.a' o inizia con **-l** o **-L**.

Se una particolare piattaforma richiede la presenza di una particolare libreria sulla vostra piattaforma, potete aggiungerla editando il file 'Setup' ed eseguendo `python setup.py build`. Per esempio, se il modulo definito dalla riga

```
foo foomodule.c
```

e dovesse essere linkato con la libreria math 'libm.a' sulla vostra piattaforma, aggiungerete semplicemente **-lm** alla riga:

```
foo foomodule.c -lm
```

Opzioni arbitrarie predisposte per il compilatore od il linker possono essere fornite mediante **-Xcompiler** *arg* e le opzioni **-Xlinker** *arg*:

```
foo foomodule.c -Xcompiler -o32 -Xlinker -shared -lm
```

La prossima opzione dopo **-Xcompiler** e **-Xlinker** verrà aggiunta all'appropriata riga di comando, così nel suddetto esempio il compilatore passerà l'opzione **-o32** ed il linker passerà **-shared**. Se l'opzione di un compilatore richiede un argomento, dovrete fornire opzioni multiple **-Xcompiler**; per esempio, per passare `-x c++` il file 'Setup' dovrebbe contenere `-Xcompiler -x -Xcompiler c++`.

Le opzioni di compilazione possono essere fornite attraverso l'impostazione della variabile d'ambiente CFLAGS. Se impostata, il contenuto di CFLAGS verrà aggiunto alle opzioni del compilatore, specificate nel file 'Setup'.

6.2 Usare compilatori non-Microsoft su Windows

Borland C++

Questa sezione descrive i passi necessari per usare Distutils con il compilatore Borland C++ versione 5.5.

Prima di tutto dovete sapere che il formato dei file oggetto di Borland è differente dal formato usato dalla versione Python che potete scaricare dal sito web di Python o di ActiveState. (Python viene compilato con Microsoft Visual C++, che usa COFF come formato dei file oggetto.) Per questa ragione dovete convertire la libreria di Python 'python20.lib' nel formato Borland. Potete farlo in questo modo:

```
coff2omf python20.lib python20_bcpp.lib
```

Il programma 'coff2omf' viene fornito insieme al compilatore Borland. Il file 'python20.lib' è nella directory 'Libs' della vostra installazione Python. Se la vostra estensione usa altre librerie (zlib,...) dovrete convertire anche queste.

I file convertiti devono risiedere nella stessa directory delle normali librerie.

Come gestisce Distutils l'uso di queste librerie con i loro nomi cambiati? Se l'estensione necessita di una libreria (per esempio 'foo') Distutils cerca prima di trovare la libreria con il suffisso '_bcpp' (per esempio 'foo_bcpp.lib') e quindi usa questa libreria. Nel caso in cui non la trovasse, come può accadere per una libreria particolare, userà il nome predefinito ('foo.lib')[1].

Per fare compilare a Distutils le vostre estensioni con Borland C++ dovete ora digitare:

```
python setup.py build --compiler=bcpp
```

Se volete usare il compilatore Borland C++ come predefinito, potete specificarlo nel file di configurazione di sistema per Distutils (vedete la sezione 5.)

Vedete anche:

C++Builder Compiler
(http://www.borland.com/bcppbuilder/freecompiler/)
Informazioni sul compilatore libero C++ di Borland, inclusi i link alle pagine dei download.

Creating Python Extensions Using Borland's Free Compiler
(http://www.cyberus.ca/~g_will/pyExtenDL.shtml)
Il documento descrive come usare la riga di comando del compilatore C++ libero della Borland in Python.

GNU C / Cygwin / MinGW

Questa sezione descrive i passi necessari per usare Distutils con i compilatori GNU C/C++ nelle loro distribuzioni Cygwin e MinGW.[2] Se l'interprete Python è stato compilato con Cygwin, dovrebbe funzionare tutto, rendendo inutile leggere i passi successivamente descritti.

Questi compilatori hanno bisogno di alcune librerie speciali. Questa prova è più complessa di quella per il C++ di Borland, perché non c'è un programma per convertire le librerie.

Prima dovete creare una lista di simboli che le DLL Python esporteranno. Potete trovare un buon programma per fare questa prova presso http://starship.python.net/crew/kernr/mingw32/Notes.html, vedete PExports 0.42h.

```
pexports python20.dll >python20.def
```

Quindi create da queste informazioni una libreria di importazione per gcc.

```
dlltool --dllname python20.dll --def python20.def --output-lib libpython20.a
```

[1] Questo significa anche che potete sostituire tutte le esistenti librerie COFF con le librerie OMF con lo stesso nome.
[2] Controllate http://sources.redhat.com/cygwin/ e http://www.mingw.org/ per maggiori informazioni

La libreria risultante verrà posizionata nella stessa directory di ‘python20.lib’. Dovrebbe essere la directory ‘libs’ sotto la vostra directory Python d’installazione.

Se le vostre estensioni usano altre librerie (zlib,...) dovrete convertire anche quelle. I file convertiti dovranno risiedere nella stessa directory come in una normale libreria. Per fare compilare le vostre estensioni a Distutils con Cygwin dovete ora digitare

```
python setup.py build --compiler=cygwin
```

e per Cygwin in modalità no-cygwin [3] o per MinGW digitate:

```
python setup.py build --compiler=mingw32
```

Se volete usare ognuna di queste opzioni/compilatori in modo predefinito, dovreste prendere in considerazione la possibilità di scriverle nel vostro file di configurazione globale per Distutils (vedete la sezione 5.)

Vedete anche:

Building Python modules on MS Windows platform with MinGW
(http://www.zope.org/Members/als/tips/win32_mingw_modules)
: Informazioni sulla compilazione delle librerie richieste per l’ambiente MinGW.

http://pyopengl.sourceforge.net/ftp/win32-stuff/
: Conversione ed importazione di librerie nel formato Cygwin/MinGW e Borland ed uno script per creare il registro delle voci necessarie a Distutils per localizzare dove è stata compilata o installata la versione in uso di Python.

[3] Quindi non avete un’emulazione POSIX disponibile, ma non avete neanche bisogno di ‘cygwin1.dll’.