

Il progetto GNU

di Richard Stallman

originariamente pubblicato sul libro *Open Sources*

La prima comunità di condivisione del software

Quando cominciai a lavorare nel laboratorio di Intelligenza Artificiale del MIT [N.d.T. Massachusetts Institute of Technology] nel 1971, entrai a far parte di una comunità in cui ci si scambiavano i programmi, che esisteva già da molti anni. La condivisione del software non si limitava alla nostra comunità; è un cosa vecchia quanto i computer, proprio come condividere le ricette è antico come il cucinare. Ma noi lo facevamo più di quasi chiunque altro.

Il laboratorio di Intelligenza Artificiale usava un sistema operativo a partizione di tempo (timesharing) chiamato ITS (Incompatible Timesharing System) che il gruppo di hacker (1) del laboratorio aveva progettato e scritto in linguaggio assembler per il Digital PDP-10, uno dei grossi elaboratori di quel periodo. Come membro di questa comunità, hacker di sistema nel gruppo laboratorio, il mio compito era migliorare questo sistema.

Non chiamavamo il nostro software "software libero", poiché questa espressione ancora non esisteva, ma si trattava proprio di questo. Quando persone di altre università o di qualche società volevano convertire il nostro programma per il proprio sistema ed utilizzarlo, erano le benvenute. Se si vedeva qualcuno usare un programma sconosciuto ed interessante, si poteva sempre chiedere di vederne il codice sorgente, in modo da poterlo leggere, modificare, o prenderne cannibalizzarne alcune parti per creare un nuovo programma.

(1) L'uso del termine "hacker" nel senso di "pirata" è una confusione di termini creata dai mezzi di informazione. Noi hacker ci rifiutiamo di riconoscere questo significato, e continuiamo ad utilizzare la parola nel senso di "uno che ami programmare, e a cui piaccia essere bravo a farlo"

La comunità si dissolve

La situazione cambiò drasticamente all'inizio degli anni '80 quando la Digital smise di produrre la serie PDP-10. La sua architettura, elegante e potente negli anni '60, non poteva essere estesa in modo naturale ai più grandi spazi di indirizzamento che si stavano rendendo possibili negli anni '80. Questo significò che quasi tutti i programmi che formavano ITS divennero obsoleti.

La comunità di hacker del laboratorio di Intelligenza Artificiale si era già dissolta non molto tempo prima. Nel 1981 la Symbolics, nata da una costola del laboratorio stesso, gli aveva sottratto quasi tutti gli hacker; l'ormai esiguo gruppo rimasto fu

dunque incapace di sostenersi (il libro "Hackers" di Steve Levy narra questi eventi, oltre a fornire una fedele ricostruzione di questa comunità ai suoi inizi). Quando il laboratorio di Intelligenza Artificiale nel 1982 acquistò un nuovo PDP-10, i sistemisti decisero di utilizzare il sistema timesharing non libero della Digital piuttosto che ITS.

I moderni elaboratori di quell'epoca, come il VAX o il 68020, avevano il proprio sistema operativo, ma nessuno di questi era libero: si doveva firmare un accordo di non-diffusione persino per ottenerne una copia eseguibile.

Questo significava che il primo passo per usare un computer era promettere di negare aiuto al proprio vicino. Una comunità cooperante era vietata. La regola creata dai proprietari di software proprietario era: «se condividi il software col tuo vicino sei un pirata. Se vuoi modifiche, pregaci di farle».

L'idea che la concezione sociale di software proprietario -- cioè il sistema che impone che il software non possa essere condiviso o modificato -- sia antisociale, contraria all'etica, semplicemente sbagliata, può apparire sorprendente a qualche lettore. Ma che altro possiamo dire di un sistema che si basa sul dividere utenti e lasciarli senza aiuto? Quei lettori che trovano sorprendente l'idea possono aver data per scontata la concezione sociale di software proprietario, o averla giudicata utilizzando lo stesso metro suggerito dal mercato del software proprietario. I produttori di software hanno lavorato a lungo e attivamente per diffondere la convinzione che c'è un solo modo di vedere la cosa.

Quando i produttori di software parlano di "difendere" i propri "diritti" o di "fermare la pirateria", quello che dicono è in realtà secondario. Il vero messaggio in quelle affermazioni sta nelle assunzioni inesprese, che essi danno per scontate; vogliono che siano accettate acriticamente. Esaminiamole, dunque.

Una prima assunzione è che le aziende produttrici di software abbiano il diritto naturale indiscutibile di proprietà sul software, e di conseguenza, abbiano controllo su tutti i suoi utenti. Se questo fosse un diritto naturale, non potremmo sollevare obiezioni, indipendentemente dal danno che possa recare ad altri. È interessante notare che, negli Stati Uniti, sia la costituzione che la giurisprudenza rifiutano questa posizione: il diritto d'autore non è un diritto naturale, ma un monopolio imposto dal governo che limita il diritto naturale degli utenti ad effettuare delle copie.

Un'altra assunzione inespressa è che la sola cosa importante del software sia il lavoro che consente di fare -- vale a dire che noi utenti non dobbiamo preoccuparci del tipo di società in cui ci è permesso vivere.

Una terza assunzione è che non avremmo software utilizzabile (o meglio, che non potremmo mai avere un programma per fare questo o quell'altro particolare lavoro) se non riconoscessimo ai produttori il controllo sugli utenti di quei programmi. Questa assunzione avrebbe potuto sembrare plausibile, prima che il movimento del software libero dimostrasse che possiamo scrivere quantità di programmi utili senza bisogno di metterci dei catenacci.

Se rifiutiamo di accettare queste assunzioni, giudicando queste questioni con comuni criteri di moralità e di buon senso dopo aver messo al primo posto gli interessi degli utenti, tenendo conto che gli utenti vengono prima di tutto, arriviamo a conclusioni del tutto differenti. Chi usa un calcolatore dovrebbe essere libero di modificare i programmi per adattarli alle proprie necessità, ed essere libero di condividere il software, poiché aiutare gli altri è alla base della società.

Non c'è modo in questa sede di trattare approfonditamente i ragionamenti che portano a questa conclusione; il lettore interessato può cercare le informazioni in rete a questo indirizzo: <http://www.gnu.org/philosophy/why-free.html>.

Una difficile scelta morale

Una volta che il mio gruppo si fu sciolto, continuare come prima fu impossibile. Mi trovai di fronte ad una difficile scelta morale.

La scelta facile sarebbe stata quella di unirsi al mondo del software proprietario, firmando accordi di non-diffusione e promettendo di non aiutare i miei compagni hacker. Con ogni probabilità avrei anche sviluppato software che sarebbe stato distribuito secondo accordi di non-diffusione, contribuendo così alla pressione su altri perché a loro volta tradissero i propri compagni.

In questo modo avrei potuto guadagnare, e forse mi sarei divertito a programmare. Ma sapevo che al termine della mia carriera mi sarei voltato a guardare indietro, avrei visto anni spesi a costruire muri per dividere le persone, e avrei compreso di aver contribuito a rendere il mondo peggiore.

Avevo già sperimentato cosa significasse un accordo di non diffusione per chi lo firmava, quando qualcuno rifiutò a me e al laboratorio AI del MIT il codice sorgente del programma di controllo della nostra stampante; l'assenza di alcune funzionalità nel programma rendeva oltremodo frustrante l'uso della stampante. Per cui non mi potevo dire che gli accordi di non-diffusione fossero innocenti. Ero molto arrabbiato quando quella persona si rifiutò di condividere il programma con noi; non potevo far finta di niente e fare lo stesso con tutti gli altri.

Un'altra possibile scelta, semplice ma spiacevole, sarebbe stata quella di abbandonare l'informatica. In tal modo le mie capacità non sarebbero state mal utilizzate, tuttavia sarebbero state sprecate. Non sarei mai stato colpevole di dividere o imporre restrizioni agli utenti di calcolatori, ma queste cose sarebbero comunque successe.

Allora cercai un modo in cui un programmatore potesse fare qualcosa di buono. Mi chiesi dunque: c'erano un programma o dei programmi che io potessi scrivere, per rendere nuovamente possibile l'esistenza di una comunità?

La risposta era semplice: innanzitutto serviva un sistema operativo. Questo è difatti il software fondamentale per iniziare ad usare un computer. Con un sistema operativo si possono fare molte cose; senza, non è proprio possibile far funzionare il computer. Con un sistema operativo libero, avremmo potuto avere nuovamente una comunità in cui hacker possono cooperare, e invitare chiunque ad unirsi al

gruppo. E chiunque sarebbe stato in grado di usare un calcolatore, senza dover cospirare fin dall'inizio per sottrarre qualcosa ai propri amici.

Essendo un programmatore di sistemi, possedevo le competenze adeguate per questo lavoro. Così, anche se non davo il successo per scontato, mi resi conto di essere la persona giusta per farlo. Scelsi di rendere il sistema compatibile con Unix, in modo che fosse portabile, e che gli utenti Unix potessero passare facilmente ad esso. Il nome GNU fu scelto secondo una tradizione hacker, come acronimo ricorsivo che significa "GNU's Not Unix" [N.d.T. GNU non è Unix].

Un sistema operativo non si limita solo al suo nucleo, che è proprio il minimo per eseguire altri programmi. Negli anni '70, qualsiasi sistema operativo degno di questo nome includeva interpreti di comandi, assembleri, compilatori, interpreti di linguaggi, debugger, editor di testo, programmi per la posta e molto altro. ITS li aveva, Multics li aveva, VMS li aveva e Unix li aveva. Anche il sistema operativo GNU li avrebbe avuti.

Tempo dopo venni a conoscenza di questa massima, attribuita a Hillel (1):

Se non sono per me stesso, chi sarà per me?
E se sono solo per me stesso, che cosa sono?
E se non ora, quando?

La decisione di iniziare il progetto GNU si basò su uno spirito simile.

(1) Essendo ateo, non seguo alcuna guida religiosa, ma a volte mi trovo ad ammirare qualcosa che qualcuno di loro ha detto.

"Free" come libero

Il termine "free software" [N.d.T. il termine free in inglese significa sia gratuito che libero] a volte è mal interpretato: non ha niente a che vedere col prezzo del software; si tratta di libertà. Ecco, dunque, la definizione di software libero: un programma è software libero per un dato utente se:

- * l'utente ha la libertà di eseguire il programma per qualsiasi scopo;
- * l'utente ha la libertà di modificare il programma secondo i propri bisogni (perché questa libertà abbia qualche effetto in pratica, è necessario avere accesso al codice sorgente del programma, poiché apportare modifiche ad un programma senza disporre del codice sorgente è estremamente difficile);
- * l'utente ha la libertà di distribuire copie del programma, gratuitamente o dietro compenso;
- * l'utente ha la libertà di distribuire versioni modificate del programma, così che la comunità possa fruire dei miglioramenti apportati.

Poiché "free" si riferisce alla libertà e non al prezzo, vendere copie di un programma non contraddice il concetto di software libero. In effetti, la libertà di vendere copie di programmi è essenziale: raccolte di software libero vendute su CD-ROM sono importanti per la comunità, e la loro vendita è un modo di raccogliere

fondi importante per lo sviluppo del software libero. Di conseguenza, un programma che non può essere liberamente incluso in tali raccolte non è software libero.

A causa dell'ambiguità del termine "free", si è cercata a lungo un'alternativa, ma nessuno ne ha trovata una valida. La lingua inglese ha, più termini e sfumature di ogni altra, ma non ha una parola semplice e non ambigua che significhi libero; "unfettered" è la parola più vicina come significato [NdT: unfettered è una parola di tono aulico o arcaico che significa libero da ceppi, vincoli o inibizioni]. Alternative come "liberated", "freedom" e "open" hanno altri significati o non sono adatte per altri motivi [NdT: rispettivamente, liberato, libertà, aperto].

Software GNU e il sistema GNU

Sviluppare un intero sistema è un progetto considerevole. Per raggiungere l'obiettivo decisi di adattare e usare parti di software libero tutte le volte che fosse possibile. Per esempio, decisi fin dall'inizio di usare TeX come il principale programma di formattazione di testo; qualche anno più tardi, decisi di usare l'X Window System piuttosto che scrivere un altro sistema a finestre per GNU.

A causa di questa decisione, il sistema GNU e la raccolta di tutto il software GNU non sono la stessa cosa. Il sistema GNU comprende programmi che non sono GNU, sviluppati da altre persone o gruppi di progetto per i propri scopi, ma che possiamo usare in quanto software libero.

L'inizio del progetto

Nel gennaio 1984 lasciai il mio posto al MIT e cominciai a scrivere software GNU. Dovetti lasciare il MIT, per evitare che potesse interferire con la distribuzione di GNU come software libero. Se fossi rimasto, il MIT avrebbe potuto rivendicare la proprietà del lavoro, ed avrebbe potuto imporre i propri termini di distribuzione, o anche farne un pacchetto proprietario. Non avevo alcuna intenzione di fare tanto lavoro solo per vederlo reso inutilizzabile per il suo scopo originario: creare una nuova comunità di condivisione di software. Ad ogni buon conto, il professor Winston -- allora responsabile del laboratorio AI del MIT -- mi propose gentilmente di continuare ad utilizzare le attrezzature del laboratorio stesso.

I primi passi

Poco dopo aver iniziato il progetto GNU, venni a sapere del Free University Compiler Kit, noto anche come VUCK (la parola olandese che sta per "free" inizia con la V). Era un compilatore progettato per trattare più linguaggi, fra cui C e Pascal, e per generare codice binario per diverse architetture. Scrissi al suo autore chiedendo se GNU avesse potuto usarlo. Rispose in modo canzonatorio, dicendo che l'università era sì libera, ma non il compilatore. Decisi allora che il mio primo

programma per il progetto GNU sarebbe stato un compilatore multilinguaggio e multiplatforma.

Sperando di evitare di dover scrivere da me l'intero compilatore, ottenni il codice sorgente del Pastel, un compilatore multiplatforma sviluppato ai Laboratori Lawrence Livermore. Il linguaggio supportato da Pastel, in cui il Pastel stesso era scritto, era una versione estesa del Pascal, pensata come linguaggio di programmazione di sistemi. Io vi aggiunsi un frontend per il C, e cominciai il porting per il processore Motorola 68000, ma fui costretto a rinunciare quando scoprii che il compilatore richiedeva diversi megabyte di memoria sullo stack, mentre il sistema Unix disponibile per il processore 68000 ne permetteva solo 64K.

Mi resi conto allora che il compilatore Pastel interpretava tutto il file di ingresso creandone un albero sintattico, convertiva questo in una catena di "istruzioni", e quindi generava l'intero file di uscita senza mai liberare memoria. A questo punto, conclusi che avrei dovuto scrivere un nuovo compilatore da zero. Quel nuovo compilatore è ora noto come Gcc; non utilizza niente del compilatore Pastel, ma riuscii ad adattare e riutilizzare il frontend per il C che avevo scritto. Questo però avvenne qualche anno dopo; prima, lavorai su GNU Emacs.

GNU Emacs

Cominciai a lavorare su GNU Emacs nel settembre 1984, e all'inizio del 1985 cominciava ad essere utilizzabile. Così potei iniziare ad usare sistemi Unix per scrivere; fino ad allora, avevo scritto sempre su altri tipi di macchine, non avendo nessun interesse ad imparare vi né ed.

A questo punto alcuni cominciarono a voler usare GNU Emacs, il che pose il problema di come distribuirlo. Naturalmente lo misi sul server ftp anonimo del computer che usavo al MIT (questo computer, prep.ai.mit.edu, divenne così il sito ftp primario di distribuzione di GNU; quando alcuni anni dopo andò fuori servizio, trasferimmo il nome sul nostro nuovo ftp server). Ma allora molte delle persone interessate non erano su Internet e non potevano ottenere una copia via ftp, così mi si pose il problema di cosa dir loro.

Avrei potuto dire: «trova un amico che è in rete disposto a farti una copia». Oppure avrei potuto fare quel che feci con l'originario Emacs su PDP-10, e cioè dir loro: «spediscimi una busta affrancata ed un nastro, ed io te lo rispedisco con sopra Emacs». Ma ero senza lavoro, e cercavo un modo di far soldi con il software libero. E così feci sapere che avrei spedito un nastro a chi lo voleva per 150 dollari. In questo modo, creai un'impresa di distribuzione di software libero, che anticipava le compagnie che oggi distribuiscono interi sistemi GNU basati su Linux.

Un programma è libero per tutti?

Se un programma è software libero quando esce dalle mani del suo autore, non significa necessariamente che sarà software libero per chiunque ne abbia una copia. Per esempio, il software di pubblico dominio (software senza copyright) è

software libero, ma chiunque può farne una versione modificata proprietaria. Analogamente, molti programmi liberi sono protetti da diritto d'autore, ma vengono distribuiti con semplici licenze permissive che permettono di farne versioni modificate proprietarie.

L'esempio emblematico della questione è l'X Window System. Sviluppato al MIT, e pubblicato come software libero con una licenza permissiva, fu rapidamente adottato da diverse società informatiche. Queste aggiunsero X ai loro sistemi Unix proprietari, solo in forma binaria, e coperto dello stesso accordo di non-diffusione. Queste copie di X non erano software più libero di quanto lo fosse Unix.

Gli autori dell'X Window System non ritenevano che questo fosse un problema, anzi se lo aspettavano ed era loro intenzione che accadesse. Il loro scopo non era la libertà, ma semplicemente il "successo", definito come "avere tanti utenti". Non erano interessati che questi utenti fossero liberi, ma solo che fossero numerosi.

Questo sfociò in una situazione paradossale, in cui due modi diversi di misurare la quantità di libertà risultavano in risposte diverse alla domanda «questo programma è libero»? Giudicando sulla base della libertà offerta dai termini distributivi usati dal MIT, si sarebbe dovuto dire che X era software libero. Ma misurando la libertà dell'utente medio di X, si sarebbe dovuto dire che X era software proprietario. La maggior parte degli utenti di X usavano le versioni proprietarie fornite con i sistemi Unix, non la versione libera.

Il permesso d'autore (copyleft) e la GNU GPL

Lo scopo di GNU consisteva nell'offrire libertà agli utenti, non solo nell'ottenere ampia diffusione. Avevamo quindi bisogno di termini di distribuzione che evitassero che il software GNU fosse trasformato in software proprietario. Il metodo che usammo si chiama "permesso d'autore"(1).

Il permesso d'autore (copyleft)(2) usa le leggi sul diritto d'autore (copyright), ma le capovolge per ottenere lo scopo opposto: invece che un metodo per privatizzare il software, diventa infatti un mezzo per mantenerlo libero.

Il succo dell'idea di permesso d'autore consiste nel dare a chiunque il permesso di eseguire il programma, copiare il programma, modificare il programma, e distribuirne versioni modificate, ma senza dare il permesso di aggiungere restrizioni. In tal modo, le libertà essenziali che definiscono il "free software" (software libero) sono garantite a chiunque ne abbia una copia, e diventano diritti inalienabili.

Perché un permesso d'autore sia efficace, anche le versioni modificate devono essere libere. Ciò assicura che ogni lavoro basato sul nostro sia reso disponibile per la nostra comunità, se pubblicato. Quando dei programmatori professionisti lavorano su software GNU come volontari, è il permesso d'autore che impedisce ai loro datori di lavoro di dire: «non puoi distribuire quei cambiamenti, perché abbiamo intenzione di usarli per creare la nostra versione proprietaria del programma».

La clausola che i cambiamenti debbano essere liberi è essenziale se vogliamo garantire libertà a tutti gli utenti del programma. Le aziende che privatizzarono l'X Window System di solito avevano apportato qualche modifica per portare il programma sui loro sistemi e sulle loro macchine. Si trattava di modifiche piccole rispetto alla mole di X, ma non banali. Se apportare modifiche fosse una scusa per negare libertà agli utenti, sarebbe facile per chiunque approfittare di questa scusa.

Una problematica correlata è quella della combinazione di un programma libero con codice non libero. Una tale combinazione sarebbe inevitabilmente non libera; ogni libertà che manchi dalla parte non libera mancherebbe anche dall'intero programma. Permettere tali combinazioni aprirebbe non uno spiraglio, ma un buco grosso come una casa. Quindi un requisito essenziale per il permesso d'autore è tappare il buco: tutto ciò che venga aggiunto o combinato con un programma protetto da permesso d'autore dev'essere tale che il programma risultante sia anch'esso libero e protetto da permesso d'autore.

La specifica implementazione di permesso d'autore che utilizziamo per la maggior parte del software GNU è la GNU General Public License (licenza pubblica generica GNU), abbreviata in GNU GPL. Abbiamo altri tipi di permesso d'autore che sono utilizzati in circostanze specifiche. I manuali GNU sono anch'essi protetti da permesso d'autore, ma ne usano una versione molto più semplice, perché per i manuali non è necessaria la complessità della GPL.

(1) Nel 1984 o 1985, Don Hopkins, persona molto creativa, mi mandò una lettera. Sulla busta aveva scritto diverse frasi argute, fra cui questa: "Permesso d'autore--tutti i diritti rovesciati". Utilizzai l'espressione "permesso d'autore" per battezzare il concetto di distribuzione che allora andavo elaborando.

(2) [NdT: si tratta di un gioco di parole, che qui viene reso con "permesso di autore": copyright (diritto di autore) è formato dalle parole "copy" (copia) e "right" (diritto, ma anche destra), opposto di "left" (sinistra, ma anche lasciato).]

La Free Software Foundation

Man mano che l'interesse per Emacs aumentava, altre persone parteciparono al progetto GNU, e decidemmo che era di nuovo ora cercare finanziamenti. Così nel 1985 fondammo la Free Software Foundation (Fondazione per il software libero), una organizzazione senza fini di lucro per lo sviluppo di software libero. La FSF fra l'altro si prese carico della distribuzione dei nastri di Emacs; più tardi estese l'attività aggiungendo sul nastro altro software libero (sia GNU che non GNU) e vendendo manuali liberi.

La FSF accetta donazioni, ma gran parte delle sue entrate è sempre stata costituita dalle vendite: copie di software libero e servizi correlati. Oggi vende CD-ROM di codice sorgente, CD-ROM di programmi compilati, manuali stampati professionalmente (tutti con libertà di ridistribuzione e modifica), e distribuzioni Deluxe (nelle quali compiliamo l'intera scelta di software per una piattaforma a richiesta).

I dipendenti della Free Software Foundation hanno scritto e curato la manutenzione di diversi pacchetti GNU. Fra questi spiccano la libreria C e la shell. La libreria C di GNU è utilizzata da ogni programma che gira su sistemi GNU/Linux per comunicare con Linux. È stata sviluppata da un membro della squadra della Free Software Foundation, Roland McGrath. La shell usata sulla maggior parte dei sistemi GNU/Linux è Bash, la Bourne Again Shell(1), che è stata sviluppata da Brian Fox, dipendente della FSF.

Finanziammo lo sviluppo di questi programmi perché il progetto GNU non riguardava solo strumenti di lavoro o un ambiente di sviluppo: il nostro obiettivo era un sistema operativo completo, e questi programmi erano necessari per raggiungere quell'obiettivo.

(1) "Bourne Again Shell" è un gioco di parole sul nome "Bourne Shell", che era la normale shell di Unix [NdT: "Bourne again" richiama l'espressione cristiana "born again", "rinato" (in Cristo)].

Il supporto per il software libero

La filosofia del software libero rigetta una diffusa pratica commerciale in particolare, ma non è contro il commercio. Quando un'impresa rispetta la libertà dell'utente, c'è da augurarle ogni successo.

La vendita di copie di Emacs esemplifica un modo di condurre affari col software libero. Quando la FSF prese in carico quest'attività, dovette trovare un'altra fonte di sostentamento. La trovai nella vendita di servizi relativi al software libero che avevo sviluppato, come insegnare argomenti quali programmazione di Emacs e personalizzazione di GCC, oppure sviluppare software, soprattutto adattamento di GCC a nuove architetture.

Oggi tutte queste attività collegate al software libero sono esercitate da svariate aziende. Alcune distribuiscono raccolte di software libero su CD-ROM, altre offrono consulenza a diversi livelli, dall'aiutare gli utenti in difficoltà, alla correzione di errori, all'aggiunta di funzionalità non banali. Si cominciano anche a vedere aziende di software che si fondano sul lancio di nuovi programmi liberi.

Attenzione, però: diverse aziende che si fregiano del marchio "open source" (software aperto) in realtà fondano le loro attività su software non libero che funziona insieme con software libero. Queste non sono aziende di software libero, sono aziende di software proprietario i cui prodotti attirano gli utenti lontano dalla libertà. Loro li chiamano "a valore aggiunto", il che riflette i valori che a loro farebbe comodo che adottassimo: la convenienza prima della libertà. Se noi riteniamo che la libertà abbia più valore, li dovremmo chiamare prodotti "a libertà sottratta".

Obiettivi tecnici

L'obiettivo principale di GNU era essere software libero. Anche se GNU non avesse avuto alcun vantaggio tecnico su Unix, avrebbe avuto sia un vantaggio sociale, permettendo agli utenti di cooperare, sia un vantaggio etico, rispettando la loro libertà.

Tuttavia risultò naturale applicare al lavoro le regole classiche di buona programmazione; per esempio, allocare le strutture dati dinamicamente per evitare limitazioni arbitrarie sulla dimensione dei dati, o gestire tutti i possibili codici a 8 bit in tutti i casi ragionevoli.

Inoltre, al contrario di Unix che era pensato per piccole dimensioni di memoria, decidemmo di non supportare le macchine a 16 bit (era chiaro che le macchine a 32 bit sarebbero state la norma quando il sistema GNU sarebbe stato completo), e di non preoccuparci di ridurre l'occupazione di memoria a meno che eccedesse il megabyte. In programmi per i quali non era essenziale la gestione di file molto grandi, spingemmo i programmatori a leggere in memoria l'intero file di ingresso per poi analizzare il file senza doversi preoccupare delle operazioni di I/O.

Queste decisioni fecero sì che molti programmi GNU superassero i loro equivalenti Unix sia in affidabilità che in velocità di esecuzione.

Donazioni di computer

Man mano che la reputazione del progetto GNU andava crescendo, alcune persone iniziarono a donare macchine su cui girava Unix. Queste macchine erano molto utili, perché il modo più semplice di sviluppare componenti per GNU era di farlo su di un sistema Unix così da sostituire pezzo per pezzo i componenti di quel sistema. Ma queste macchine sollevavano anche una questione etica: se fosse giusto per noi anche solo possedere una copia di Unix.

Unix era (ed è) software proprietario, e la filosofia del progetto GNU diceva che non avremmo dovuto usare software proprietario. Ma, applicando lo stesso ragionamento per cui la violenza è ammessa per autodifesa, conclusi che fosse legittimo usare un pacchetto proprietario, se ciò fosse stato importante nel crearne un sostituto libero che permettesse ad altri di smettere di usare quello proprietario.

Tuttavia, benchè fosse un male giustificabile, era pur sempre un male. Oggi non abbiamo più alcuna copia di Unix, perché le abbiamo sostituite con sistemi operativi liberi. Quando non fu possibile sostituire il sistema operativo di una macchina con uno libero, sostituimmo la macchina.

L'elenco dei compiti GNU

Mentre il progetto GNU avanzava, ed un numero sempre maggiore di componenti di sistema venivano trovati o sviluppati, diventò utile stilare un elenco delle parti ancora mancanti. Usammo questo elenco per ingaggiare programmatori che scrivessero tali parti, e l'elenco prese il nome di elenco dei compiti GNU. In aggiunta ai componenti Unix mancanti inserimmo nell'elenco svariati progetti utili

di programmazione o di documentazione che a nostro parere non dovrebbero mancare in un sistema operativo veramente completo.

Oggi non compare quasi nessun componente Unix nell'elenco dei compiti GNU; tutti questi lavori, a parte qualcuno non essenziale, sono già stati svolti. D'altro canto l'elenco è pieno di quei progetti che qualcuno chiamerebbe "applicazioni": ogni programma che interessi ad una fetta non trascurabile di utenti sarebbe un'utile aggiunta ad un sistema operativo.

L'elenco comprende anche dei giochi, e così è stato fin dall'inizio: Unix comprendeva dei giochi, perciò era naturale che così fosse anche per GNU. Ma poiché non c'erano esigenze di compatibilità per i giochi, non ci attenemmo alla scelta di giochi presenti in Unix, preferendo piuttosto fornire un elenco di diversi tipi di giochi potenzialmente graditi agli utenti.

La licenza GNU per le librerie

La libreria C del sistema GNU utilizza un tipo speciale di permesso d'autore, la "Licenza Pubblica GNU per le Librerie"(1), che permette l'uso della libreria da parte di software proprietario. Perché quest'eccezione?

Non si tratta di questioni di principio: non c'è nessun principio che dica che i prodotti software proprietari abbiano il diritto di includere il nostro codice (perché contribuire ad un progetto fondato sul rifiuto di condividere con noi?). L'uso della licenza LGPL per la libreria C, o per qualsiasi altra libreria, è una questione di strategia.

La libreria C svolge una funzione generica: ogni sistema operativo proprietario ed ogni compilatore includono una libreria C. Di conseguenza, rendere disponibile la nostra libreria C solo per i programmi liberi non avrebbe dato nessun vantaggio a tali programmi liberi, avrebbe solo disincentivato l'uso della nostra libreria.

C'è un'eccezione a questa situazione: sul sistema GNU (termine che include GNU/Linux) l'unica libreria C disponibile è quella GNU. Quindi i termini di distribuzione della nostra libreria C determinano se sia possibile o meno compilare un programma proprietario per il sistema GNU. Non ci sono ragioni etiche per permettere l'uso di applicazioni proprietarie sul sistema GNU, ma strategicamente sembra che impedirne l'uso servirebbe più a scoraggiare l'uso del sistema GNU che non a incoraggiare lo sviluppo di applicazioni libere.

Ecco perché l'uso della licenza LGPL è una buona scelta strategica per la libreria C, mentre per le altre librerie la strategia va valutata caso per caso. Quando una libreria svolge una funzione particolare che può aiutare a scrivere certi tipi di programmi, distribuirla secondo la GPL, quindi limitandone l'uso ai soli programmi liberi, è un modo per aiutare gli altri autori di software libero, dando loro un vantaggio nei confronti del software proprietario.

Prendiamo come esempio GNU-Readline, una libreria scritta per fornire a Bash la modificabilità della linea di comando: Readline è distribuita secondo la normale licenza GPL, non la LGPL. Ciò probabilmente riduce l'uso di Readline, ma questo

non rappresenta una perdita per noi; d'altra parte almeno una applicazione utile è stata resa software libero proprio al fine di usare Readline, e questo è un guadagno tangibile per la comunità.

Chi sviluppa software proprietario ha vantaggi economici, gli autori di programmi liberi hanno bisogno di avvantaggiarsi a vicenda. Spero che un giorno possiamo avere una grande raccolta di librerie coperte dalla licenza GPL senza che esista una raccolta equivalente per chi scrive software proprietario. Tale libreria fornirebbe utili moduli da usare come i mattoni per costruire nuovi programmi liberi, e costituendo un sostanziale vantaggio per la scrittura di ulteriori programmi liberi.

(1) [NdT: nel 1999 la FSF ha cambiato nome alla licenza LGPL che ora si chiama "Lesser GPL", GPL attenuata, per non suggerire che si tratti della forma di licenza preferenziale per le librerie.]

Togliersi il prurito?

Eric Raymond afferma che «ogni buon programma nasce dall'iniziativa di un programmatore che si vuole togliere un suo personale prurito». È probabile che talvolta succeda così, ma molte parti essenziali del software GNU sono state sviluppate al fine di completare un sistema operativo libero. Derivano quindi da una idea e da un progetto, non da una necessità contingente.

Per esempio, abbiamo sviluppato la libreria C di GNU perché un sistema di tipo Unix ha bisogno di una libreria C, la Bourne-Again Shell (bash) perché un sistema di tipo Unix ha bisogno di una shell, e GNU tar perché un sistema di tipo Unix ha bisogno un programma tar. Lo stesso vale per i miei programmi: il compilatore GNU, GNU Emacs, GDB, GNU Make.

Alcuni programmi GNU sono stati sviluppati per fronteggiare specifiche minacce alla nostra libertà: ecco perché abbiamo sviluppato gzip come sostituto per il programma Compress, che la comunità aveva perduto a causa dei brevetti sull'algoritmo LZW. Abbiamo trovato persone che sviluppassero LessTif, e più recentemente abbiamo dato vita ai progetti GNOME e Harmony per affrontare i problemi causati da alcune librerie proprietarie (come descritto più avanti). Stiamo sviluppando la GNU Privacy Guard per sostituire i diffusi programmi di crittografia non liberi, perché gli utenti non siano costretti a scegliere tra riservatezza e libertà.

Naturalmente, i redattori di questi programmi sono coinvolti nel loro lavoro, e varie persone vi hanno aggiunto diverse funzionalità secondo le loro personali necessità ed i loro interessi. Tuttavia non è questa la ragione dell'esistenza di tali programmi.

Sviluppi inattesi

All'inizio del progetto GNU pensavo che avremmo sviluppato l'intero sistema GNU e poi lo avremmo reso disponibile tutto insieme, ma le cose non andarono così.

Poiché i componenti del sistema GNU sono stati implementati su un sistema Unix, ognuno di essi poteva girare su sistemi Unix molto prima che esistesse un sistema GNU completo. Alcuni di questi programmi divennero diffusi e gli utenti iniziarono ad estenderli e a renderli utilizzabili su nuovi sistemi: sulle varie versioni di Unix, incompatibili tra loro, e talvolta anche su altri sistemi.

Questo processo rese tali programmi molto più potenti e attirò finanziamenti e collaboratori al progetto GNU; tuttavia probabilmente ritardò di alcuni anni la realizzazione di un sistema minimo funzionante, perché il tempo degli autori GNU veniva impiegato a curare la compatibilità di questi programmi con altri sistemi e ad aggiungere nuove funzionalità ai componenti esistenti, piuttosto che a proseguire nella scrittura di nuovi componenti.

GNU-Hurd

Nel 1990 il sistema GNU era quasi completo, l'unica parte significativa ancora mancante era il kernel. Avevamo deciso di implementare il nostro kernel come un gruppo di processi server che girassero sul sistema Mach. Mach è un microkernel sviluppato alla Carnegie Mellon University e successivamente all'Università dello Utah; GNU Hurd è un gruppo di server (o "herd of gnus": mandria di gnu) che gira su Mach svolgendo le funzioni del kernel Unix. L'inizio dello sviluppo fu ritardato nell'attesa che Mach fosse reso disponibile come software libero, come era stato promesso.

Una ragione di questa scelta progettuale fu di evitare quella che sembrava la parte più complessa del lavoro: effettuare il debugging del kernel senza un debugger a livello sorgente. Questo lavoro era già stato fatto, appunto in Mach, e avevamo previsto di effettuare il debugging dei server Hurd come programmi utente, con GDB. Ma questa fase si rivelò molto lunga, ed il debugging dei server multi-thread che si scambiano messaggi si è rivelato estremamente complesso. Per rendere Hurd robusto furono così necessari molti anni.

Alix

Originariamente il kernel GNU non avrebbe dovuto chiamarsi Hurd; il suo nome originale era Alix, come la donna di cui ero innamorato in quel periodo. Alix, che era amministratrice di sistemi Unix, aveva sottolineato come il suo nome corrispondesse ad un comune schema usato per battezzare le versioni del sistema Unix: scherzosamente diceva ai suoi amici: «qualcuno dovrebbe chiamare un kernel come me». Io non dissi nulla ma decisi di farle una sorpresa scrivendo un kernel chiamato Alix.

Le cose non andarono così. Michael Bushnell (ora Thomas), principale autore del kernel, preferì il nome Hurd, e chiamò Alix una parte del kernel, quella che serviva

a intercettare le chiamate di sistema e a gestirle inviando messaggi ai server che compongono HURD.

Infine io e Alix ci lasciammo e lei cambiò nome; contemporaneamente la struttura di Hurd veniva cambiata in modo che la libreria C mandasse messaggi direttamente ai server, e così il componente Alix scomparve dal progetto. Prima che questo accadesse, però, un amico di Alix si accorse della presenza del suo nome nel codice sorgente di Hurd e glielo disse. Così il nome raggiunse il suo scopo.

Linux e GNU/Linux

GNU Hurd non è pronto per un uso non sperimentale, ma per fortuna è disponibile un altro kernel: nel 1991 Linus Torvalds sviluppò un Kernel compatibile con Unix e lo chiamò Linux. Attorno al 1992, la combinazione di Linux con il sistema GNU ancora incompleto produsse un sistema operativo libero completo (naturalmente combinarli fu un notevole lavoro di per sé). E grazie a Linux che oggi possiamo utilizzare una versione del sistema GNU.

Chiamiamo GNU/Linux questa versione del sistema, per indicare la sua composizione come una combinazione del sistema GNU col kernel Linux.

Le sfide che ci aspettano

Abbiamo dimostrato la nostra capacità di sviluppare un'ampia gamma di software libero, ma questo non significa che siamo invincibili e inarrestabili. Diverse sfide rendono incerto il futuro del software libero, e affrontarle richiederà perseveranza e sforzi costanti, talvolta per anni. Sarà necessaria quella determinazione che le persone sanno dimostrare quando danno valore alla propria libertà e non permettono a nessuno di sottrargliela. Le quattro sezioni seguenti parlano di queste sfide.

Hardware segreto

Sempre più spesso, i costruttori di hardware tendono a mantenere segrete le specifiche delle loro apparecchiature; questo rende difficile la scrittura di driver liberi che permettano a Linux e XFree86 di supportare nuove periferiche. Anche se oggi abbiamo sistemi completamente liberi, potremmo non averli domani se non saremo in grado di supportare i calcolatori di domani.

Esistono due modi per affrontare il problema. Un programmatore può ricostruire le specifiche dell'hardware usando tecniche di reverse engineering. Oppure si può scegliere hardware supportato dai programmi liberi: man mano che il nostro numero aumenta, la segretezza delle specifiche diventerà una pratica controproducente.

Il reverse engineering è difficile: avremo programmatori sufficientemente determinati da dedicarvisi? Sì, se avremo costruito una forte consapevolezza che

avere programmi liberi sia una questione di principio e che i driver non liberi non sono accettabili. E succederà che molti di noi accettino di spendere un po' di più o perdere un po' più di tempo per poter usare driver liberi? Sì, se il desiderio di libertà e la determinazione ad ottenerla saranno diffusi.

Librerie non libere

Una libreria non libera che giri su sistemi operativi liberi funziona come una trappola per i creatori di programmi liberi. Le funzionalità attraenti della libreria fungono da esca; chi usa la libreria cade nella trappola, perché il programma che crea è inutile come parte di un sistema operativo libero (a rigore, il programma potrebbe esservi incluso, ma non funzionerebbe, visto che manca la libreria). Peggio ancora, se un programma che usa la libreria proprietaria diventa diffuso, può attirare altri ignari programmatori nella trappola.

Il problema si concretizzò per la prima volta con la libreria Motif, negli anni '80. Sebbene non ci fossero ancora sistemi operativi liberi, i problemi che Motif avrebbe causato loro erano già chiari. Il progetto GNU reagì in due modi: interessandosi presso diversi progetti di software libero perché supportassero gli strumenti grafici X liberi in aggiunta a Motif, e cercando qualcuno che scrivesse un sostituto libero di Motif. Il lavoro richiese molti anni: solo nel 1997 LessTif, sviluppato dagli "Hungry Programmers", divenne abbastanza potente da supportare la maggior parte delle applicazioni Motif.

Tra il 1996 e il 1998 un'altra libreria non libera di strumenti grafici, chiamata Qt, veniva usata in una significativa raccolta di software libero: l'ambiente grafico KDE.

I sistemi liberi GNU/Linux non potevano usare KDE, perché non potevamo usare la libreria; tuttavia, alcuni distributori commerciali di sistemi GNU/Linux, non scrupolosi nell'attenersi solo ai programmi liberi, aggiunsero KDE ai loro sistemi, ottenendo così sistemi che offrivano più funzionalità, ma meno libertà. Il gruppo che sviluppava KDE incoraggiava esplicitamente altri programmatori ad usare Qt, e milioni di nuovi "utenti Linux" non sospettavano minimamente che questo potesse costituire un problema. La situazione si faceva pericolosa.

La comunità del software libero affrontò il problema in due modi: GNOME e Harmony.

GNOME (GNU Network Object Model Environment, modello di ambiente per oggetti di rete) è il progetto GNU per l'ambiente grafico (desktop). Intrapreso nel 1997 da Miguel de Icaza e sviluppato con il supporto di Red Hat Software, GNOME si ripromise di fornire funzionalità grafiche simili a quelle di KDE, ma usando esclusivamente software libero. GNOME offre anche dei vantaggi tecnici, come il supporto per svariati linguaggi di programmazione, non solo il C++. Ma il suo scopo principale era la libertà: non richiedere l'uso di alcun programma che non fosse libero.

Harmony è una libreria compatibile con Qt, progettata per rendere possibile l'uso del software KDE senza dover usare Qt.

Nel novembre 1998 gli autori di Qt annunciarono un cambiamento di licenza che, una volta operativo, avrebbe reso Qt software libero. Non c'è modo di esserne certi, ma credo che questo fu in parte dovuto alla decisa risposta della comunità al problema posto da Qt quando non era libero (la nuova licenza è scomoda ed iniqua, per cui rimane comunque preferibile evitare l'uso di Qt).

Come risponderemo alla prossima allettante libreria non libera? Riuscirà la comunità in toto a comprendere l'importanza di evitare la trappola? Oppure molti di noi preferiranno la convenienza alla libertà, creando così ancora un grave problema? Il nostro futuro dipende dalla nostra filosofia.

Brevetti sul software

Il maggior pericolo a cui ci troviamo di fronte è quello dei brevetti sul software, che possono rendere inaccessibili al software libero algoritmi e funzionalità per un tempo che può estendersi fino a vent'anni. I brevetti sugli algoritmi di compressione LZW furono depositati nel 1983, e ancor oggi non possiamo distribuire programmi liberi che producano immagini GIF compresse. Nel 1998 un programma libero per produrre audio compresso MP3 venne ritirato sotto minaccia di una causa per violazione di brevetto.

Ci sono modi per affrontare la questione brevetti: possiamo cercare prove che un brevetto non sia valido oppure possiamo cercare modi alternativi per ottenere lo stesso risultato. Ognuna di queste tecniche, però, funziona solo in certe circostanze; quando entrambe falliscono un brevetto può obbligare tutto il software libero a rinunciare a qualche funzionalità che gli utenti desiderano. Cosa dobbiamo fare quando ciò accade?

Chi fra noi apprezza il software libero per il valore della libertà rimarrà comunque dalla parte dei programmi liberi; saremo in grado di svolgere il nostro lavoro senza le funzionalità coperte da brevetto. Ma coloro che apprezzano il software libero perché si aspettano che sia tecnicamente superiore probabilmente grideranno al fallimento quando un brevetto ne impedisce lo sviluppo. Perciò, nonostante sia utile parlare dell'efficacia pratica del modello di sviluppo "a cattedrale", e dell'affidabilità e della potenza di un dato programma libero, non ci dobbiamo fermare qui; dobbiamo parlare di libertà e di principi.

Documentazione libera

La più grande carenza nei nostri sistemi operativi liberi non è nel software, quanto nella carenza di buoni manuali liberi da includere nei nostri sistemi. La documentazione è una parte essenziale di qualunque pacchetto software; quando un importante pacchetto software libero non viene accompagnato da un buon manuale libero si tratta di una grossa lacuna. E di queste lacune attualmente ne abbiamo molte.

La documentazione libera, come il software libero, è una questione di libertà, non di prezzo. Il criterio per definire libero un manuale è fondamentalmente lo stesso che per definire libero un programma: si tratta di offrire certe libertà a tutti gli utenti. Deve essere permessa la redistribuzione (compresa la vendita commerciale), sia in formato elettronico che cartaceo, in modo che il manuale possa accompagnare ogni copia del programma.

Autorizzare la modifica è anch'esso un aspetto cruciale; in generale, non credo sia essenziale permettere alle persone di modificare articoli e libri di qualsiasi tipo. Per esempio, non credo che voi o io dobbiamo sentirci in dovere di autorizzare la modifica di articoli come questo, articoli che descrivono le nostre azioni e il nostro punto di vista.

Ma c'è una ragione particolare per cui la libertà di modifica è cruciale per la documentazione dei programmi liberi. Quando qualcuno esercita il proprio diritto di modificare il programma, aumentandone o alterandone le funzionalità, se è coscienzioso modificherà anche il manuale, in modo da poter fornire una documentazione utile e accurata insieme al programma modificato. Un manuale che non permetta ai programmatori di essere coscienziosi e completare il loro lavoro non soddisfa i bisogni della nostra comunità.

Alcuni limiti sulla modificabilità non pongono alcun problema; per esempio, le richieste di conservare la nota di copyright dell'autore originale, i termini di distribuzione e la lista degli autori vanno bene. Non ci sono problemi nemmeno nel richiedere che le versioni modificate dichiarino esplicitamente di essere tali, così pure che intere sezioni non possano essere rimosse o modificate, finché queste sezioni vertono su questioni non tecniche. Restrizioni di questo tipo non creano problemi perché non impediscono al programmatore coscienzioso di adattare il manuale perché rispecchi il programma modificato. In altre parole, non impediscono alla comunità del software libero di beneficiare appieno dal manuale.

D'altro canto, deve essere possibile modificare tutto il contenuto tecnico del manuale e poter distribuire il risultato in tutti i formati usuali, attraverso tutti i normali canali di distribuzione; diversamente, le restrizioni creerebbero un ostacolo per la comunità, il manuale non sarebbe libero e avremmo bisogno di un altro manuale.

Gli sviluppatori di software libero avranno la consapevolezza e la determinazione necessarie a produrre un'intera gamma di manuali liberi? Ancora una volta, il nostro futuro dipende dalla nostra filosofia.

Dobbiamo parlare di libertà

Stime recenti valutano in dieci milioni il numero di utenti di sistemi GNU/Linux quali Debian GNU/Linux e Red Hat Linux. Il software libero ha creato tali vantaggi pratici che gli utenti stanno approdando ad esso per pure ragioni pratiche.

Gli effetti positivi di questa situazione sono evidenti: maggior interesse a sviluppare software libero, più clienti per le imprese di software libero e una

migliore capacità di incoraggiare le aziende a sviluppare software commerciale libero invece che prodotti software proprietari.

L'interesse per il software, però, sta crescendo più in fretta della coscienza della filosofia su cui è basato, e questa disparità causa problemi. La nostra capacità di fronteggiare le sfide e le minacce descritte in precedenza dipende dalla determinazione nell'essere impegnati per la libertà. Per essere sicuri che la nostra comunità abbia tale determinazione, dobbiamo diffondere l'idea presso i nuovi utenti man mano che entrano a far parte della comunità.

Ma in questo stiamo fallendo: gli sforzi per attrarre nuovi utenti nella comunità sono di gran lunga maggiori degli sforzi per l'educazione civica della comunità stessa. Dobbiamo fare entrambe le cose, e dobbiamo mantenere un equilibrio fra i due impegni.

"Open Source"

Parlare di libertà ai nuovi utenti è diventato più difficile dal 1998, quando una parte della comunità decise di smettere di usare il termine "free software" e usare al suo posto "open source".

Alcune delle persone che suggerirono questo termine intendevano evitare che si confondesse "free" con "gratis", un valido obiettivo. D'altra parte, altre persone intendevano mettere da parte lo spirito del principio che aveva dato la spinta al movimento del software libero e al progetto GNU, puntando invece ad attrarre i dirigenti e gli utenti commerciali, molti dei quali afferiscono ad una ideologia che pone il profitto al di sopra della libertà, della comunità, dei principi. Perciò la retorica di "open source" si focalizza sulla possibilità di creare software di buona qualità e potente ma evita deliberatamente le idee di libertà, comunità, principio.

Le riviste che si chiamano "Linux..." sono un chiaro esempio di ciò: sono piene di pubblicità di software proprietario che gira sotto GNU/Linux; quando ci sarà il prossimo Motif o Qt, queste riviste avvertiranno i programmatori di starne lontano o accetteranno la sua pubblicità?

L'appoggio delle aziende può contribuire alla comunità in molti modi; a parità di tutto il resto è una cosa utile. Ma ottenere questo appoggio parlando ancor meno di libertà e principi può essere disastroso; rende ancora peggiore lo sbilanciamento descritto tra diffusione ed educazione civica.

"Software libero" (free software) e "sorgente aperto" (open source) descrivono più o meno la stessa categoria di software, ma dicono cose differenti sul software e sui valori. Il progetto GNU continua ad usare il termine "software libero" per esprimere l'idea che la libertà sia importante, non solo la tecnologia.

Prova!

La filosofia di Yoda ("Non c'è provare") suona bene, ma per me non funziona. Ho fatto la maggior parte del mio lavoro angustiato dal timore di non essere in grado di svolgere il mio compito e nel dubbio, se fossi riuscito, che non fosse sufficiente per raggiungere l'obiettivo. Ma ci ho provato in ogni caso perché nessuno tranne me si poneva tra il nemico e la mia città. Sorprendendo me stesso, qualche volta sono riuscito.

A volte ho fallito, alcune delle mie città sono cadute; poi ho trovato un'altra città minacciata e mi sono preparato ad un'altra battaglia. Con l'andar del tempo ho imparato a cercare le possibili minacce e a mettermi tra loro e la mia città, facendo appello ad altri hacker perché venissero e si unissero a me.

Oggigiorno spesso non sono da solo. È un sollievo ed una gioia quando vedo un reggimento di hacker che scavano trincee per difendere il confine e quando mi rendo conto che questa città può sopravvivere; per ora. Ma i pericoli diventano più grandi ogni anno, ed ora Microsoft ha esplicitamente preso di mira la nostra comunità. Non possiamo dare per scontato il futuro della libertà; non diamolo per scontato! Se volete mantenere la vostra libertà dovete essere pronti a difenderla.

Per informazioni e domande su GNU e la FSF rivolgersi (in inglese) a gnu@gnu.org. Ci sono anche altri modi di contattare la FSF (inglese).

Copyright (C) 1998 Richard Stallman

La copia letterale e la distribuzione di questo articolo nella sua integrità sono permesse con ogni mezzo, a patto che questa nota sia riprodotta.

Ultimo aggiornamento: 12 Sep 2000 chsong Finita di tradurre mercoledì 7 luglio 1999