Command file & Argument template

Lo strano nome con cui è intitolato questo articolo non dovrebbe preoccupare troppo gli utenti di Amiga. Anche se suonano un po' strano, ci accorgeremo presto che si tratta di due "feature" dell'AmigaDOS tutt'altro che inutili che stanno solo ad aspettare di essere utilizzate da tutti, spesso e volentieri. Un command file, lo dice il ragionamento stesso, è un file contenente un certo numero di comandi AmigaDOS che è possibile eseguire tramite il comando EXECUTE. Gli Argument template, servono per conoscere «on line» come deve essere usato un determinato comando, se non abbiamo sottomano il manuale dell'AmigaDOS e ci sfugge la sua sintassi.

di Andrea de Prisco

Command File

A dire il vero, nella seconda puntata di Amighevole abbiamo già usato un Command file anche se non vi abbiamo espressamente detto che era tale. Ci stiamo riferendo alla Startup-Sequence della directory S che al momento del boot o del re-boot viene eseguita dal sistema.

Per andare automaticamente in CLI, come detto, è stato sufficiente accedere a questo command file tramite l'editor ED,togliere le due linee che caricavano il workbench e chiudevano la sessione CLI e risalvare la Startup-Sequence così modificata.

Oltre a questa, in AmigaDOS è possibile creare quanti command file vogliamo e per mandarli in esecuzione (proprio come se fosse un programmino) digitiamo il comando EXECUTE seguito dal nome del file di comandi. Tanto per assaporare subito un esempio, dato che la Startup-Sequence è già presente sul dischetto sul quale state lavorando provate a digitare:

```
EXECUTE Startup-Sequence
```

avrete l'effetto di provocare nuovamente quello che succede al momento del boot ad opera della Startup-Sequence stessa (dipende da questa).

Si noti che non è necessario specificare s/startup-sequence, in quanto il comando execute accede al device logico S: che per default è associato alla directory S. In altre parole, se digitiamo la linea:

```
EXECUTE NomeFile
```

NomeFile sarà prima cercato nella directory corrente e se non è trovato viene scandita la directory associata a S: (esse-duepunti). Come vedremo tra poco, è possibile passare parametri a un command file così come è possibile usare costrutti IF-THEN-ELSE, etichette e goto. Procediamo con ordine.

Facciamo un primo esempio

Lo scorso mese abbiamo mostrato come implementare i comandi AmigaDOS nella RAM di Amiga in modo da averli sempre tutti disponibili senza necessità di tenere il disco workbench nel drive. Bastavano in tutto tre comandi, per l'esattezza:

```
MAKEDIR RAM: Comandi
COPY C TO RAM: Comandi
ASSIGN C: RAM: Comandi
```

Questo mese, come esempio, implementeremo un command file di nome Comandi InRam contenente i tre comandi di cui sopra. All'occorrenza basterà digitare soltanto:

```
EXECUTE Comandi InRam
```

Come è prevedibile, ci avvarremo ancora una volta del comando ED scrivendo:

```
ED S/Comandi InRam
```

e, una volta «entrati» nell'editor (foto 1), digiteremo i tre comandi di cui sopra e al termine un ESC X per uscire salvando il nostro command file così creato. Il prefisso S/ serve per mettere il file nella directory S, detta appunto directory delle sequenze, come detto, per default associata al device logico S:.

Passaggio dei parametri

Nella definizione di un command file è possibile specificare una lista di parametri formali ai quali, al momento dell'esecuzione, saranno sostituiti i parametri attuali specificati di seguito al comando EXECUTE CommandFile. Per definire i parametri, è disponibile la direttiva .key (puntokey) seguita dai nomi dei parametri usati. Ad esempio, se il nostro command file prende come parametri due file, ne fa l'unione, il sort, la visualizzazione del file ordinato e al termine li cancella da disco tutti e due, lasciando il solo file unione (non ordinato), sarà scritto nel seguente modo:

```
.key file1, file2
join <file1> <file2> as FileUnione
sort FileUnione to * delete <file1> <file2>
```

Commentiamolo brevemente. La prima linea serve, come detto, per dare un nome formale ai parametri che verranno passati al command file. Nel nostro caso abbiamo scelto file1 e file2, ma avremo potuto scegliere qualsiasi altra coppia di nomi. L'importante è che all'interno del command file ciò che useremo come file 1 sarà il primo dei file passati come parametro e file2 il secondo.

Per usare, all'interno del command file, i parametri passati dall'esterno al momento della chiamata, si usano i delimitatori < e > (minore e maggiore). Ovvero: «file1» sarà il file passato come primo parametro, «file2» il secondo. La seconda linea del command file semplicemente esegue il join dei due file creandone uno nuovo di nome FileUnione. La terza linea mostra il sort sul video e la quarta cancella dal dischetto i file sorgente. Posto che il nostro command file si chiami (per usare ED è necessario dare un nome al file che stiamo creando) «Saldafile» e che i due file da unire presenti su disco si chiamino a loro volta «tizio» e

Foto 1

Foto 2

«caio», la chiamata avverrebbe semplicemente con:

```
EXECUTE SaldaFile tizio caio
```

È possibile anche definire valori di default col carattere di controllo $ oppure con la direttiva .def (puntodef). Nel primo caso, ad ogni occorrenza del file immetteremo tra i simboli < e > anche il default preceduto da $ (es.: <file1$pippo> se file1 non ha parametro attuale assume come valore pippo), nel secondo caso, per avere lo stesso effetto su tutte le occorrenze di file1 scriveremo, di seguito alla direttiva .key, la linea

```
.def file1 «pippo»
```

IF, ELSE ed etichette

Come abbiamo già preannunciato, un command file può anche contenere salti condizionati e incondizionati. I primi tramite strutturazione IF (then) ELSE, i secondi grazie ai comandi LAB e SKIP per definire e saltare ad una etichetta.

Cominciamo proprio da queste. In un punto qualsiasi di un command file è possibile introdurre una etichetta tramite il comando LAB. Ad esempio, se vogliamo etichettare un punto della nostra sequenza con un nome, ad esempio... PIPPO, scriveremo la linea (nel punto desiderato):

```
LAB PIPPO
```

per saltarvi (solo in avanti!, purtroppo) useremo:

```
SKIP PIPPO
```

La strutturazione condizionale ha invece questa forma:

```
IF condizione
comando1
.
.
comandoN
ELSE
comando1
.
.
comandoN
ENDIF
```

Il ramo ELSE è facoltativo, nel qualcaso può essere sostituito direttamente con ENDIF.

Se la condizione dà esito positivo (vero) sarà eseguita la prima lista di comandi e se è presente un else sarà saltata la seconda. Se la condizione dá esito negativo (falso) sarà eseguito il ramo else e saltata la lista dei comandi tra IF e ELSE. Se il ramo else non è presente, in caso di esito negativo si saltano tutti i comandi fino all'ENDIF.

Ovviamente nelle due liste comandi è possibile anche mettere uno skip, in modo da implementare il salto condizionato ad etichette. Prima di fare un esempio (credo che sia proprio necessario, ndr) c'è da elencare le possibili condizioni che possono seguire il comando IF. Se l'oggetto passato è un file o il nome di una directory, e il parametro formale si chiamava ad esempio file1, possiamo chiederci se questo esiste sul disco, eventualmente per segnalare un errore o per crearlo. Scriveremo:

```
IF EXISTS <file1>
comando1
.
.
comandoN
ELSE
ECHO «errore: file inesistente»
ENDIF
```

Se vogliamo controllare l'eguaglianza di due parametri o di un parametro e una stringa scriveremo:

```
IF <parametro1> EQ <parametro2>
```

oppure

```
IF «<parametro>» EQ «stringa»
```

non senza ricordarvi che in questo modo è possibile controllare se un parametro è stato passato o meno:

```
IF «<parametro>» EQ ««
```

Oltre a ciò possiamo controllare l'esito di un comando, facendolo seguire da un IF con condizione WARN, ERROR o FAIL. Nel primo caso la condizione è soddisfatta per return-code >=5, nel secondo se >=10, nel terzo se >=20.

Infine, possiamo negare il risultato di una qualsiasi condizione logica facendola precedere da NOT, ad esempio:

```
IF NOT EXISTS <file1>
IF NOT «<parametro>» EQ ««
IF NOT ERROR
IF NOT FAIL
```

ecc. ecc.

Final Example

Per concludere la nostra rassegna sulle istruzioni che possono comporre un command file, mostreremo un esempio abbastanza completo e utile per chi lavora spesso e volentieri in CLI. Molti si saranno chiesti come creare un dischetto boot-abile, ovvero un dischetto che, introdotto dopo un reset o dopo il Kickstart, mandi automaticamente in esecuzione il programma in esso contenuto. Per farlo sono necessarie, a dire il vero, un bel po' di operazioni e proprio in casi come questo che conviene preparare un command file apposito. Vediamo prima cosa si dovrebbe fare da tastiera. Una volta in possesso di un dischetto contenente un programma ma non il necessario per farlo partire in autostart, le operazioni sono:

1) installare il disco

Foto 3

Foto 4

2) inserire su questo le librerie necessarie

3) preparare una apposita Startup-Sequence

Per installare un dischetto, si usa il comando INSTALL seguito dal drive nel quale è inserito il dischetto. Si noti che, siccome il comando install è un programmino contenuto sul disco sistema, se non disponiamo del drive esterno, digitare INSTALL DFO: equivale a re-installare il disco contenuto in tale drive che sicuramente è anche già boot-abile. Con un solo drive, l'unica soluzione è di creare un Ram disk con tutti i comandi (o almeno il solo install) e riprovare.

Le librerie necessarie al boot (e al funzionamento corretto anche dopo il boot) sono contenute nelle directory L e LIBS. La seconda fase corrisponde a copiare sul nuovo dischetto tali due directory ovviamente non senza averle prima create. Quindi, sempre da tastiera (non stiamo ancora facendo l'ipotesi di uso di un command file) digiteremo, se disponiamo di due drive:

```
MAKEDIR DF1:L
MAKEDIR DF1:LIBS
COPY L TO DF1:L ALL
COPY LIBS TO DF1:LIBS ALL
```

A questo punto il dischetto è già in grado di funzionare correttamente anche se ancora non manda in esecuzione il programma in esso contenuto. Quest'ultima fase consiste semplicemente nel creare uno startup-sequence contenente il nome del programma da caricare. Per prima cosa costruiremo la directory S

```
MAKEDIR DF1:S
```

Dopo di ciò, da tastiera useremo il comando ED oppure un copy da schermo (ne abbiamo parlato lo scorso mese, vi ricordate?) nella forma:

```
COPY * TO DF1:S/Startup-Sequence
```

In questo secondo caso, dopo aver digitato le istruzioni per mandare in esecuzione il programma (in genere basta solo digitare il nome) per salvare la Startup-Sequence basta un control \ (left slash, accanto al backspace). Giusto per essere sicuri di esserci spiegati bene, se il programma contenuto sul nostro dischetto si chiama PIPPO dopo il copy di cui sopra digiteremo Pippo, un [return], e il fatidico control \. Fatto.

In figura 1 (pag. 112) è mostrato il command file in grado di eseguire tutto questo automaticamente. Se ad esempio tale command file si chiama InstallDrive, per eseguirlo basterà semplicemente digitare:

```
EXECUTE InstallDrive DF1:
```

se il disco da installare è posto nel drive esterno, altrimenti l'indicazione di drive sarebbe stata diversa.

Commentiamolo brevemente. La prima linea specifica il nome del parametro che useremo: drive (ripetiamo, il nome è assolutamente fittizio). Di seguito a questo, il primo IF controlla se è stato omesso il parametro, nel qualcaso viene mandato su video un messaggio di help e con «SKIP fine», si salta a «LAB fine» dove è presente un QUIT che fa terminare il command file. Se, di contro, «tutto è a posto» installiamo il <drive> procediamo alla creazione delle directory L, LIBS e S se queste non sono già presenti sul dischetto (come è facile supporre). Infine i comandi di copy per L, LIBS e startup-sequence, quest'ultimo da video come visto prima.

Giusto per provare che quanto abbiamo detto non sono poi troppo «frottole», formattiamo un dischetto, mettiamo su questo un qualsiasi programma in nostro possesso, naturalmente, eseguibile da cli e facciamo partire il nostro command file, ad esempio col disco appena creato nel drive esterno e il disco di sistema in quello interno:

```
EXECUTE InstallDrive DF1:
```

e il gioco è fatto.

Con un solo drive

Per adoperare il command file InstallDrive appena mostrato disponendo di un solo drive (quello interno) occorre eseguire un po' di operazioni per creare un RAM disk opportuno. In figura 2 (pag. 112) è mostrato un command file che provvede a tale scopo ovvero ricopia su RAM tutto quello che ci servirà una volta inserito il dischetto non ancora installato nel drive interno. Si noti come anche il programma InstallDrive viene caricato in RAM in quanto occorre invocare questo appena terminata tale inizializzazione. Per ipotesi si suppone che InstallDrive sia stato inserito, come di consueto, nella directory S delle sequenze. Detto questo (e soprattutto: digitati i due command file) per installare un dischetto e renderlo boot-abile senza disporre di un drive esterno, porremo nel nostro drive il disco contenente le due sequenze, digiteremo:

```
EXECUTE InstallRam
```

(posto che così avevamo deciso di chiamare il file di figura 2), al termine introdurremo il disco da installare e digiteremo (come prevedibile):

```
EXECUTE InstallDrive DFO:
```

tutto qui.

Argument Template

Come detto in apertura, gli Argument Template sono di aiuto quando non ricordiamo la sintassi di un comando AmigaDOS. Per farli saltare

Figura 1

```
.key drive
if "<drive>" eq ""
echo "USAGE: execute InstallDrive DFn:"
skip fine
endif
install <drive>
if not exists <drive>l
makedir <drive>l
endif
if not exists <drive>libs
makedir <drive>libs
endif
if not exists <drive>s
makedir <drive>s
endif
copy sys:l all to <drive>l
copy sys:libs all to <drive>libs
echo "enter startup-sequence...
     terminate with CTRL \"
copy * to <drive>s/startup-sequence
lab fine
quit
```

Figura 2

```
makedir ram:comandi
copy c to ram:comandi all
assign c: ram:comandi
makedir ram:t
makedir ram:l
makedir ram:libs
copy l to ram:l all
copy libs to ram:libs all
copy s/installdrive to ram:
assign sys: ram:
cd sys:
echo "insert disk to install... "
echo "...and type EXECUTE InstallDrive DF0:"
```

fuori, è sufficiente digitare il comando seguito dal punto interrogativo. Proviamo ad esempi a digitare DIR ? (foto 3). Il sistema ci risponderà:

DIR, OPT/K:

mostrando il cursore lampeggiante di seguito ai due punti.

Se invece di DIR avessimo chiesto il template di RENAME, avremo ottenuto:

FROM/A, TO=AS/A

Cerchiamo di capire il significato. Quanto mostrato rappresenta le parole chiave di un determinato comando. Una parola chiave, a seconda da cosa è seguita, può essere facoltativa, obbligatoria, richiedere un argomento o semplicemente un flag per specificare una particolare opzione di quel comando. Nei due esempi mostrati, dir (da non confondere col comando), opt, from, to, as sono parole chiave. Il fatto che sia indicato to=as vuol dire che le due parole sono assolutamente equivalenti. /A che segue alcune chiavi vuol dire che associate a queste *deve* essere dato un argomento, mentre /K indica che la chiave deve essere indicata se intendiamo associare a questa un argomento. L'assenza di ulteriori specifiche di chiave (come dir nel comando DIR) indica che l'argomento è facoltativo e può essere dato senza chiave. Prendiamo RENAME e vediamo cosa ci dice il suo template. Innanzitutto è necessario specificare due argomenti (il nome del file esistente e il nuovo nome da dare a questo). Le parole chiave sono facoltative se intendiamo riferirci all'ordine degli argomenti. Ad esempio possiamo scrivere:

RENAME tizio caio

che eseguirà un rename del file tizio dandogli come nuovo nome caio. Oppure possiamo indicare le chiavi:

RENAME FROM tizio TO caio
RENAME FROM tizio AS caio

e possiamo anche capovolgere gli argomenti, se leghiamo ad ognuno di essi la corrispondente chiave (in modo che Amiga non si sbagli):

RENAME TO caio FROM tizio
RENAME AS caio FROM tizio

Torniamo al nostro DIR. Il template era DIR, OPT/K: vuol dire che possiamo specificare una directory cui fare riferimento (la parola chiave è anch'essa facoltativa) e se intendiamo usare una opzione *dobbiamo* specificare la chiave OPT. Da questo le possibili forme del comando DIR:

DIR
DIR DF1:
DIR C
DIR DFO:FONT/ELITE
DIR OPT I
DIR OPT A
DIR C OPT I
DIR dir C OPT A

eccetera eccetera (come da template).

Si noti che una volta chiesto il template, possiamo rispondere alla richiesta del sistema che ci indica cosa manca al comando che abbiamo digitato. Ovvero, una volta digitato DIR ? di seguito al template (dove ci aspetta il cursore lampeggiante) possiamo dire cosa vogliamo, ad esempio (foto 4) OPT I oppure C oppure DF1: o ciò che ci pare.

Per concludere, vi mostreremo uno dei template più complicati dell'AmigaDOS: quello del comando LIST

DIR,P=PAT/K,KEYS/S,DATES/S,NODATES/S,TO/K,S/K,SINCE/K,UPTO/K,QUICK/S

Procediamo con ordine. Innanzitutto possiamo indicare (come per DIR) la directory che intendiamo listare. Oltre a questo abbiamo 4 opzioni di tipo flag (specifica /S) che possiamo adoperare per scegliere determinate opzioni di list. La prima KEYS serve per visualizzare oltre a tutto il resto anche il blocco in cui ha inizio il file in questione. NODATES serve per eliminare la visualizzazione della data nei file mentre QUICK si usa per avere un list veloce dei file (nessuna indicazione oltre al nome). Infine, l'opzione DATES, in unione a QUICK permette di mostrare solo nome e data di ogni file (complementare dunque a NODATES).

Esempi di richieste di list possono essere:

LIST NODATES
LIST QUICK
LIST QUICK DATES
LIST KEYS
LIST KEYS QUICK DATES

Come chiavi obbligatorie se usate con argomento (specifica /K) troviamo P=PAT, TO, S, SINCE e UPTO. La prima consente l'uso di pattern per selezionare determinati elementi della lista. Di questo però ce ne occuperemo un'altra volta: l'argomento è lungo. TO serve per specificare un file o un device dove spedire la lista:

LIST TO PRT:

esegue il list su carta

LIST TO NomeFile

crea un file contenente la lista. La chiave S permette di indicare una sottostringa di nome file in modo da selezionare solo quegli elementi che la contengono. Ad esempio potremmo digitare:

LIST S .info

per ottenere solo i file che contengono nel nome la dicitura .info. Infine, con SINCE e UPTO è possibile specificare una data per ottenere rispettivamente i file creati prima o dopo della data indicata:

LIST SINCE YESTERDAY
LIST SINCE 25-12-86
LIST UPTO TODAY
LIST UPTO 30-11-85

Dulcis in fundo, sono possibili richieste quanto complicato vogliamo, nate da combinazioni di tutte le possibilità viste sopra. Ad esempio:

LIST KEYS QUICK NODATES S .info UPTO TOMORROW TO PRT:

arrivederci.

MC

EDP USA 1987

MOSTRA E SEMINARI

MILANO 3-7 MARZO 1987

FIERA DI MILANO - Padiglione 42 - Porta Meccanica

PRODOTTI: Grafica, mini, micro, personal computers, stampanti, plotter, componenti, software, telematica, data communication, reti locali e geografiche, intelligenza artificiale.

ARGOMENTI DEI SEMINARI:
Image processing - CAD/CAM nell'industria automobilistica - Intelligenza artificiale
Reti locali e geografiche - Layout di circuiti stampati inhouse - Video conferenze.

ORARIO MOSTRA: 3-6 marzo dalle 9 alle 18 • 7 marzo dalle 9 alle 13
La Mostra è riservata agli operatori del settore.
Ingresso studenti pomeriggio 3 marzo previa prenotazione: L. 5.000.

Per informazioni rivolgersi a:

CENTRO COMMERCIALE AMERICANO
VIA GATTAMELATA, 5 - 20149 MILANO
TEL. (02) 46.96.451 - TELEX 330208 USIMC-I