UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Dipartimento di Ingegneria "Enzo Ferrari"

Corso di Laurea in Ingegneria Informatica

SHELL DEI SISTEMI UNIX

Tutor:

Chiar.mo Prof. Letizia Leonardi

Elaborato di Laurea di:

Pallante Laura

Anno Accademico 2021-2022

Sommario

Introduzione 3
1 Definizione e caratteristiche di una shell 4
2 Storia e sviluppo 7
3 Bourne shell 9
3.1 Funzionamento generale 9
3.2 Programmazione Script 11
4 Bourne Again Shell 13
4.1 Features 13
5 C shell 17
5.1 Sintassi 17
5.2 Features 18
5.3 Diffusione 18
6 Korn shell 20
6.1 Features 20
7 POSIX 22
7.1 Shell POSIX 22
8 Confronto tra shell 24
8.1 Uso interattivo 24
8.2 Script per uso personale 25
8.3 Script per uso pubblico 26
Conclusione 27
Appendice – Storia dello standard POSIX 28
Bibliografia e sitografia 32

Introduzione

Per un utente poco esperto, la scelta della shell da utilizzare può risultare difficile. Con questo elaborato si cercherà di fornire una panoramica sulle shell più comunemente usate, mostrandone le caratteristiche, i pregi e le peculiarità, per dare supporto nella scelta della shell da usare nel proprio ambiente UNIX/Linux.

L'elaborato sarà articolato in otto capitoli.

Nel primo capitolo verrà trattato in generale del concetto di shell e delle sue possibili applicazioni nel sistema operativo. Nel secondo ci si concentrerà invece più sull'aspetto storico e sulla progressiva diffusione ed evoluzione delle diverse shell nel tempo.

Nei capitoli successivi si tratterà nello specifico di alcune delle shell più diffuse e utilizzate, come la Bourne shell (capitolo 3), la Bourne Again shell (capitolo 4), la C shell (capitolo 5), la Korn shell (capitolo 6) e la POSIX shell (capitolo 7). Verranno evidenziate le caratteristiche peculiari di ognuna e le motivazioni storiche dietro le scelte di progetto fatte.

Nell'ottavo e ultimo capitolo si confronteranno le diverse shell di cui si è trattato nei capitoli precedenti. Verranno considerati i possibili utilizzi delle shell analizzando, a seconda del caso, quale shell sia meglio utilizzare.

1 Definizione e caratteristiche di una shell

Con il termine shell si intende un programma che ha la funzione di esporre i servizi offerti dal sistema operativo all’utente o ad altri programmi. In tal senso la shell è lo strato più esterno del sistema operativo, da cui il suo nome che si può tradurre con “guscio” o “involucro”.

In generale, le shell non interagiscono direttamente con il kernel del sistema operativo. Pur essendo applicazioni speciali, utilizzano le kernel API allo stesso modo di tutte le altre applicazioni. Una shell gestisce l’interazione tra utente e sistema operativo attraverso tre passaggi: attende l’input dell’utente, lo interpreta e infine mostra all’utente l’output generato dal sistema operativo.

Una shell può essere progettata per accettare input di varia natura. Principalmente si suddividono in shell linea di comando (CLI, Command-Line Interface, **Figura 1.1**) e shell con interfaccia grafica (GUI, Graphical User Interface, **Figura 1.2**), ma ce ne sono anche altre meno comuni che si possono utilizzare in ambiti più specifici, come ad esempio le shell ad interpretazione vocale.

Le shell di tipo GUI sono generalmente più semplici e intuitive da utilizzare, soprattutto per un utente poco pratico, ma offrono meno funzionalità. Per questo motivo i sistemi operativi che presentano una shell GUI permettono l’accesso anche a una shell CLI attraverso cui l’utente esperto può accedere a tutti i servizi.

In passato nei sistemi Unix-like, come ad esempio Linux, l’unica interfaccia utente disponibile era una shell di tipo CLI. Con il passare del tempo è stata aggiunta anche un’interfaccia di tipo GUI a tali sistemi con l’obiettivo di facilitarne l’utilizzo. Tale GUI è rimasta come caratteristica marginale del sistema, una interfaccia presente ma che non consente l’accesso a tutte le funzionalità effettivamente disponibili.

In questo elaborato si tratteranno aspetti riguardanti le shell CLI e in generale qualunque riferimento a una shell è da intendersi relativo a tale tipologia.

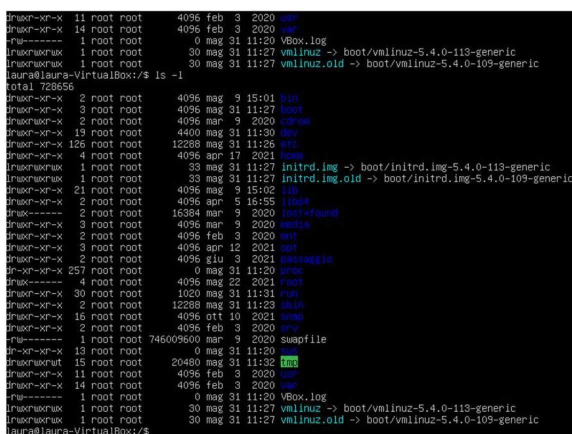

Figura 1.1: Shell di tipo CLI

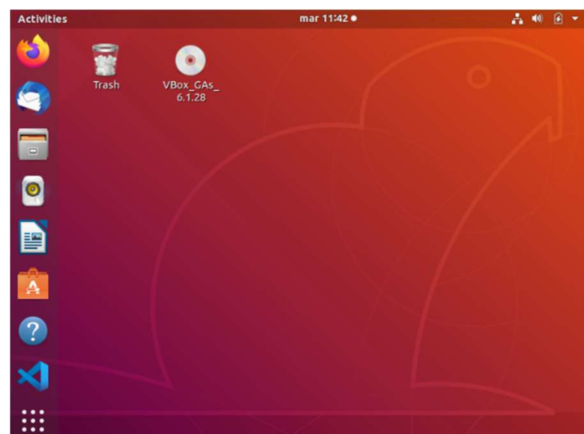

Figura 1.2: Shell di tipo GUI

Nei prossimi capitoli si tratteranno nello specifico le principali shell CLI sviluppate, tra cui la Bourne Shell, la Bourne Again Shell, la C-shell, la Korn shell ed altre. Ognuna di queste shell presenta caratteristiche che la distinguono dalle altre, ma tutte offrono alcune funzionalità di base:

- Interpretazione da linea di comando
- Parole riservate
- Meta-caratteri (wild cards)
- Gestione e accesso ai comandi
- Gestione dei file: ridirezione di input/output e pipe
- Mantenimento di variabili
- Controllo dell'ambiente
- Programmazione Shell

INTERPRETAZIONE DA LINEA DI COMANDO:

Parte interattiva della shell che riconosce e interpreta quanto scritto dall'utente.

PAROLE RISERVATE:

Ogni shell ha una lista di parole che hanno un significato speciale

META-CARATTERI (WILD CARDS):

Sono caratteri speciali che vengono sostituiti automaticamente dalla shell al momento dell'esecuzione di un comando. Permettono principalmente di eseguire operazioni su molteplici file senza specificarli uno per uno.

GESTIONE E ACCESSO AI COMANDI:

Quando viene digitato un comando, la shell ricerca attraverso una lista di directory il file relativo a tale comando e poi crea un processo figlio per eseguire il file trovato.

GESTIONE DEI FILE: RIDIREZIONE DI INPUT/OUTPUT E PIPE:

La maggior parte dei comandi UNIX riceve l'input dalla tastiera del terminale e manda l'output sul monitor del terminale. Esistono caratteri speciali per modificare tale comportamento, reindirizzando l'input e l'output verso altri file o comandi (pipe).

MANTENIMENTO DI VARIABILI:

La shell è in grado di mantenere variabili in cui memorizzare dati per un uso futuro.

CONTROLLO DELL'AMBIENTE:

Quando la shell viene avviata crea un ambiente di lavoro specifico per l'utente sulla base di alcuni file di configurazione. Tali file possono essere modificati dall'utente sulla base delle sue preferenze.

PROGRAMMAZIONE SHELL:

La shell è anche un linguaggio di programmazione. Si possono infatti creare file contenenti una serie di comandi shell combinati con assegnamenti di variabili e strutture per il controllo di flusso. Tali file vengono chiamati "script".

.

2 Storia e sviluppo

La prima shell per i sistemi Unix risale al 1971. Negli anni precedenti, tra il 1950 e il 1970, veniva utilizzato un semplice interprete da linea di comando, integrato nel resident monitor. Il resident monitor era un system software program, precursore dei sistemi operativi. Il termine "resident" si riferisce al fatto che questo programma era sempre presente nella memoria del computer. In generale il suo compito era quello di gestire il computer prima e dopo che ogni scheda perforata (**Figura 2.1**) veniva inserita ed eseguita.

Questo primo interprete, conosciuto come COMCON o DEC TOPS-10, era in grado di svolgere un numero limitato di operazioni, tra cui quello di eseguire un programma dell'utente, gestire le pratiche di log in e out, manipolare dispositivi e file e fornire informazioni varie relative a processi utente o di sistema.

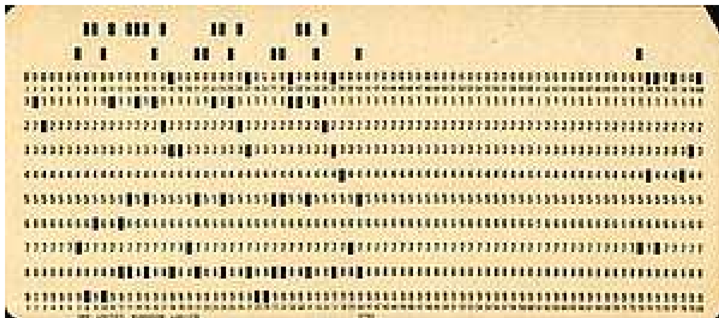

Figura 2.1: Scheda perforata

Il concetto di shell è stato per la prima volta proposto nel 1964 da Louis Pouzin. Come membro del personale del centro informatico al MIT Pouzin partecipò alla creazione del Compatible Time-Sharing System (CTSS), il primo sistema operativo a supportare operazioni in time sharing e il batch processing contemporaneamente. Pouzin scrisse dozzine di comandi CTSS da zero, realizzando in seguito che sarebbe stato vantaggioso costruire un programma che aiutasse ad automatizzare la creazione di tali comandi. Realizzò quindi RUNCOM, un programma che consentiva l'esecuzione di comandi contenuti all'interno di una directory e che può essere considerato l'antenato delle shell CLI e degli shell script.

La prima shell venne sviluppata da Ken Thompson nel 1971 con il nome di Thompson shell. Fu sviluppata nei laboratori AT&T Bell e venne rilasciata nella prima versione di Unix. In questa prima shell venne introdotto, ad esempio, il concetto di ridirezione tramite `<` e `>`, tuttora utilizzato. Era una semplice shell CLI, creata intenzionalmente con un design minimalista e non pensata per lo scripting.

In seguito, John Mashey iniziò a modificare la Thompson Shell per renderla più appropriata per la programmazione. Il risultato fu la PWB shell, detta anche Mashey Shell, distribuita tra il 1975 e il 1977. Questa includeva meccanismi di controllo di flusso più avanzati e introduceva le variabili di shell, rimanendo compatibile con la shell di Thompson. Alla fine, la Thompson shell venne rimpiazzata come shell principale per i sistemi UNIX dalla Bourne

Shell (`sh`) nella versione Unix 7 e dalla C shell (`csh`) nei sistemi 2BSD (Berkeley Software Distribution, sistemi basati sulle prime versioni di Unix, oggi fuori produzione), entrambe rilasciate nel 1979.

La Bourne shell portò allo sviluppo di numerose altre shell (**Figura 2.2**), come ad esempio la Korn shell (`ksh`), la Almquist shell (`ash`) e la diffusissima Bourne Again Shell (`bash`).

Nel 1992 viene rilasciata dalla IEEE Computer society la shell POSIX (Portable Operating System Interface), una shell CLI che si basa sullo standard definito da POSIX-IEEE P1003.2.

Sono successivamente state sviluppate altre shell, come ad esempio la Z shell (`zsh`), la Scheme shell (`scsh`) e la Pyshell, ognuna ideata come specializzazioni applicative. Nonostante ciò, le shell derivate direttamente dalla Bourne sono tutt'oggi le più universalmente utilizzate.

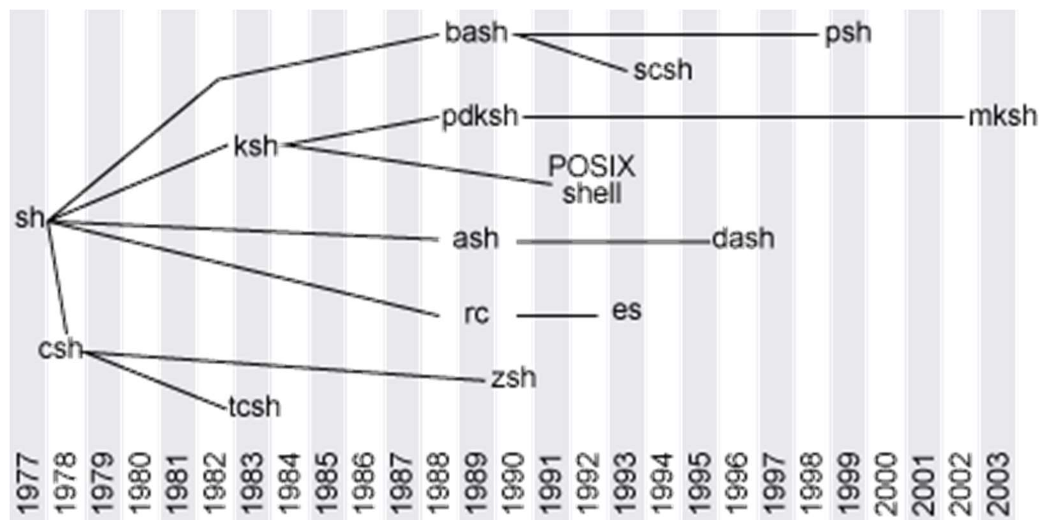

Figura 2.2: Shell Linux dal 1977

3 Bourne shell

Lo sviluppo della Bourne Shell è iniziato nel 1976 con il lavoro di Stephen Bourne ai Laboratori Bell. Nel 1979 è apparsa per la prima volta nella Versione 7 di Unix.

In questa shell sono state implementate funzionalità che non erano presenti nella shell di Thompson, che veniva precedentemente utilizzata. Nella Bourne Shell è infatti possibile utilizzare script di comandi come filtri, operazione prima non possibile in quanto l'input standard per eseguire uno script era lo script stesso. Sono state aggiunte anche la possibilità di utilizzare variabili, la sostituzione di comandi, e il comando `goto` è stato eliminato in favore di primitive di controllo di flusso come `if` e `for`.

La Bourne Shell è stata la prima a implementare la maggior parte delle features comuni anche alle shell sviluppate successivamente. Nei capitoli futuri, riguardanti le altre tipologie di shell, non verranno ripetuti i concetti già esposti in questo capitolo, ma si sottolineeranno le differenze e le peculiarità proprie di ogni CLI a confronto con quelle della Bourne Shell.

3.1 Funzionamento generale

Il comando per invocare la Bourne Shell è `sh` e il file associato a questa si trova in `/bin/sh`.

In uno scenario standard la prima invocazione della shell Bourne avviene quando un utente esegue il login su un sistema UNIX. In particolare, il file che corrisponde alla shell da eseguire è specificato nella riga del file `/etc/passwd` relativa a quell'utente.

Appena eseguita la shell legge il file `/etc/profile` per inizializzare alcune variabili di ambiente fondamentali come `PATH` e `TERM`. Poi la shell colloca l'utente nella sua home directory associata e legge il file `.profile`, dove si trovano personalizzazioni della shell relative solo a quell'utente. Infine, la shell mostra il prompt `$` o `#` ed attende un comando.

Il prompt di default per il super-utente è il simbolo `#`, mentre per tutti gli altri utenti è il simbolo del dollaro `$`.

La shell Bourne può essere invocata con varie opzioni (**Tabella 3.1**), che possono risultare utili in varie circostanze come, ad esempio, per fare il debug di uno script.

`-e`	Modalità non-interattiva
`-i`	Modalità interattiva
`-r`	Modalità ristretta
`-n`	Legge i comandi ma non li esegue
`-x`	Mostra i comandi quando li esegue
`-t`	Esegue un solo comando e poi si chiude

Tabella 3.1: Esempi di opzioni per l'esecuzione della shell

Nella modalità interattiva lo standard input e output della shell sono collegati a un terminale, mentre la modalità non interattiva viene utilizzata per l'esecuzione di script.

L'utente può selezionare un'opzione per la shell anche mentre questa è già in esecuzione tramite il comando `set`.

Durante la normale operatività di una shell vi sono alcune variabili che hanno sempre un valore (**Tabella 3.2**). Queste sono le variabili di ambiente che vengono inizializzate all'avvio.

`$CDPATH`	Percorsi validi per la ricerca tramite comando `cd`
`$HOME`	Directory home dell'utente
`$IFS`	Caratteri che possono essere usati come Internal Field Separators, di norma spazio, tab e a capo
`$MAIL`	Percorso per un file specifico (casella di posta) usato dal sistema di e-mail UNIX
`$PATH`	Percorsi per la ricerca di comandi ed eseguibili
`$PS1`	Stringa di prompt primaria, di norma $
`$PS2`	Stringa di prompt secondaria, di norma >
`$TERM`	Tipo di terminale in uso

Tabella 3.2: Variabili di ambiente di un sistema UNIX tipico

Alcuni caratteri non-alfanumerici vengono riservati per funzionalità di shell specifiche. Si possono generalmente dividere in cinque categorie:

- Gestione delle variabili
  - Per usare il valore di una variabile in un comando si inserisce il nome della variabile preceduto dal simbolo `$`. Per assegnare un valore ad una variabile si usa il carattere `=` tra il nome della variabile e il valore da assegnargli.
- Nomi di variabili di Shell
  - Variabili come `$#`, `$*` o `$?` contengono informazioni sulla shell in esecuzione e vengono generate e aggiornate automaticamente dalla shell e non dall'utente
- Generazione di nomi di file
  - I caratteri `*`, `?` e `[-,!]` anche chiamati wildcard vengono usati per fare il pattern matching con nomi di file nel sistema
- Controllo di dati e programmi
  - Caratteri come `>`, `<`, `:`, `|` vengono utilizzati per il controllo dell'esecuzione e dei flussi di dati
- Apici ed escape
  - Gli apici singoli o doppi vengono usati per incapsulare in una stringa caratteri speciali o separati da IFS. Il backslash è usato per incapsulare un carattere riservato, oppure per inserire un carattere speciale come tab o a capo

3.2 Programmazione Script

Molte delle innovazioni della Bourne shell rispetto alla shell di Thompson hanno come scopo quello di facilitare la creazione di programmi. La Bourne shell è infatti comunemente definita come un interprete di comandi interattivo e un linguaggio di programmazione di comandi. Il tipo di programma per cui la Bourne shell è più adatta viene comunemente chiamato script, un piccolo programma generalmente contenuto in un solo file che ha lo scopo di automatizzare l'esecuzione di compiti che verrebbero altrimenti eseguiti individualmente da un operatore umano.

In quanto linguaggio di programmazione la Bourne shell non utilizza un compilatore, bensì fa uso di un interprete che esegue riga per riga il codice contenuto in un file sorgente.

Variabili definite da utente

Lo strumento fondamentale per la programmazione in shell sono le variabili definite e gestite dall'utente. Le variabili di shell sono salvate in memoria come sequenze di caratteri, il concetto di tipo di dato non è implementato, per cui anche il type-checking è assente.

L'algebra tra variabili viene generalmente svolta tramite il comando `expr`, ma questo supporta solo operazioni tra numeri interi. Il comando `expr` offre anche qualche funzionalità di manipolazione di stringhe. Le operazioni in virgola mobile non sono invece supportate nativamente nella Bourne shell.

Passaggio di argomenti

Quando viene eseguito un comando o un programma shell stringhe successive all'invocazione vengono interpretate come parametri da passare al programma. Uno script ha accesso ai parametri con cui è stato invocato tramite variabili speciali come `$*` e `$#`.

Controllo di flusso

La Bourne shell mette a disposizione del programmatore vari costrutti per il controllo del flusso. Sono presenti costrutti condizionali come l'`if-then-else` ed il `case`, e costrutti iterativi come il `while`, l'`until` ed il `for`.

Oltre a questi costrutti elementari sono presenti opzioni di sintassi più specializzate come l'unione di due comandi tramite `&&` e `||` per cui il secondo comando viene eseguito solo se il primo ha successo o fallisce.

Infine, ci sono le pipe che permettono l'esecuzione contemporanea di più comandi tramite l'utilizzo di una sintassi molto maneggevole.

Controllo degli errori

I comandi e gli script nell'ambiente shell, oltre il loro output standard, terminano tutti con un codice di uscita, indicato come un numero intero. In generale un codice di uscita 0 indica un'esecuzione corretta senza errori, mentre codici di uscita maggiori indicano che l'operazione non è andata a buon fine in qualche modo.

Per risalire al codice di uscita dell'ultima operazione eseguita viene utilizzata la variabile `$?`. Il meccanismo dei codici di uscita è fondamentale in tutti gli ambiti della programmazione shell, ma è particolarmente utile per scrivere script in grado di gestire eventuali operazioni che non vanno a buon fine.

Lettura interattiva dei dati

Oltre il passaggio di dati tramite argomento la Bourne shell permette agli script di richiedere in maniera interattiva dati all'utente. Per questo si usa il comando `read` che mette in pausa l'esecuzione dello script in attesa che l'utente inserisca un a capo. I caratteri inseriti prima dell'a capo vengono memorizzati nelle variabili passate come argomenti a `read`, interpretati come stringhe separate da IFS. Se ci sono più stringhe che variabili quelle in eccesso vengono tutte memorizzate nell'ultima variabile.

Funzioni

È possibile anche definire funzioni in shell. Una volta definita una funzione si comporta in maniera molto simile a un comando, ma a differenza di un comando, che viene eseguito in una istanza di shell separata, una funzione viene eseguita nella stessa shell in cui è stata chiamata. Grazie a ciò una funzione è in grado di modificare i valori delle variabili all'interno dell'ambiente da cui è stata chiamata.

4 Bourne Again Shell

La Bourne Again shell (`bash`) è stata sviluppata nel 1989 da Brian Fox e Chet Ramey della Free Software Foundation ed è la shell del progetto GNU (GNU's Not Unix!). Infatti, tutt'oggi è utilizzata come shell di default sui sistemi operativi GNU. Anche se Bash è cronologicamente successiva sia alla Korn shell che alla C shell è utile esaminarla per prima, poiché è la più universalmente diffusa e documentata.

Superficialmente Bash non presenta molte differenze dalla Bourne shell, ma include nuove funzionalità introdotte inizialmente nella C shell e nella Korn shell. Per questo motivo la maggior parte degli script `sh` possono essere eseguiti in `bash` senza modifiche.

Questa shell è caratterizzata dalla portabilità, infatti può essere compilata sulla maggior parte dei sistemi UNIX, poiché molte delle variabili dipendenti dall'ambiente vengono determinate a tempo di compilazione. Bash è stata anche importata come shell per molte piattaforme non-Unix, come ad esempio Windows 95. Bash non implementa di default lo standard POSIX, ma può comunque essere configurata per essere POSIX compliant eseguendo il comando `set -o posix`.

Bash è stata concepita in origine come una versione open-source della Bourne shell e viene distribuita sotto la licenza GNU General Public License gratuitamente. Può essere utilizzata e ridistribuita (sotto determinate condizioni) da chiunque ma non è di dominio pubblico. Non ha una garanzia e quindi l'autore e il manutentore non devono essere considerati come fonte di supporto tecnico. Bash è disponibile tramite ftp (file transfer protocol) da ogni sito di archivio di GNU.

4.1 Features

Il comando per invocare la Bourne Again Shell è `bash` e il file associato si trova in `/bin/bash`.

Bash viene considerata una shell ottima per l'utilizzo sia da parte di nuovi utenti, sia da parte di utenti più esperti. Questo grazie alla sua natura general-purpose, ampie opzioni di personalizzazione e universalità. Di seguito una lista di alcune delle funzionalità presenti:

Editing della linea di comando: strumento indispensabile per correggere errori nei comandi digitati. L'utente può muovere il cursore in qualunque posizione della linea di comando e modificare il testo in quella posizione.

Completamento automatico: bash è in grado di completare automaticamente nomi di variabili, nomi utente, nomi di host, comandi e nomi di file. Per le parole che iniziano con il simbolo `$` viene tentato il completamento di variabile, per quelle che iniziano con il simbolo

`~` viene tentato il completamento di nomi utente, per quelle che iniziano con `@` viene tentato il completamento con nomi di host. Per altre parole si tenta per primo il completamento di comando e successivamente il completamento con nomi di file. Il comando generale di completamento viene dato ogni volta che viene premuto il tasto `tab`. In **Tabella 4.1** sono riportati alcuni dei comandi di completamento aggiuntivi.

`M-?`	Lista di tutti i possibili completamenti
`M-/`	Forza il tentativo di completamento con nomi di file
`M-~`	Forza il tentativo di completamento con nomi utente
`M-$`	Forza il tentativo di completamento con nomi di variabili
`M-@`	Forza il tentativo di completamento con nomi di host

Tabella 4.4.1: Comandi di completamento

`M-` si riferisce al tasto "meta", che non è più presente nella maggioranza delle tastiere moderne. Questo tasto viene solitamente emulato negli ambienti Linux dal tasto `AltGr`.

History: Con il termine "history" si intende la lista dei comandi che sono stati precedentemente eseguiti. Di default Bash conserva fino a 500 comandi nella propria history. Il primo comando accettato verrà riportato nella history con indice 1 e i comandi eseguiti successivamente verranno inseriti con indici crescenti. I comandi che sono stati accettati verranno aggiunti al file `~/.bash_history`, file a cui la history fa riferimento.

In **Tabella 4.2** sono riportati alcuni dei comandi riconosciuti da Bash per il richiamo degli elementi presenti nella history.

`C-p`	Richiama il comando precedente nella history
`C-n`	Richiama il comando successivo nella history
`M-<`	Punta al primo comando della history
`M->`	Punta l'ultimo comando nella history

Tabella 4.4.2: Comandi per il richiamo della history

`C-` si riferisce al tasto `Ctrl`. Nelle tastiere che possiedono i tasti freccia è possibile usare il tasto `freccia su` al posto di `C-p` e il tasto `freccia giù` al posto di `C-n`.

Alias: gli alias sono un modo semplice per abbreviare comandi lunghi, per utilizzare diversi comandi usando una sola parola, per invocare un certo programma sempre con le stesse opzioni utili. Vengono sono creati usando il comando `alias` e vengono distrutti usando il comando `unalias`. La sintassi di base è

```
alias name=value
```

Gestione attività: Bash consente di sospendere e riavviare processi e di spostare attività (job) in foreground o background. Viene supportato anche il comando `disown` che consente alle attività in background di continuare l'esecuzione nonostante la chiusura del processo (shell) genitore.

Operazioni aritmetiche: Bash supporta le operazioni aritmetiche dalla base 2 alla base 64 oltre alla maggior parte degli operatori aritmetici presenti in C. Valutazione e sostituzione delle espressioni numeriche sulla linea di comando sono built-in ed è possibile inserire espressioni aritmetiche come comandi.

Variabili: dalla versione Bash 2.0 vengono supportate variabili array di dimensione illimitata. Si possono quindi utilizzare variabili sia di tipo scalare che di tipo array.

Per impostare il valore di una variabile si utilizza uno statement di assegnamento standard, identico a quello presente in `sh`:

```
name=value
```

Le variabili di tipo array possono essere settate in tre modi diversi. Una prima versione consente di settare un singolo elemento:

```
Name[index]=value
```

Una seconda forma di assegnamento può essere usata per settare più elementi contemporaneamente. In questa versione vengono utilizzati indici consecutivi a 0.

```
name=(value0 … valueN-1)
```

Il terzo metodo prevede l'utilizzo di un indice prima dell'indicazione del valore:

```
name=([2]=value2  [0]=value0  [1]=value1)
```

Non è presente un valore massimo per gli indici negli array e non è necessario l'utilizzo di tutti gli indici consecutivi.

Bash supporta inoltre due comandi `declared` e `typeset` per la modifica di attributi di variabile. Tramite questi comandi è possibile cambiare il tipo di una variabile, renderla read-only e indicarla per l'esportazione. I due comandi sono sinonimi, tuttavia il comando `typeset` è ritenuto obsoleto per cui è preferibile l'utilizzo di `declared`.

I comandi `declared` e `typeset` accettano le opzioni presenti in **Tabella 4.3**:

`-a`	Set/unset dell'attributo array per una variabile
`[-/+] i`	Set/unset dell'attributo intero per una variabile
`[-/+] r`	Set/unset dell'attributo read only per una variabile
`[-/+] x`	Set/unset dell'attributo export per una variabile
`-p`	Mostra gli attributi di una variabile

Tabella 4.3: Attributi per declared e typeset

Una variabile settata come intero, quando usata in una espressione aritmetica, si comporta in modo simile al tipo `int` nel linguaggio di programmazione C. Le variabili di tipo read-only a seguito dell'assegnamento del valore non possono essere modificate.

Variabili di ambiente: all'avvio Bash imposta il valore di alcune nuove variabili (oltre a impostare il valore di quelle che erano già presenti nella Bourne shell). In **Tabella 4.4** è indicata una lista parziale di alcune di queste variabili aggiuntive.

`$PWD`	La directory corrente
`$UID`	ID numerico dell'utente corrente
`$BASH`	Pathname completo usato per invocare l'istanza corrente di bash
`$REPLY`	Ultima linea di input letta dal comando `read`
`$RANDOM`	Genera un intero random tra 0 e 32767
`$SECONDS`	Il numero di secondi che sono passati da quando la shell è stata invocata
`$HISTCMD`	Il numero indice del comando corrente nella history list

Tabella 4.4: Alcune nuove variabili d'ambiente

5 C shell

La C shell è stata sviluppata nella seconda metà degli anni ’70 da Bill Joy mentre era uno studente presso l’University of California, Berkeley. È stata rilasciata per la prima volta nel 1979 con la seconda distribuzione di software Berkeley (2BSD), di cui faceva parte anche la prima versione dell’editor di testo vi.

Ciò che distingueva la C shell da altre shell all’epoca erano le sue funzionalità interattive e il suo stile di sintassi. Questo la rendeva, quando è stata rilasciata, la shell più facile e veloce da usare sui sistemi UNIX.

Il comando per invocare la C shell è csh.

Poco dopo il suo rilascio, la C shell ha ricevuto due importanti aggiornamenti da parte di Ken Greer della Carnegie Mellon University e Mike Ellis dei laboratori Fairchild. Entrambi gli aggiornamenti hanno aggiunto funzionalità ispirate a quelle presenti sul sistema operativo TENEX, ed è per questo che la versione aggiornata della C shell è stata rinominata tcsh.

Dato che la tcsh ha solo aggiunto funzionalità alla csh originale è completamente retrocompatibile. Infatti, in molti sistemi operativi che implementano la csh e la tcsh come macOS e Red Hat Linux uno dei due file è semplicemente un link simbolico all’altro. In altri casi, come Ubuntu, è stata fatta la scelta di lasciare separate le due versioni.

5.1 Sintassi

La caratteristica più ovvia della C shell, quella da cui prende il proprio nome, è la sua sintassi particolare. Il sistema operativo UNIX era scritto completamente in linguaggio C, e l’obiettivo principale della C shell era quello di offrire un linguaggio di parsing dei comandi che fosse più stilisticamente coerente con il resto del sistema.

Da una prospettiva moderna la sintassi della C shell non sembra assomigliare particolarmente a quella del linguaggio C, soprattutto quando paragonata ad altri linguaggi di scripting. Ai tempi in cui l’unico paragone possibile era con la Bourne shell, per molti la sintassi della C shell era comunque un grande passo in avanti.

Una innovazione importante della C shell rispetto alla Bourne è la grammatica delle espressioni. Quando la Bourne shell incontra una keyword del controllo di flusso, il test che segue per determinare se la condizione è verificata deve essere eseguito in un altro processo, in seguito il processo originale esamina il valore di ritorno del processo test.

La C shell invece utilizza la grammatica delle espressioni per verificare la condizione senza aprire nuovi processi. Questo rende la C shell più veloce nell’esecuzione degli script.

Altri cambiamenti invece hanno natura più stilistica, come ad esempio endif invece di fi per indicare la fine di un costrutto if, favorire l’uso delle parentesi invece di keyword nei costrutti come if e for, eliminare la necessità di terminare ogni case con ;;, ecc.

5.2 Features

La C shell è stata la prima shell Unix a introdurre molte delle funzionalità per l'uso interattivo che oggi sono presenti su tutte le shell moderne, come:

- History dei comandi
- Operatori per l'editing
- Alias
- Stack delle directory (lista di ultime directory visitate)
- Notazione tilde per la home directory
- Controllo delle attività (job)
- Hashing dei percorsi

La tcsh ha poi introdotto altre due funzionalità fondamentali:

- Completamento automatico della linea di comando programmabile
- Editing della linea di comando

5.3 Diffusione

Lo sviluppo della C shell rende molto chiare le motivazioni dietro lo sviluppo dello standard POSIX. Tcsh offriva a sviluppatori e user nuove funzionalità molto utili, rare in altre shell dell'epoca, ma software scritto per la tcsh non poteva essere usato senza modifiche su sistemi operativi che utilizzavano altre shell. Allo stesso modo la tcsh non poteva eseguire script scritti in linguaggio sh. Non c'era un modo "giusto" di scegliere quale shell utilizzare e per quale shell scrivere il software. Cercare di scrivere software compatibile con tutte le versioni e configurazioni di Unix diventava sempre più laborioso.

Nonostante anche lo stesso Stephen Bourne fosse del parere che la csh fosse superiore alla sh in quanto all'utilizzo interattivo, non è mai diventata la shell più usata per lo scripting. Nell'epoca in cui era stata da poco rilasciata non vi era garanzia che la csh fosse presente su un qualunque sistema Unix. Alcuni anni dopo, quando la csh era diventata disponibile sulla gran parte dei sistemi Unix, lo standard POSIX era già stato sviluppato, e le parti dello standard in merito alle shell erano state basate in gran parte sulla sintassi della Korn shell, la quale era a sua volta stata sviluppata a partire dalla Bourne shell.

La C shell ricevette inoltre varie critiche riguardanti alcune scelte di design e implementazione:

- Erano presenti vari difetti di sintassi del linguaggio. Ad esempio, alcuni comandi avevano funzioni pressoché identiche, ma metodi di uso leggermente diversi senza un vero motivo.
- Alcune funzionalità disponibili nella Bourne shell non erano presenti in C shell, principalmente il supporto per le funzioni e la possibilità di manipolare i file legati allo standard input e allo standard output.

- L’implementazione del parser non era pienamente ricorsiva, quindi i comandi avevano un limite di complessità, superato il quale l’esecuzione si bloccava con un messaggio di errore criptico.
- I messaggi di errore in generale erano poco utili per individuare i problemi negli script.

6 Korn shell

La Korn shell è stata sviluppata presso AT&T Bell Laboratories da David Korn nella prima metà degli anni '80. È stata annunciata ufficialmente all'USENIX nel 1983 e poi rilasciata nel 1986. È ufficialmente diventata parte della distribuzione SVR4 (System V Release 4) di Unix nel 1988. La Korn shell era inizialmente un software con licenza proprietaria dell'AT&T. Nel 2000 è stato rilasciato il codice sorgente sotto una particolare licenza AT&T, ma dal 2005 ha una licenza pubblica sotto Eclipse. Ora è disponibile nella collezione di software open source AT&T Software Technology (AST).

La Korn shell originale ksh88 è stata usata come base per lo standard POSIX-IEEE Std 1003.2-1992. Questa prima versione della Korn shell è diventata la shell di default su AIX (una serie di sistemi operativi tipo Unix di proprietà di IBM) nella versione 4. Versioni successive della Korn sono shell di default per molti sistemi Unix e Unix-like.

Questa shell è stata sviluppata con il proposito di creare un superset della Bourne shell che comprendesse tutte le funzionalità aggiuntive offerte dalla C shell, senza però perdere la portabilità. Ha inoltre introdotto aggiunte proprie, come metacaratteri estesi e nuove keyword di controllo di flusso (ad esempio il loop select e il comando let).

Poiché era inizialmente un software proprietario, ma ha influenzato fortemente un importante standard, sono state sviluppate molteplici versioni alternative gratuite e open source, tra cui la Public Domain Korn Shell (`pdksh`), e la MirBSD Korn Shell (`mksh`).

6.1 Features

Le feature della Korn shell coincidono quasi completamente con quelle della Bourne Again shell. Le due shell presentano alcune differenze, per lo più legate alla performance, tra cui:

- La bash ha varie opzioni di personalizzazione dell'esperienza interattiva in più rispetto alla ksh, come ad esempio un modo immediato di riportare la directory corrente nella linea del prompt.
- La sintassi dei loop della ksh è tale per cui è possibile settare un valore all'interno del loop e poi richiamarlo fuori dal loop.
- La bash gestisce in maniera più pulita i codici di uscita delle pipe.
- La ksh usa il comando print invece del comando echo, per riportare in output le stringhe. print è generalmente più veloce ed efficiente di echo.
- Gli array associativi (oggi chiamati più comunemente "dizionari"), da sempre presenti nella ksh, sono stati introdotti relativamente di recente nella bash, per cui alcune release di sistemi che usano bash di default potrebbero non avere ancora questa funzionalità.
- La ksh supporta l'uso di coprocessi, processi figli asincroni legati allo standard input e output del processo padre tramite pipe.
- Svariati casi di sintassi e comportamenti di comandi leggermente diversi, principalmente in circostanze non esplicitamente definite dallo standard POSIX.

Generalmente gli esperti sono del parere che la ksh sia un linguaggio leggermente più adatto allo scripting, mentre la bash offre un'esperienza interattiva leggermente migliore.

C'è però da tenere in mente che se si ha un interesse per la portabilità l'approccio migliore allo scripting sarebbe quello di usare solo funzionalità disponibili nella Bourne shell originale. L'utilizzo della ksh per lo scripting è ulteriormente svantaggiato dal fatto che anche eseguendo lo stesso script su sistemi che dispongono di ksh è spesso molto difficile determinare se il sistema in questione usa ksh88 o una delle numerose versioni di ksh uscite negli anni, o addirittura una qualche altra shell con il "carattere" della Korn.

È importante sottolineare che nell'utilizzo di tutti i giorni le differenze che è possibile notare tra la ksh e la bash sono talmente minuscole che, se si è esperti di una delle due, è difficile giustificare il passaggio all'altra.

7 POSIX

Molte delle motivazioni per le scelte progettuali e le caratteristiche della shell POSIX sono strettamente legate alle circostanze e alla storia del suo sviluppo. Questi aspetti sono stati approfonditi più in dettaglio nell'appendice **"Storia dello standard POSIX".**

7.1 Shell POSIX

Lo standard che tratta delle shell è IEEE POSIX 1003.2. Questo standard non era stato pensato per essere rigido e assoluto, ma era progettato per essere flessibile al punto di consentire sia la coesistenza di software simili (in modo che il codice preesistente non diventasse obsoleto), sia la possibilità di aggiungere nuove funzionalità (in modo che i venditori avessero un incentivo per innovare). Il risultato fu che la maggior parte dei venditori Unix si conformarono allo standard.

La parte dello standard relativa alle shell descrive i servizi che devono essere presenti su tutti i sistemi e altri che sono opzionali, a seconda della natura del sistema. Una di queste opzioni è la User Portability Utilities option, che definisce lo standard per le shell ad uso interattivo e i servizi interattivi, come ad esempio vi (cioè il VIsual Editor).

Lo standard POSIX è stato basato principalmente sulla Bourne shell di System V. Perciò si potrebbe assumere che le funzionalità presenti nella korn shell, ma non nella bourne non sono trattate nello standard. In realtà, durante il suo sviluppo è stato deciso di inserire alcune delle funzionalità esclusive della korn shell in POSIX, tra cui la sintassi con $(( )) per le espressioni aritmetiche e con $(…) per la sostituzione dei comandi, e per l'espansione della tilde. Ci sono poi altre funzionalità presenti nella korn shell che non sono state inserite nello standard, e perciò la loro sintassi è accettata ma il loro funzionamento non è standardizzato.

Lo standard POSIX supporta le funzioni, ma la semantica definitavi è più debole di quella presente nella korn shell. Ad esempio, le funzioni da definizione di POSIX non prevedono variabili locali.

Alcune delle funzionalità introdotte rispetto alla Bourne shell sono state:

- L'ordine di ricerca dei comandi è stato modificato per permettere a certi comandi built-in di essere sovrascritti da funzioni. I comandi built-in sono inoltre stati divisi in due gruppi: quelli processati prima delle funzioni e quelli processati dopo (nel senso che alcuni hanno priorità sulle funzioni, come ad esempio break, continue, exec).

- Un nuovo comando built- in, "command", che consente di inserire nuovi comandi in uno dei due gruppi descritti sopra.

- Una nuova keyword, "!", assume il ruolo della negazione logica per il codice di uscita di un comando. Se un comando "cmd_esempio" ritorna 0, "!cmd_esempio" ritorna 1, se invece "cmd_esempio" ritorna un valore diverso da 0, "!cmd_esempio" ritorna 0.

Infine, dato che lo standard POSIX era scritto per promuovere la portabilità, al suo interno si evita di menzionare le funzionalità relative alla shell di uso interattivo, come alias e modalità di editing. Inoltre, si evita di menzionare anche i requisiti per l'utilizzo di multitasking per i job in background. Questa scelta è stata fatta per consentire la portabilità su sistemi non multitasking, come ad esempio MS-DOS.

Quindi non esiste una sola shell POSIX ma ne esistono tante interpretazioni diverse, più o meno derivate da quella di Bourne. In diversi sistemi operativi Unix sono installate di default diverse shell, ma per convenzione la shell che si trova in /bin/sh è sempre POSIX compliant.

8 Confronto tra shell

La maggior parte delle versioni di UNIX attuali rendono disponibili tre shell diverse: la Bourne e/o la POSIX shell, la C shell e la Korn shell. Tuttavia, è possibile utilizzare anche altre shell, come ad esempio la Bourne Again shell. Scegliere la giusta shell da utilizzare è una decisione importante perché verrà impiegata una quantità di tempo e sforzo considerevole per imparare a usarla efficientemente. La scelta giusta può consentire di beneficiare di molteplici funzionalità di UNIX con il minimo sforzo possibile.

Gli utilizzi principali di una shell sono:

- come interfaccia testuale con il sistema operativo
- come mezzo per scrivere script ad uso personale
- come linguaggio di programmazione per creare nuovi script per altri utilizzatori

Le diverse shell sono in grado di fornire un livello di supporto differente per ognuna di queste tre esigenze. Questo capitolo descriverà i vantaggi e gli svantaggi dell'utilizzo delle shell sopra citate per soddisfare le tre esigenze appena elencate.

8.1 Uso interattivo

Il primo punto da tenere in considerazione nella scelta di una shell per uso interattivo è che la decisione presa influenzerà soltanto l'utilizzatore. È una scelta personale, che non preclude l'utilizzo di script scritti in altre shell.

I fattori principali che possono influenzare la scelta per una shell interattiva sono:

- **Esperienza pregressa**: alcune shell sono ricche di strumenti non standardizzati che possono arricchire e facilitare il loro utilizzo interattivo. Se si è già molto esperti con una di queste shell i vantaggi del passaggio ad un'altra shell potrebbero essere pochi e potrebbero richiedere un lungo periodo di apprendimento. I due stili di sintassi principali sono quello della Bourne shell e quello della C shell. Avere, infatti, familiarità con la Bourne shell è un vantaggio quando si utilizzano varianti di questa, come ad esempio la Korn shell, ma anche uno svantaggio nel caso si utilizzi la C shell.
- **Apprendimento**: alcune shell hanno un grado di complessità maggiore di altre. Per apprendere nuove funzionalità è necessario tempo da dedicare all'apprendimento e alla pratica. La famiglia della Bourne shell, ad esempio, ha un linguaggio di programmazione più ricco e criptico rispetto a quello della C shell.
- **Gestione e modifica dei comandi**: tutte le shell offrono qualche funzionalità per aiutare nella scrittura di nuovi comandi, ma alcune hanno più funzionalità di altre.
- **Wildcard e shortcut:** alias, wildcard e completamento possono risultare utili per risparmiare tempo e per la scrittura dei comandi.
- **Portabilità:** se si utilizza sempre lo stesso terminale, con lo stesso software e applicazioni UNIX, e si interagisce raramente con sistemi non familiari, conviene

utilizzare gli strumenti migliori disponibili per il proprio sistema. Se invece non fosse così e si dovesse interagire con computer diversi che posseggono versioni di UNIX differenti allora converrà utilizzare strumenti che sono disponibili su tutti i sistemi utilizzati. Conoscere una shell che sia disponibile su tutti i sistemi è importante e porta a grandi vantaggi.

sh: in confronto ad altre shell, la Bourne shell ha in generale meno funzionalità. Questo la rende la più semplice da apprendere e porta grandissimi vantaggi in termini di portabilità. Date le poche funzionalità che mette a disposizione dell'utente, essere molto esperti nel suo utilizzo non disincentiva l'apprendimento di altre shell più elaborate.

csh: per certi aspetti la difficoltà dell'apprendimento e dell'utilizzo della C shell sono particolarmente alti rispetto ai benefici che offre. L'ostacolo maggiore non è la pura quantità di materiale da apprendere, ma la scarsa documentazione disponibile. Infine, la sua sintassi particolare la rende incompatibile con qualunque shell della famiglia Bourne.

ksh: le difficoltà di apprendimento della Korn shell, invece, derivano principalmente dalla miriade di strumenti disponibili. Poiché è un'estensione della Bourne shell è anche relativamente portabile. Un esperto della Korn shell ha a disposizione tutti gli strumenti per un utilizzo efficiente di un qualunque aspetto del sistema operativo, perciò non ha, a priori, molti incentivi per imparare l'utilizzo di altre shell.

bash: la Bourne Again shell potrebbe essere una delle shell più difficili da apprendere complessivamente. È stata basata sulla versione della Korn shell del 1988, ma negli anni ha ricevuto numerosi aggiornamenti che ne hanno ampliato di molto le funzionalità. Anche la bash è relativamente portabile grazie alla sua somiglianza alla Bourne shell, ma di solito si trova preinstallata solo sui sistemi Linux.

8.2 Script per uso personale

Nel caso si vogliano scrivere script per uso personale conviene che per questi venga utilizzato lo stesso linguaggio che si utilizza già per i comandi interattivi.

Sia che si utilizzi una variante della C shell o una variante della Bourne shell conviene tenere in considerazione l'utilizzo del linguaggio della Bourne shell per gli script. Si potrebbe sempre verificare una circostanza in cui si vogliono utilizzare i propri script su una macchina diversa da quella che si usa di solito, in questo caso, l'utilizzo di un linguaggio diverso da quello della Bourne potrebbe portare a rischi di incompatibilità nei diversi ambienti di lavoro.

Conviene inoltre tenere in considerazione per la scrittura di script che la C shell e le sue varianti mancano di una serie di funzionalità utili per la programmazione. Ad esempio, la mancanza di sintassi relativa alle funzioni inibisce di molto la programmazione strutturata. Ogni funzione deve essere scritta in un file separato o essere dichiarata come un alias. Nelle varianti della C shell è inoltre assente la sostituzione dei parametri e mancano molti dei test e delle variabili relativi ai file.

8.3 Script per uso pubblico

Gli script shell pensati per l'uso pubblico devono avere una portabilità duratura nel tempo. Per questo motivo la maggior parte degli script vengono scritti nel linguaggio della Bourne shell. Script che non stati scritti nel linguaggio Bourne o POSIX richiederanno l'installazione della differente shell nel sistema operativo.

Alcune versioni di UNIX consentono di specificare l'interprete shell da usare per un certo script inserendo un comando speciale nella prima linea dello script, detto "shebang". Inserendo ad esempio `#! /bin/sh` nella maggior parte dei moderni sistemi UNIX verrà forzato l'utilizzo della Bourne shell per l'esecuzione del file. Con l'utilizzo della shebang è quindi possibile disaccoppiare la scelta del linguaggio usato nello script dalla scelta della shell interattiva dall'utente.

Per ottenere una migliore portabilità è consigliabile seguire queste linee guida:

- per progetti di grandi dimensioni e importanza conviene scegliere la shell con cui si ha più familiarità e che si preferisce. Sarà poi compito degli altri utenti l'installazione della eventuale specifica shell nei propri sistemi.
- se lo script che si sta scrivendo potrebbe entrare nel dominio pubblico converrà utilizzare il linguaggio della Bourne shell.
- se il progetto deve rispettare alcuni obiettivi di compatibilità allora converrà utilizzare un linguaggio in grado di soddisfare in primis tali obiettivi.
- in tutti gli altri casi converrà scegliere il linguaggio che si ritiene più efficiente, che crea meno problemi e che consenta di massimizzare la propria produttività.

Conclusione

In questo elaborato sono state esaminate in maniera dettagliata alcune delle shell più diffuse e attualmente utilizzate nei sistemi UNIX/Linux. L'analisi si è conclusa con il confronto tra le diverse shell in un'ottica progettuale, per la ricerca della shell che si allinea maggiormente agli scopi dell'utente.

Si è concluso che, per l'utilizzo interattivo della shell, sia importante tenere in considerazione anche le conoscenze pregresse e il tipo di produttività ricercata dall'utilizzatore, oltre alle feature esclusive offerte da ciascuna shell.

Per quanto riguarda l'utilizzo della shell come linguaggio per la produzione di script, invece, la scelta dipende anche da quali saranno i fruitori di tale script. Se lo script sarà dedicato all'uso personale converrà scegliere un linguaggio con cui si ha già competenza ed esperienza, mentre se lo script sarà per il consumo pubblico converrà utilizzare un linguaggio che favorisca la portabilità e la compatibilità sui diversi sistemi.

Appendice – Storia dello standard POSIX

Posix è diventato l'acronimo per l'importante standard conosciuto come Portable Operating System Interface for Computer Environments.

L'organizzazione chiamata originariamente /usr/group era il precursore di POSIX quando è stata fondata come gruppo commerciale degli utenti UNIX nel 1980, per focalizzarsi sul sistema operativo UNIX di AT&T. L'organizzazione è stata successivamente rinominata UniForum. All'epoca in cui il gruppo si stava formando numerose compagnie avevano iniziato a installare sistemi operativi UNIX-based e UNIX-compatible sui minicomputer e microcomputer. Le differenze tra le varie versioni hanno destato preoccupazione nei membri del /usr/group, che ha quindi stabilito una commissione formale apposita.

Per mantenere il processo gestibile, il gruppo iniziale era limitato a 40 persone. A capo del gruppo c'era Heinz Lycklama di Interactive Systems. È stato deciso che era richiesta una maggioranza dei due terzi per prendere decisioni.

All'epoca AT&T stava introducendo la sua UNIX System 3 che usava la Versione 7 di UNIX. L'università California stava sviluppando un derivato chiamato BSD Versione 4. Diversi venditori commerciali (tra cui Microsoft) stavano promuovendo versioni di UNIX per diversi sistemi sui minicomputer. Il primo microprocessore target della commissione fu Zilog 8000, con una implementazione di Onyx, e diverse implementazioni su sistemi PDP Digitale 11/23. A questo punto, almeno tre prodotti Unix-like erano disponibili sul mercato (Idris, Coherent e Unos). I percorsi divergenti delle versioni di licenza di AT&T e l'esistenza di implementazioni sviluppate in modo indipendente hanno impostato il tono del lavoro allo standard.

LINEE GUIDA

La partecipazione dei tre fornitori non sotto licenza di AT&T ha portato alla istituzione di due linee guida chiave nel gruppo:

- Il documento risultante doveva essere indipendente da uno specifico prodotto come punto di riferimento. In questo modo sviluppatori indipendenti possono sviluppare sistemi conformi con lo standard, senza dover comprare per forza sistemi da un unico fornitore.

- Il focus doveva essere centrato sulla descrizione di un'interfaccia.

Questo approccio con target inquadrati da venditori indipendenti è stato fondamentale per l'accettazione nel processo degli standard accreditati. Il focus sulla portabilità di applicazioni fu inoltre importante per stabilire lo scope per il progetto. Il gruppo si focalizzò sulle interfacce dei programmi piuttosto che sugli aspetti amministrativi, comunicativi, di utility o di shell del sistema. I sistemi tradizionali Unix organizzavano la documentazione in una maniera che parallelizzava l'implementazione, separavano gli oggetti implementati come funzioni di libreria da quelli implementati in maniera intrinseca. Il progetto di /usr/group eliminò tale distinzione sostituendola con un rigoroso elenco in ordine alfabetico. Processi analoghi furono necessari anche per vari servizi specifici al linguaggio C e altri sistemi operativi.

A metà del 1984 il documento era completo e fu distribuito ai membri di /usr/group per le votazioni. I risultati furono pubblicati come "1984 /usr/group Standard". Il gruppo poi pensò a come portare il lavoro davanti all'American National Standard Institute (ANSI) e/o all' International Organization for Standardization (ISO), per farlo accettare come standard accreditato. Indipendentemente dal progetto di /usr/group, la IEEE aveva autorizzato nel 1983 un altro progetto per lavorare a uno standard per i kernel di sistemi operativi basati sui sistemi operativi Unix. Alcuni membri del gruppo /usr/group si unirono al progetto di IEEE con il tentativo di promuovere l'allineamento, per poi scoprire che il progetto era temporaneamente inattivo.

Nonostante ciò, /usr/group trasferì il suo lavoro all'IEEE nel gennaio del 1985 e i membri attivi della commissione si allinearono con la IEEE. L'IEEE incontrava l'obiettivo principale del gruppo: possedeva lo status di sviluppatore accreditato di standard ANSI. Ma il fattore decisivo fu che il lavoro allo standard IEEE dipendeva da coinvolgimento individuale dei professionisti. Questo differenziava il lavoro agli standard IEEE da gruppi in cui la rappresentazione istituzionale era più comune.

RAGGIUNGIMENTO DEL CONSENSO

Per approvare un qualunque dettaglio era necessaria una maggioranza del 75% non solo degli sviluppatori, ma di tutti i membri del progetto. Ogni ente interessato allo sviluppo di Posix aveva interessi personali in merito. Perciò il lavoro da fare era sostanziale, nuovi metodi e procedure per il confronto tra i membri sono state trovate e implementate con lo scopo di riuscire a concludere il lavoro in tempi utili.

SCRITTURA DELLE MOTIVAZIONI DELLE SCELTE

Nonostante ciò, non tutte le obiezioni erano risolvibili. Passò del tempo prima che il gruppo si rendesse conto della necessità di documentare non solo le decisioni prese, ma anche le motivazioni dietro ogni decisione. /usr/group ingaggiò il personale per scrivere le motivazioni delle scelte per il documento nel 1986. Le motivazioni sono state uno strumento utile per identificare, ad esempio, falle nelle motivazioni date in precedenza. Per accelerare lo standard IEEE, sia /usr/group che AT&T resero disponibili i loro testi come base. A partire dal lavoro iniziale di /usr/group, è stata iniziata la stesura di 3 documenti: il documento di Posix, lo standard del linguaggio C X3J11 e la definizione delle interfacce per il System V di AT&T (SVID).

AT&T aiutò il gruppo a capire dove Posix differiva da System V, e ciò costituì un altro elemento attraverso cui identificare obiezioni e trovare motivazioni adatte.

/usr/group adottò a questo punto un ruolo di commissione tecnica e fornì un forum per le discussioni e lo sviluppo di concetti oltre l'ambito del lavoro dell'IEEE. Concetti come l'operazione in real time, sicurezza e internalizzazione. Il gruppo formatosi contemporaneamente che lavorava per lo standard del linguaggio C scatenò una delle prime dispute "territoriali" incontrate dal gruppo Posix. Poiché il documento di /usr/group del 1984 non aveva separato la parte riguardante il linguaggio C dalla parte riguardane il sistema operativo, a questo punto entrambi i gruppi (Posix e X3J11) stavano sviluppando interfacce in alcune aree sovrapposte. La faccenda si risolse con una decisione unilaterale dal gruppo Posix di concedere il più possibile al gruppo di X3J11. Il gruppo Posix determinò che tutti i sistemi Posix avrebbero usato il linguaggio C, ma che il linguaggio C non avrebbe avuto una

dipendenza da Posix. Perciò, fu deciso che l'ambiente di C doveva essere il più ricco possibile. Comunque, il gruppo del C non accettò tutte le cose che il gruppo Posix era disposto a cedere e molto fu reintegrato nel documento IEEE in seguito.

ESPANSIONE DELLA PARTECIPAZIONE

La transizione sotto uno standard formale attirò altre persone. Il gruppo originale di 40 crebbe in fretta a 70 e aumentò anche la partecipazione dei principali venditori di computer. Anche il National Institute of Standards and Technology, prima noto come National Bureau of Standards (NBS), fu un partecipante attivo. Inizialmente NBS centrava solo con il processo di valutazione. Quando divenne chiaro che il governo avrebbe goduto di numerosi vantaggi offerti dai sistemi aperti, NBS iniziò ad avere un ruolo proattivo.

La convergenza dell'interesse degli utenti di NBS e l'impatto dei nuovi sistemi tecnologici (microprocessori, workstation e RISC) si combinarono per focalizzare molte risorse su Posix, rendendo il progetto di Posix molto diverso dagli altri standard. Il gruppo di lavoro di Posix cercò di far uscire il documento in maniera che rispondesse ai bisogni dell'industria, invece di soddisfare formalismi. L'organizzazione del materiale non era adeguatamente strutturata, ed i requisiti erano espressi in un linguaggio poco coinciso. Tuttavia, se queste cose fossero state tentate nel 1985 il documento iniziale non sarebbe stato completato in tempo e la finestra per l'accettazione sarebbe stata chiusa.

VERSIONE TRIAL-USE

Nel 1986 il gruppo è riuscito a votare con successo una versione trial-use di Posix. È stato rivisto il titolo iniziale del progetto per rimuovere riferimenti al concetto di "kernel" e focalizzare l'attenzione sulle interfacce. Il documento è uscito in tempi record: lo sviluppo ebbe inizio nel gennaio del 1985; il ballottaggio finale nel dicembre dello stesso anno; pubblicazione del documento per il trial-use nell'aprile del 1986.

La versione trial-use ha consentito al gruppo di rilasciare un documento che rappresentava il 90% del risultato definitivo e che era in grado di evidenziare in maniera efficace i problemi da affrontare restanti. A partire dalla riunione prima del ballottaggio del 1985 sono stati cambiati alcuni aspetti delle modalità di incontri. Fino a quel momento tutto il lavoro era stato fatto in un unico gruppo principale. Dal 1986 il gruppo principale è stato diviso in piccole unità, ognuna focalizzata su un'area specifica, ogni unità poi portava problemi e proposte davanti al gruppo intero per discuterne. Nel 1986 si è iniziato ad espandere il lavoro, aggiungendo gruppi per le shell e strumenti, per metodi di test, per l'operazione real time.

Successivamente al rilascio della versione trial-use sono effettivamente emerse varie decisioni da prendere prima di avere una versione completa: problemi con segnali, gestione di nomi di file troppo lunghi, gestione di operazioni input/output e di interruzione tra processi, locking di file e record.

In questo periodo è stato anche iniziato il lavoro per lo standard internazionale ISO. Il gruppo Posix si è posto come obiettivo quello di rendere le release degli standard IEEE e ISO documenti identici in tutti i casi dove ciò era possibile.

La prima release completa dello standard Posix risale al 1988 con il nome IEEE Std 1003.1-1988. Fu inizialmente rilasciato come un documento unico riguardante l'interfaccia di programmazione principale (core) ma fin dal 1986 furono iniziati documenti che trattavano

parti del sistema più nello specifico: Shells and Utilities (1003.2), Test Methods (1003.3), Real Time (1003.4). Andando avanti lo standard Posix finì per essere composto da ben diciannove documenti separati.

Bibliografia e sitografia

Robin Burk and David B. Horvath, CCP, et al. - *UNIX® Unleashed, System Administrator's Edition*

http://web.archive.org/web/20040620012238/http://docs.rinet.ru/UNIXs/fm.htm

James Isaak – *The history of Posix: a study in the standards process*

Bill Rosenblatt and Arnold Robbins – *Learning the Korn Shell, second edition* https://docstore.mik.ua/orelly/unix3/korn/index.htm

IBM Corporation – *Operating system shells*

https://www.ibm.com/docs/en/aix/7.2?topic=administration-operating-system-shells

IEEE and The Open Group – *The Open Group Base Specifications Issue 7, 2018 edition IEEE Std 1003.1-2017*

https://pubs.opengroup.org/onlinepubs/9699919799/utilities/V3_chap02.html#tag_18_09