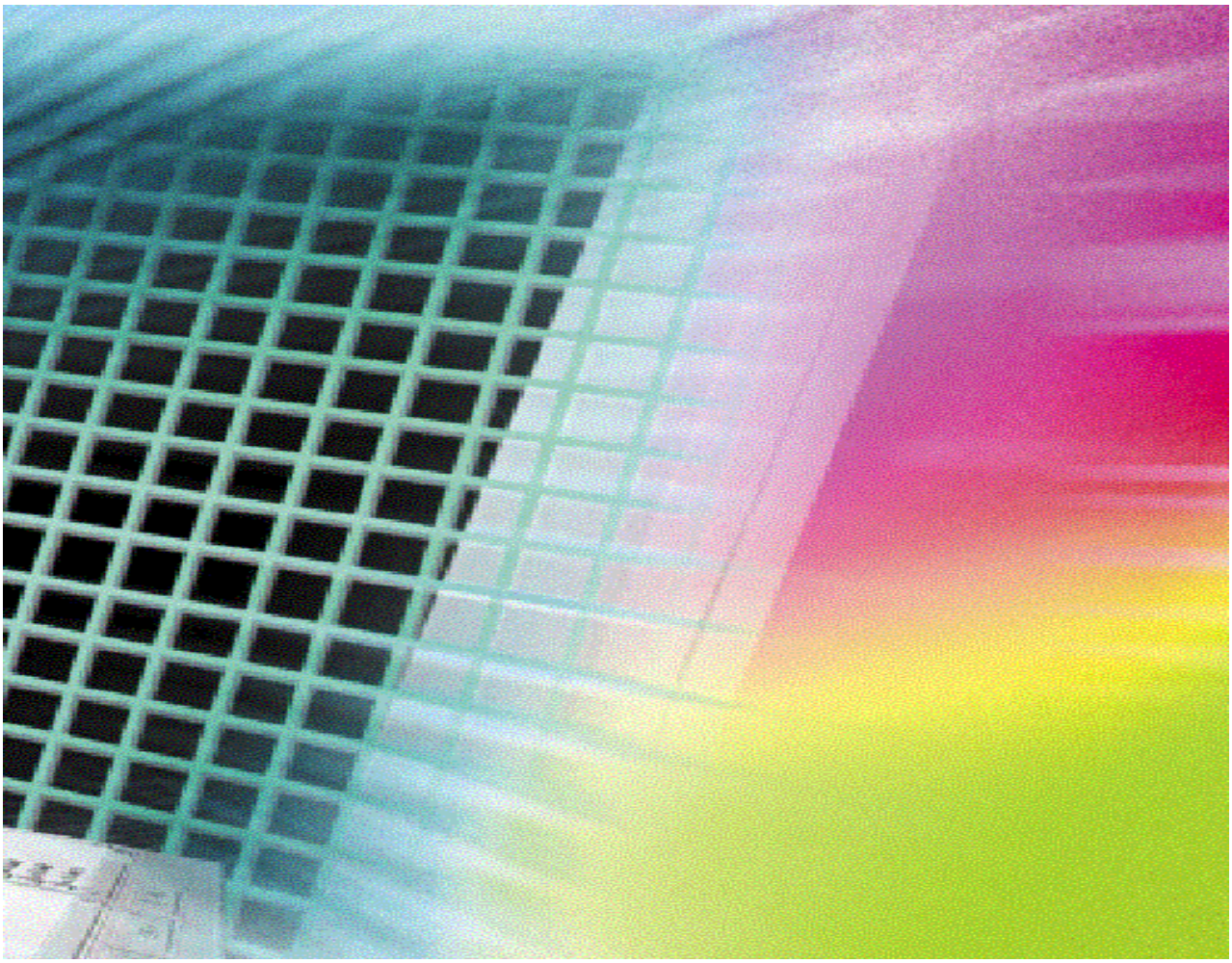
MASSIMO UBERTINI



PHP

WWW.UBERTINI.IT

PIATTAFORMA AMP (APACHE, MYSQL, PHP)

AMP SOTTO WINDOWS

Apache (www.apache.org)

È buona regola effettuare tutte le modifiche ai file di configurazione, sia di Apache sia del modulo PHP, a server spento in modo da essere sicuri che al riavvio del server stesso siano caricate le nuove impostazioni.
Dopo aver attivato il server web, le pagine locali saranno raggiungibili, mediante il browser, all'indirizzo **http://localhost** oppure **http://127.0.0.1**.
Apache, come server web, ha necessità di essere associato al numero della macchina su cui è installato in modo da poter "rispondere" alle richieste dei visitatori connessi. Teoricamente il problema sorge quando decidiamo di eseguire l'installazione in locale; infatti, presupponendo che faremo le prove off line, non avremo un IP canonico da inserire. In che modo, allora, associamo il server alla nostra macchina?
La risposta, da quanto detto prima, l'abbiamo già: i PC non connessi ad Internet, ma con un accesso remoto installato, hanno un indirizzo di default indicato dal numero 127.0.0.1 o, in alternativa, localhost. Volendo, possiamo raggiungere le nostre pagine anche in un altro modo oltre a *localhost* o *127.0.0.1*. Bisogna installare una connessione di accesso remoto sul PC (di qualunque tipo: modem analogico, ISDN, XDSL). L'accesso remoto, infatti, installa dei "pacchetti di rete" che permettono un'ulteriore identificazione del PC stesso oltre ai nomi di default (localhost, 127.0.0.1). In pratica, possiamo assegnare al PC qualsiasi nome; per fare questo apriamo il Pannello di controllo di Windows e clicchiamo sull'icona **Prestazione e manutenzione**; selezioniamo **Sistema** nella finestra **Proprietà del sistema** nella scheda **Nome computer** avremo davanti tre campi di testo:

1. Descrizione computer.
2. Nome Computer.
3. Gruppo di lavoro.

Se decidiamo di raggiungere le nostre pagine in locale digitando, ad esempio, il nome *pippo* basterà inserire questa parola nel campo **Descrizione computer** e riavviare il PC.
Da questo momento, una volta installato, il nostro server web sarà raggiungibile in tre modi diversi:

1. http://localhost
2. http://127.0.0.1
3. http://pippo (oppure *nome_scelto_da_noi*; per comodità)

Passiamo all'installazione vera e propria del server, vediamo i passi da seguire.

1. Clicchiamo sul file .EXE; dopo aver accettato i termini della licenza, avremo di fronte una schermata con tre campi da riempire; anche se non è un'operazione fondamentale per i primi due (potrebbero essere lasciati vuoti visto che siamo in locale), inseriamo il nome del nostro PC (quello che abbiamo scelto prima *pippo*) mentre nel terzo campo (è fondamentale) scriviamo la nostra e-mail.
2. Spuntiamo il radiobox relativo a **Run as a service for All Users -- (Recommended)** e clicchiamo su *Next*.
3. Selezioniamo *Setup Complete* e andiamo avanti.
4. Scegliamo la directory dove installare Apache. Per comodità lasciamo quella di default: **C:\Programmi\Apache Group** (farò riferimento sempre a questa d'ora in poi).
5. Clicchiamo su **Install**, aspettiamo che finiscano le operazioni e riavviamo il PC.

Facciamo una prova: dal menu **Start** aprire la cartella **Apache httpd Server** e selezionare *Start Apache in Console*.

Se tutto è andato bene, si apriranno due finestre del DOS; una sarà richiusa immediatamente mentre nell'altra, rimasta attiva, ci sarà scritto *Apache/1.3.24 <Win32> running...*
Il web server è acceso e pronto a lavorare. Per un'ulteriore conferma aprire il browser e digitare **http://localhost** o più semplicemente **http://pippo**: avremo davanti una semplicissima pagina web.
Vi starete forse chiedendo da dove sia uscita la pagina HTML che abbiamo appena visto. Ebbene, di default, Apache "cerca" le pagine da caricare in una cartella (all'interno della directory Apache Group\Apache) chiamata **htdocs** ed è qui che, in teoria, andranno i nostri lavori in PHP per essere visualizzati. È possibile, comunque, variare il percorso di questa cartella e, ad esempio, sceglierne una a piacere dopo averla creata. Mettiamo di voler avere, come directory base, la cartella **test**; dal menu "start/Tutti i programmi/Apache httpd Server/Configure Apache Server", selezioniamo il file **Edit the Apache httpd.conf Configuration File** (è il file che gestisce tutte le opzioni del server e si edita con il Notepad). Una volta aperto, con la funzione *Cerca*, arriviamo alla riga **DocumentRoot** (di default dovrebbe essere "C:/Programmi/Apache Group/Apache/htdocs") e cambiamo **htdocs** in **test**. Poco più sotto troveremo **<Directory "C:/Programmi/Apache Group/Apache/htdocs">** anche in questo caso dovremo sostituire *htdocs* in *test* (affinché le nuove impostazioni abbiano effetto sarà necessario riavviare il server). Per adesso sono finite le modifiche da apportare al file di configurazione del web server; d'ora in poi la cartella di lavoro sarà quella appena creata e sarà qui che dovremo mettere tutte le pagine da provare. Per poter lavorare correttamente dobbiamo conoscere i comandi base utilizzati da Apache. Nelle versioni precedenti, all'interno della cartella Apache httpd Server, c'erano delle icone che permettevano di far ripartire o fermare il server. A partire dalla 1.3.20, invece, è presente solo l'icona per lo *start* mentre le altre operazioni andranno eseguite tramite DOS (non usare CTRL + C che chiuderebbe la sessione invece di arrestarla). Apriamo, quindi, il prompt del DOS e, digitando questa riga, spostiamoci nella cartella dove è installato Apache: C:\>cd programmi\apache~1\apache (la tilde (~) si ottiene premendo il tasto ALT alla sinistra della barra spaziatrice, digitando consecutivamente i caratteri 1 2 6 del tastierino numerico e rilasciando l'ALT). Una volta nella directory di Apache possiamo scegliere tra tre operazioni.

1. Fermare il server digitando **apache -k stop**
2. Far ripartire il server scrivendo **apache -k restart**
3. Attivare il server direttamente da DOS digitando **apache -k start**

Questi comandi ci saranno molto utili in seguito quando, apportate alcune modifiche per il PHP, dovremo far ripartire la macchina. Per non avere sempre in primo piano la finestra del DOS quando avviamo il server, create un collegamento sul desktop dell'icona *Start Apache in Console*; cliccate con il tasto destro del mouse sul collegamento e scegliete *Proprietà*; dal menu a tendina del comando *Esegui* selezionate **Ridotto a icona** e date l'OK in questo modo saranno più veloci le operazioni di avvio e non avrete più il problema del prompt. Sempre relativamente al DOS impostare, come directory di lavoro, quella in cui è installato Apache; infatti, il DOS lo userete spesso solo nei casi in cui vorrete fermare o far ripartire il server in questa maniera non dovremo spostarci ogni volta ma ci troveremo direttamente nella cartella giusta. Per fare questo aprite un prompt e cliccate con il tasto destro del mouse sulla barra del titolo; selezionate *Proprietà* e, nel campo *Directory di lavoro*, scrivete il percorso per arrivare alla cartella di Apache: in questo caso sarà **c:\programmi\apache~1\apache**; infine cliccate su *Applica*.

PHP (www.php.net)

Ora che Apache è in grado di funzionare correttamente, possiamo installare il modulo PHP. Dobbiamo scompattare il file in una qualsiasi directory sul nostro hard disk; anche in questo caso, per comodità, ho scelto **C:\Programmi** e, d'ora in poi, farò riferimento sempre ad essa. Una volta scompattato noteremo che la cartella ha nome **PHP-4.1.2-Win32** rinominiamola semplicemente in **PHP**.
Apriamo la cartella e cerchiamo il file **PHP.ini-dist**; una volta individuato dobbiamo copiarlo ed incollarlo nella cartella *C:\Windows* (c:\winnt40 o c:\winnt per i possessori di Windows NT/2000) ricordandoci di rinominarlo in **PHP.ini**. Ora che è stato rinominato apriamo il file con un qualsiasi editor di testo; con la funzione *Cerca* individuiamo la riga contenente quest'istruzione:
doc_root =
in verità questo parametro potrebbe anche essere lasciato in bianco ma, per completezza, riempiamolo con il percorso che porta alla cartella base di Apache (quella che contiene i documenti da testare); facendo riferimento a quanto settato prima nel file di Apache, scriveremo: **C:\Programmi\Apache group\Apache\test**.
Nella cartella PHP dobbiamo cercare un file chiamato "**PHP4ts.dll**", è fondamentale per il funzionamento del modulo PHP può essere considerato come il motore principale. Una volta trovato dobbiamo copiarlo nella directory **C:\Windows\System** (C:\Winnt\System32 per i possessori di Windows NT/2000).
Con i file di configurazione del PHP abbiamo finito adesso dobbiamo modificare nuovamente il file *httpd.conf* di Apache. Vediamo i passaggi da attuare.
1) Sempre con l'apposita funzione del Notepad cerchiamo la riga.
"**#LoadModule unique_id_module modules/mod_unique_id.so**" e, subito sotto, aggiungiamo quanto segue:
LoadModule PHP4_module c:/programmi/PHP/sapi/PHP4apache.dll
(se necessario modificare il percorso che porta al file PHP4apache.dll)
2) Cerchiamo la riga "**AddModule mod_setenvif.c**" e sotto aggiungiamo:
AddModule mod_PHP4.c
3) L'ultima fase cerchiamo "**AddType application/x-tar.tgz**" e aggiungiamo:
AddType application/x-httpd-PHP.PHP
AddType application/x-httpd-PHP.PHP3
Da notare che il riferimento fatto a *AddType application/x-tar.tgz* è del tutto personale; infatti le due righe aggiunte potevano essere inserite benissimo in altri punti del file di configurazione.
Per completare l'opera non ci resta che impostare le pagine predefinite da caricare. Di default, digitando http://localhost (o 127.0.0.1), Apache cercherà la pagina chiamata *index.html*; se non è presente sarà visualizzato il contenuto della directory *test*.
Dato che a noi interessa testare le pagine PHP, può essere comodo indicare al web server di cercare, oltre all'index con estensione .HTML, anche altri tipi di file.
Spostiamoci, quindi, fino alla riga "**<IfModule mod_dir.c>**" e trasformiamo *DirectoryIndex index.html* in **DirectoryIndex index.PHP index.html index.htm index.PHP3 index.phtml**.
In poche parole abbiamo aggiunto, separati da spazi, i nomi delle pagine che Apache deve cercare in automatico nella directory principale. È giunto il momento di vedere se tutto è a posto. Per verificarlo apriamo il Notepad e scriviamo semplicemente: <? PHPinfo(); ?>
Attenzione alle estensioni nascoste; se non stiamo attenti salveremo il file come index.PHP.txt e non sarà interpretato dal server.
Salviamo il documento nella cartella *test* con il nome di **index.PHP**; avviamo il nostro web server e digitiamo http://localhost (oppure http://pippo, oppure ancora http://127.0.0.1).

MySQL (www.mysql.com)

MySQL è un RDBMS (*Relational DataBase Management System*) open source. I database sono schedari. In altre parole, un database ci aiuta a tenere ordinati i nostri dati e poterli raggiungere in ogni momento e da qualsiasi parte. I database naturalmente non sono utilizzati solo in ambito web, ma oggi in ogni azienda troviamo svariati database per organizzare fatture, ordini, magazzino.
Per il nostro scopo (costruire pagine web dinamiche) i database ci aiutano ad organizzare i tanti dati che possiamo raccogliere (si pensi ad una mailing list o un forum) oppure per ordinare quello che vogliamo esporre (si pensi ad un sito di e-commerce).
Purché PHP e MySQL?
MySQL è un database molto veloce e professionale. Consente il salvataggio di grandi quantità di dati e l'accesso contemporaneo di molti utenti. Il PHP contiene al suo interno numerose funzioni per la connessione dei database MySQL e per questo motivo che quest'accoppiata sta avendo un successo enorme. Molto spesso si confonde SQL con MySQL. SQL non è una tipologia di database, ma il linguaggio utilizzato per connettersi ad essi. Anche MySQL utilizza SQL per dialogare con il resto del mondo. Mentre MySQL rappresenta vari schedari in cui sono presenti i dati, SQL è il veicolo con il quali questi dati sono messi nelle nostre mani.
L'installazione e l'esecuzione di MySQL nei sistemi Win32 è semplificata perché l'eseguibile da scaricare da www.mysql.com si occupa d'installare automaticamente i file. Terminata l'installazione (la cartella d'installazione di default è c:/MySQL) per avviare MySQL dovremo accedere alla sottodirectory c:/MySQL/bin attraverso DOS e digitare quanto segue:
start MySQLd-opt --skip-name-resolve --skip-grant-tables --language=italian [invio]
Da questo momento MySQL è in esecuzione in background nel sistema, per terminare l'esecuzione digitare quanto segue sempre da DOS e sempre all'interno della cartella bin:
MySQLadmin -u root shutdown
Apriamo nuovamente il file **PHP.INI**. Spostiamoci verso la fine del documento e cerchiamo la sezione che riguarda MySQL basterà individuare il blocco di testo che inizia così:
[MySQL]
; Allow or prevent persistent links
Modifichiamo le seguenti stringhe in modo da impostare questi valori:
mysql.allow_persistent = On
mysql.max_persistent = -1
mysql.max_links = -1
mysql.default_port = 3306
mysql.default_host = localhost
mysql.default_user = (potete lasciarlo vuoto)
mysql.default_password = (potete lasciarlo vuoto)
Fatto questo possiamo finalmente installare il database. La procedura è molto semplice: clicchiamo sull'icona "**Setup.exe**" e seguiamo le varie fasi del wizard; il consiglio è di cambiare la directory di installazione da "*C:\mysql*" in "*C:\Programmi\mysql*" così facendo avremo tutti i nostri componenti nella stessa cartella.
Al contrario di ciò che accade normalmente, MySQL non comparirà direttamente nel menu "start/Tutti i programmi"; per avviare il database dovremo aprire la cartella "**mysql**" situata nella directory "**Programmi**". Tutto ciò che a noi interessa si trova nella cartella "**bin**"; più in dettaglio lavoreremo principalmente con un solo file .EXE: **winmysqladmin.exe**
Creare un collegamento sul Desktop per sveltire le operazioni di attivazione del database. Fatto questo possiamo provare ad avviarlo: clicchiamo due volte su "wynmysqladmin.exe"; dato che è la prima volta che lo lanciamo ci sarà chiesto di immettere *Username* e *Password*; riempiamo tutti e due i campi e diamo l'OK (queste saranno i vostri dati di accesso al database . Ad esempio si può usare **pippo**.
Se tutto è andato per il verso giusto dovrebbe comparire un piccolo semaforo nella System Tray; naturalmente il semaforo dovrebbe avere la luce verde accesa viceversa un bel

semaforo rosso starebbe ad indicare che c'è stato qualche intoppo.
Clicchiamo con il mouse sull'icona del semaforo e selezioniamo **Show Me** in modo da far aprire la finestra del programma con i vari menù a scelta. Il più importante è, forse, il menù "**Databases**", ovvero la lista di tutti i *db* presenti e la relativa struttura.
Per chiudere MySQL premere sulla "X"; c'è un modo più dolce per dire al nostro software che non vogliamo più utilizzarlo: una volta che la schermata è aperta clicchiamo con il tasto destro vicino al semaforo e potremo scegliere tra le due azioni principali (selezioneremo il rispettivo sottomenù in base al nostro sistema operativo):
ShutDown this Tool
Ferma e chiude completamente l'applicazione
ShutDown this Server
Ferma momentaneamente l'applicazione e la mantiene in standby pronta per ripartire con l'opzione "**Start the Server**"

PHPMyAdmin (phpmyadmin.sourceforge.net/)

Non è altro che un'interfaccia grafica, visualizzabile tramite browser, che ci permette di interagire con MySQL. Possiamo creare, editare, cancellare e modificare sia le tabelle che i database. Scompattiamo lo ZIP e rinominiamo la cartella **PHPMyAdmin**. Copiamo questa cartella nella directory principale del nostro web server; in questo caso, quindi, dovremo andare nella cartella "**test**" all'interno di Apache. Adesso possiamo iniziare a fare le modifiche; il file su cui concentreremo la nostra attenzione è il "**config.inc.php**". Apriamolo con il Notepad, vi starete sicuramente chiedendo come mai ci sono tutti quei quadratini neri il file è praticamente illeggibile. Facciamo un breve excursus: il Notepad, come sappiamo, è un'applicazione per documenti di testo ma c'è da aggiungere che il file che abbiamo aperto è stato compilato su un sistema operativo diverso da Windows probabilmente UNIX o derivati di esso; ecco quindi il perché di quei caratteri strani stiamo facendo eseguire a Notepad un file compilato su un'altra piattaforma. Il problema si risolve facilmente e velocemente: apriamo un qualsiasi altro editor di testo (Word o Wordpad); copiamo il testo illeggibile dal Notepad e incolliamolo nel nuovo documento come per magia avremo il testo "in chiaro" e pronto per essere modificato; adesso facciamo l'operazione inversa e riportiamo il tutto nel blocco note.

Nel rincollare il testo "pulito" nel Notepad è molto probabile che rimangano uno o due spazi tra la fine del testo stesso e il fondo del documento. Questo non deve assolutamente accadere in caso contrario avremo un bell'errore in PHPMyadmin; detto in parole povere tra la riga

set_magic_quotes_runtime(0);
?>

e la fine della pagina NON ci deve essere nessuna riga vuota.

Eravamo rimasti con il file "**config.inc.php**" aperto; cerchiamo queste righe e accertiamoci che siano impostate così:

```
$cfgServers[1]['host'] = 'localhost';
$cfgServers[1]['port'] = '';
$cfgServers[1]['adv_auth'] = FALSE;
$cfgServers[1]['stduser'] = '';
$cfgServers[1]['stdpass'] = '';
$cfgServers[1]['user'] = 'pippo';
$cfgServers[1]['password'] = 'pippo';
$cfgServers[1]['only_db'] = '';
$cfgServers[1]['verbose'] = '';
$cfgServers[1]['bookmarkdb'] = '';
$cfgServers[1]['bookmarktable'] = '';
```

I campi "**cfgServers[1]['user']**" e "**$cfgServers[1]['password']**" devono essere settati con gli stessi valori utilizzati nel primo avvio di MySQL. Questa appena descritta è solamente una delle tante procedure possibili; ad esempio possiamo impostare più utenti per utilizzare il database, oppure si può impostare un maggiore livello di sicurezza prima di avere accesso a MySQL. Non dimentichiamoci comunque ciò che stavamo per fare: controllare che PHPMyAdmin funzioni. Se ricordiamo bene abbiamo messo la cartella dello script all'interno della directory di Apache; a questo punto assicuriamoci che il web server e il database siano attivi e apriamo il browser; digitiamo **http://localhost/PHPMyAdmin** e dovremmo trovarci davanti ad una pagina divisa in due frames: a sinistra il nome di due database impostati di default (mysql e test) mentre a destra i vari comandi che ci permettono di interagire per creare, modificare o cancellare tabelle. Il database **MySQL** non va assolutamente cancellato o modificato contiene, infatti, alcuni dati per l'installazione del database stesso. Per le prove abbiamo a disposizione il db **Test** o, ancora più semplicemente, possiamo crearne di nuovi.

EASYPHP (WWW.EASYPHP.ORG)

Installare
1. Scarica EasyPHP.
2. Fai doppio click sul file scaricato.
3. Seleziona la cartella di destinazione e segui le istruzioni.

Lanciare
Non si può propriamente parlare di "lanciare EasyPHP", il fatto è che sono avviati Apache e MySQL server. Dopo l'installazione, sarà stato creato un collegamento nella cartelle "start/Tutti i programmi/EasyPHP". La prima volta che si avvia EasyPHP sarà aggiunta un'icona nella System Tray, vicino all'orologio. Clicca su di essa per accedere ai menu.
1. Log file: riporta ogni errore generato da Apache e MySQL.
2. Configuration : una semplice interfaccia per configurare EasyPHP.
3. Web local : apre l'URL http://localhost/.
4. Start/Stop : avvia/ferma i server Apache e MySQL.
5. Quit.

Uso della cartella www
Per fare in modo che il tuo script possa essere eseguito, devi assicurarti di aver posizionato il tuo file della cartella "www". Il server Apache è configurato automaticamente per aprire un file indice (index) quando è digitato l'indirizzo "http://localhost/". Questa è per definizione la pagina di partenza e verifica che EasyPHP è in esecuzione. È consigliabile creare una cartella per ciascun progetto all'interno della cartella "www"; in questo modo diviene più semplice gestire diversi progetti.

La mia prima pagina in PHP
Questo esempio mostrerà la data corrente. Il codice PHP può essere inserito all'interno del codice HTML cominciando a racchiuderlo tra <? (o <?php) e concludendo con ?>.
Visualizza la data corrente.

```
<html>
<head>
<title>La mia prima pagina in PHP</title>
</head>
<body>
Data Corrente : <? print (Date("l F d, Y")); ?>
</body>
</html>
```

3. Salvare la pagina.

Create una nuova directory all'interno di "www" oppure utilizzate la cartella creata durante l'installazione : "projet1". Salvare la pagina contenente lo script PHP in una delle seguenti estensioni: .php, .php3, .php4. Queste estensioni sono state settate da EasyPHP. Le estensioni possono cambiare con la società che vi offre l'hosting: ricordatevi di cambiare le estensioni se necessario. Per questo esempio utilizziamo l'estensione .php: "data.php"
Azioni da evitare: cercare di lanciare lo script facendo doppio clic sul file all'interno della cartella del progetto: questo genera una pagina di errore.
Azioni corrette: lanciare EasyPHP, connettersi tramite il vostro browser a "http://localhost", aprire la cartella del progetto, quindi diccate su "data.php". A questo punto dovreste vedere la pagina correttamente interpretata dal server Apache.

Installazione

Iniziamo con lo scaricare i sorgenti dai rispettivi siti (www.apache.org , www.php.net, www.mysql.com).
Avremo dunque i sue file php-4.0.4pl1.tar.gz e apache_1.3.20.tar.gz.
Questi pacchetti contengono i file sorgente dei due programmi, prima del loro utilizzo dobbiamo compilare i vari file, assicuratevi di aver installato il compilatore del C/C++.
Scompattiamo i due file nella nostra directory personale (ad esempio in /root): gunzip file php-4.0.4pl1.tar.gz; tar xvf php-4.0.4pl1.tar.gz; stessa cosa per Apache: gunzip apache_1.3.20.tar.gz; tar xvf apache_1.3.20.tar.gz.
Dopo queste semplici operazioni avremo due nuove cartelle, apache_1.3.20 e php-4.0.4pl1. Prima di cominciare la compilazione e l'installazione vera e propria alcune premesse sono doverose.
Per installare questi due pacchetti dovete avere i permessi di amministratore (ossia accedere al sistema come "root").
Installeremo il php direttamente all'interno dell'Apache e non come modulo.
Se avete un PC datato abbiate un pochino di pazienza e non interrompete mai la compilazione una volta iniziata.
Tutte queste operazioni devono essere eseguite dalla riga di comando.
Cominciamo: entriamo nella cartella dell'apache e scriviamo.
./configure -prefix="path/in/cui/vogliamo/installare/apache" [invio]
Attendiamo che il file di configurazione crei il Makefile.
Andiamo ora alla cartella del php e scriviamo.
./configure -with-apache="path-dei-sorgenti-di-apache" -enable-track-vars-prefix=path/in/cui/vogliamo/installare/php [invio]
Quando il makefile è stato creato scriviamo.
make [invio];
e
make install [invio];
Se l'operazione di compilazione non da nessun errore dovreste aver installato con successo il PHP.
Ora occorre tornare nella cartella dei sorgenti di Apache per ultimare la compilazione del server web, anche qui scriviamo.
./configure -enable-track-vars [invio]
dopo aver riconfigurato Apache possiamo passare alla sua compilazione ed installazione.
make [invio]
make install [invio]
Alla fine della compilazione dovreste avere un messaggio di successo.
Fate partire Apache entrando nella sua sottocartella "bin" e digitando.
./apachectl start
Vi consiglio di copiare questo file (apachectl) nella vostra directory "/bin" in modo da poter far partire l'apache da qualsiasi posizione all'intemo di Linux.

Configuriamo PHP attraverso il php.ini

Attraverso questo file è possibile personalizzare alcune importanti impostazioni o abilitare/disabilitare molte funzioni del PHP.
La prima voce che troviamo è "Language Options". Altra parte molto interessante è quell'intitolata "Resource Limits". Nella prima voce possiamo settare il numero di secondi per l'expired di uno script (30 secondi di default), nella seconda voce potete scegliere la quantità di memoria RAM da riservare al PHP. Nella parte successiva, "Error Handing and

logging", potete personalizzare i messaggi di errore in caso di sbagli nel codice.
Nella sezione "Paths e Directories" ci sono varie voci molto interessanti da personalizzare.
doc_roor è la cartella che contiene le pagine in php.
extension_dir è la directory in cui si trovano le estensioni per altri servizi del PHP.
Nella sezione "File Upload" potete settare tutti i valori per l'upload dei file direttamente dalle pagine web: File Upload à Settato su "ON" permette l'upload dei file , diversamente per vietare questa possibilità scrivere "OFF".
Upload_tmp_dir setta la cartella in cui riversare i file che arrivano dall'esterno, lasciate questo valore vuoto se volete settare all'interno dello script il nome di questa cartella.
Upload_max_filesize setta la grandezza massima dei file permessa nell'upload in mega.
Per tutti coloro che hanno installato PHP sotto Win32 nella sezione "Windows Extension" si possono abilitare/disabilitare i file DLL che regolano alcuni servizi. Troverete all'interno dell'elenco i file per abilitare l'IMAP, la manipolazione/ creazione dei PDF, la manipolazione delle GIF e vari altri servizi. Per abilitare il servizio occorre cancellare il punto e virgola iniziale, per disabilitarlo aggiungere il punto e virgola all'inizio. Nel php.ini sono presenti vari altri parametri: gestione della posta; direttive per MySQL; direttive per msql; gestione dei cookies; gestione dei log.

MySQL

In Linux, una volta scompattato il pacchetto nella vostra directory (assicuratevi sempre di essere amministratori di sistema) non vi resta che far partire lo script che in automatico genera il make file:
./configure[invio]
se volete installare MySQL in una directory diversa da quella di default potete scrivere:
./configure --prefix=[percorso e nome della directory in cui volete installare MySQL]
Quando lo script finisce la configurazione automatica potete installare i file con:
make [invio]
e
make install [invio]
Una volta installato, inizializzate il database creando delle tabelle di sistema: spostatevi nella directory in cui avete installato MySQL, accedete alla sottodirectory /bin ed eseguite:
./mysqk_install_db
Infine non vi resta che far partire il demone del MySQL digitando:
./safe_MySQLd &
Il simbolo & serve per far eseguire il demone in background.
Se non ricevete errori durante l'avvio del demone MySQL è stato installato ed avviato con successo nella vostra macchina. Non ci rimane che abilitare le funzioni nel PHP per il MySQL: dobbiamo ricompilare PHP.
Andiamo nella directory dove si trovano i sorgenti del PHP e riconfiguriamo l'engine in questo modo:
./configure --prefix-use-MySQL=[directory in cui avete installato MySQL] [invio]
rinstalliamo PHP scrivendo:
make [invio]
e
make install [invio]

PHP

INTRODUZIONE

Fino a qualche anno fa il web era formato da un certo numero di pagine statiche, ossia HTML puro, incapaci di aggiornare automaticamente i propri contenuti o consentire al visitatore d'interagire con la pagina stessa. L'evoluzione del web, culminata nell'introduzione della dinamicità della pagina, ha avuto come protagonisti i linguaggi cosiddetti "**server-side**" che hanno dato quel qualcosa che ancora mancava in Internet. Il PHP dunque è un linguaggio di programmazione (definito anche linguaggio di scripting) utilizzato per lo sviluppo di pagine web dinamiche, uno dei più recenti in questo campo. Il PHP nasce a metà del 1994 e da allora il suo utilizzo è andato via via aumentando. Grazie al PHP (e a tutti gli altri linguaggi "server-side") è possibile consentire agli utenti di interagire con un ampio database, farli muovere in un negozio virtuale, prenotare online un biglietto aereo e così via.
È un linguaggio di programmazione server-side che significa.

1. *Personal Home Page*.
2. *Hypertext PreProcessor*.

I linguaggi server-side (script GGI, ASP) elaborano le istruzioni sul web server e, a differenza dei linguaggi di scripting lato client (Javascript, applet Java), non dipendono dal browser (**user-agent**) che li richiama.
Il linguaggio PHP, per poter funzionare, ha bisogno di un web server su cui far "girare" il PHP; un web server non è altro che un software capace di gestire tutte le connessioni ad un determinato PC ed inoltre si fa carico di "restituire", al visitatore, la pagina PHP richiesta in formato HTML così che possa essere "letta" dal browser.

FUNZIONAMENTO DEL PHP: LATO CLIENT, LATO SERVER

La differenza tra lato client e lato server sta tutta nel modo e da chi è interpretata una pagina web quando essa è caricata. Quando un server web predisposto per il PHP riceve una richiesta dal browser di un client iniziano una serie di operazioni.
Il server:

1. Legge ed individua la pagina sul server.
2. Esegue le istruzioni PHP contenute all'interno della pagina ed esegue tutti i comandi PHP.
3. Rimanda la pagina risultante al browser.

Un esempio pratico potrebbe essere quello di una pagina in PHP che si prefigge di leggere una riga di un qualsiasi database, il server web esegue ed ottiene la riga dal database ed invia il tutto al browser del client generando codice HTML. Per questo motivo nella pagina risultante non si può vedere nessuna traccia del codice PHP, esso è stato già interpretato e "trasformato" in HTML dall'interprete (il PHP, a differenza di linguaggi come il C o il C++ è un linguaggio interpretato e non compilato).
Come fa il server a capire quando una pagina contiene del codice PHP?
Molto semplice, dal formato della pagina. Ogni pagina che contiene PHP deve avere un formato opportuno (.php o .php3 o .php4).
Il PHP è un linguaggio molto semplice da utilizzare, a cominciare dalla sintassi derivata direttamente da veri linguaggi di programmazione come C/C++, PERL, Python, Java. Nonostante esso sia un linguaggio interpretato può vantare prestazioni notevoli ulteriormente migliorate nella versione 4. Il PHP è un linguaggio molto flessibile che ci consente di fare davvero tutto, dalla creazione d'immagini alla manipolazione e creazione di documenti in PDF, dalla gestione dei cookies all'elaborazione dei form HTML e il supporto di molteplici tipologie di database. Forse la vera forza del PHP sta senz'altro nella gestione dei database, con poche righe di codice è possibile accedere qualsiasi

database, estrapolare i dati che c'interessano e inserirli nella pagina web.
Un altro punto a favore del PHP è la sua natura **OpenSource**, quindi gratuita. Infine possiamo far rilevare l'alta portabilità del PHP; esso gira su tutti in principali server web ed in linea di massima non dobbiamo apportare nessuna modifica al codice quando lo spostiamo da un server web ad un altro (la differenza è tra le piattaforme Microsoft e quelle Linux/Unix).

SINTASSI GENERALE DEL LINGUAGGIO

Affinché l'interprete PHP riesca a distinguere all'interno del codice il linguaggio da interpretare ed eseguire dall'HTML occorre utilizzare dei TAG particolari.

```
<table>
<tr>
<td>
<?php print " Ciao, mondo!"; ?>
</td>
</tr>
</table>
```

Queste righe di codice stamperanno in una tabella la parola "Ciao, mondo!". Si nota immediatamente che non vi è nessuna traccia del codice originario! In altri termini, il client non ha alcun modo per risalire alle istruzioni PHP che hanno generato la pagina richiesta. Possiamo distinguere il codice PHP delimitato dai tag <?php e ?>. L'interprete PHP sa che tutto ciò che si trova all'interno di questi delimitatori deve essere interpretato ed eseguito. Tutto ciò che si trova al di fuori dei tag PHP è normalmente eseguito dal browser. Per tutti coloro che intendono avvicinarsi al mondo del PHP è in ogni caso indispensabile un'ottima conoscenza dell'HTML perché, una volta che lo script è interpretato, il PHP restituisce semplice codice HTML (**HTML - embedded**).
Esiste anche la possibilità di utilizzare altri tag per distinguere il codice dall'HTML.

```
<? …… ?>
```

Questa sintassi è molto simile alla precedente, ma il suo uso deve essere abilitato all'interno del PHP3.INI.
Nella sezione "Language Option" dovete modificare il parametro "short_open_tag" ed inserire "On".

```
<Script language="php" ……</script>
```

Questi tag sono attivi di default e possono risultare molto utili in quegli editor HTML visuali che non conoscono le estensioni PHP; per esempio Front Page.

```
<% ……. %>
```

Questa è la sintassi stile Microsoft ASP e deve essere attivata per poter essere utilizzata. Sempre nel PHP3.INI nella sezione "Language Option" dovete modificare il parametro "asp_tags" su "On". Altra regola fondamentale è il segno di fine comando composto dal ";". Uno degli errori più diffusi per tutti coloro che si avvicinano per la prima volta al PHP è la dimenticanza del segno di fine comando che, generando un errore, non fa concludere l'esecuzione della pagina. Anche in presenza di errori di questo tipo il PHP ci viene incontro fornendoci un determinato messaggio di errore. Se trovate un "parser error" molto probabilmente avete fatto un errore di sintassi come quello della dimenticanza del ";" alla fine dell'istruzione.

COMMENTI

I commenti rendono il codice più leggibile e modificabile, anche dopo tempo. I commenti non sono eseguiti dall'interprete quindi una riga in più non cambierà la velocità di esecuzione dello script e nello stesso tempo vi aiuta a seguire il codice passo passo in tutte le operazioni che eseguite. Il PHP supporta vari tipi di commenti.
Quelli in stile C++: *//*
Commento in PHP. *i*
Commenti in stile C: */**

Commento in PHP: */
Commenti in stile PERL: #

COSTANTI

Ad un nome è associato un valore che il programmatore può utilizzare, ma che non può modificare.
__FILE__
Il nome dello script che è sottoposto al parsing.
__LINE__
La linea dove è presente questo statement.
PHP_VERSION
La versione di PHP in uso.
PHP_OS
Il nome del sistema operativo su cui è fatto girare lo script.
TRUE
Un valore sempre vero.
FALSE
Un valore sempre falso.
E_ERROR
Un errore, differente dal solito errore di parsing (ossia, scrittura del codice), non recuperabile.
E_WARNING
Come sopra, ma in questo caso il PHP può continuare l'esecuzione del codice.
E_PARSE
Un errore del parser (ossia di sintassi del file).
E_NOTICE.
Qualcosa che potrebbe essere come non essere un errore; l'esecuzione non è interrotta.
Per esempio.
<? echo "Linea ", __LINE__ . " del file " ,__FILE__; ?>
Restituirà: Linea XX del file NOMEFILE.
È da aggiungere inoltre che, se queste sono le costanti "di default" del linguaggio, è possibile definire all'interno di uno script le proprie costanti in maniera molto semplice.

VARIABILI

Le variabili possono essere viste come dei contenitori di dati (numeri o lettere) che sono memorizzate all'interno di uno script per essere poi riprese durante l'esecuzione dello stesso. Ad ogni variabile, individuata con il suo nome, corrisponde un certo valore attribuito da noi stessi. Quindi la riga:
$a = 5;
ci mostra la variabile *a* il cui contenuto è il numero 5.
Le variabili in PHP sono indicate con il segno "$" prima del nome.

```
<?php
// Programma per la somma di due numeri
$a = 5;
$b = 5;
$c = $a + $b;
print $c;
?>
```

Il risultato di questo script sarà 10.
Nelle prime due righe definisco rispettivamente che il contenuto delle variabili $a e $b sia cinque, nella terza riga definisco che il valore della variabile $c sia uguale alla somma di $a e $b ed infine stampo a video il contenuto di quest'ultima.
L'uso delle variabili è molto semplice grazie anche ad alcune particolarità del PHP che ad esempio non ci obbliga a dichiarare preventivamente le variabili che intendiamo utilizzare

all'interno del programma, ma quest'ultima avviene in contemporanea con l'assegnazione dei valori alla variabile stessa (**dichiarazione implicita**).
A disposizione del programmatore c'è anche un certo numero di variabili predefinite, in pratica di variabili il cui valore è già impostato. Queste variabili possono essere sommariamente divise in tre gruppi: le variabili di Apache (o del server), le variabili d'ambiente e le variabili PHP. Premettendo che molte di queste variano da macchina a macchina e da server a server, si possono leggere con una semplice pagina del tipo.

```
<html><body>
<? phpinfo(); ?>
</body></html>
```

Variabili di Apache

È possibile leggere il valore di ogni variabile o con la funzione phpinfo() oppure con un semplice.

```
echo "NOME_DELLA_VARIABILE";
```

Inoltre, i valori di esempio sono tratti o direttamente dalla documentazione del PHP o dal valore che esse assumono nel server utilizzato per scrivere queste pagine.
GATEWAY_INTERFACE: la versione delle specifiche CGI utilizzate dal server, ad esempio "CGI/1.1".
SERVER_NAME: il nome del server, quello che in Apache si definisce in "ServerName" in httpd.conf, ad esempio "myhost.com".
SERVER_SOFTWARE: il nome del software utilizzato per il webserver, ad esempio "Apache/1.3.9 (Unix) Debian/GNU PHP/3.0.15 mod_perl/1.21_03-dev".
SERVER_PROTOCOL: il nome e la versione del protocollo tramite il quale è stata richiesta la pagina, ad esempio "HTTP/1.0".
REQUEST_METHOD: utilizzato nelle form, può essere "GET", "POST", "HEAD", "PUT".
QUERY_STRING: se presente, la stringa tramite la quale la pagina è stata richiesta.
DOCUMENT_ROOT: la DocumentRoot del server, come configurata in httpd.conf, ad esempio "/var/www".
HTTP_ACCEPT: il contenuto dell'header Accept, ad esempio "text/*, image/*, audio/*, application/*".
HTTP_ACCEPT_CHARSET: il charset accettato, ad esempio "iso-8859-1".
HTTP_ENCODING: l'encoding della richiesta, se presente; ad esempio, "gzip".
HTTP_ACCEPT_LANGUAGE: il linguaggio, ad esempio "en".
HTTP_CONNECTION: il contenuto dell'header Connection, ad esempio "Keep-alive".
HTTP_HOST: il nome dell'host, ad esempio "localhost".
HTTP_REFERER: il nome della pagina dalla quale si proviene.
HTTP_USER_AGENT: il contenuto dell'header User_Agent, ad esempio "Mozilla/4.72 [en] (X11; I; Linux 2.2.14 i586)".
REMOTE_ADDR: l'indirizzo IP dell'utente connesso alla nostra pagina.
REMOTE_PORT: come sopra, ma riferito alla porta; ad esempio, le due variabili potrebbero avere un output del tipo: "127.0.0.1 1275".
SCRIPT_FILENAME: il nome dello script richiesto al server, ad esempio "prova.php3".
SERVER_ADMIN: il nome dell'amministratore di sistema.
SERVER_PORT: la porta sulla quale il server è in ascolto, ad esempio la porta 80.
SERVER_SIGNATURE: l'eventuale "signature" del server.
PATH_TRANSLATED: il percorso per lo script richiamato, ad esempio "/var/www/php/prova.php3".
SCRIPT_NAME: il path, a partire dalla DocumentRoot, dello script; ad esempio, /php/prova.php3".
REQUEST_URI: l'URI richiesto per accedere alla pagina.

Variabili d'ambiente

Dipendendo dalla shell utilizzata. Per leggerle, richiamate all'interno di uno script PHP.

```
echo "Il mio path è $PATH";
```

che visualizzerà il path sul sistema.

Variabili PHP
argv: un array contenente i parametri passati allo script.
argc: come sopra, ma se lo script è eseguito dalla linea di comando.
PHP_SELF: il nome dello script attualmente in esecuzione;
HTTP_COOKIE_VARS: un array associativo contenente le variabili passate allo script tramite i cookies http.
HTTP_GET_VARS: un array associativo contenente le variabili passate allo script tramite una richiesta GET.
HTTP_POST_VARS: come sopra, ma riferito al metodo POST.
È da ricordare che le ultime tra variabili hanno senso solamente se, nel file di configurazione (track_vars=On) oppure all'interno dello script con la direttiva "php_track_vars", è stato abilitato il tracking delle variabili.
Oltre a contenere numeri una variabile può benissimo contenere caratteri e frasi.

```
<?php
// Programma per la stampa di una stringa
$a = “Ciao Mondo”;
print $a;
?>
```

Questo script stamperà la frase “Ciao Mondo”.
Variabili dinamiche
Il PHP permette vari utilizzi per quanto riguarda le variabili dinamiche. Partiamo dall'assegnazione tradizionale di una variabile.

```
$variabile = "Ciao";
```

Poniamo ora il caso di avere l'esigenza di creare una nuova variabile che abbia come nome il valore della variabile sopra assegnata ($variabile).
È possibile fare questo utilizzando le variabili dinamiche, scriveremo.

```
$$variabile = "mondo";
```

Con quest’operazione il PHP esegue diverse operazioni: interpreta $$variabile partendo dalla parte interna dell'espressione ($variabile) ed in questo modo crea una nuova variabile il cui nome è uguale a "Ciao" ed il valore è uguale a "Mondo".
Nel PHP si possono annidare le variabili all'interno di altre variabili fino a livelli infiniti, ma è vivamente consigliato non spingersi oltre il secondo livello per non rendere il codice illeggibile.

TIPI DI DATO

Scalari
Integer
Possono assumere diversi valori numerici esprimibili in differenti notazioni.

```
$a = 18; # decimale
$a = -18; # decimale negativo
$a = 022; # notazione ottale; equivalente a 18 decimale
$a = 0x12; # notazione esadecimale, equivalente a 18 decimale
```

Floating point
Sono numeri in virgola mobile, ad esempio 9.876.

```
$a = 9.876;
$a = 9.87e6;
// $pi_greco è un numero in virgola mobile si noti il punto (virgola decimale)
$pi_greco = 3.14;
```

Boolean

```
// $b è una variabile di tipo boolean
$b = TRUE;
```

Strutturati
Array
Strings

La sintassi di base per le stringhe.
$string = "Io sono una stringa";
Se sono utilizzate le virgolette (""), il contenuto della stringa è espanso (o, tecnicamente, "interpolato"), come nell'esempio successivo.
$num = 10;
$string = "Il numero è $num";
che visualizzerà "Il numero è 10".
Come in tutti i linguaggi, comunque, anche con il PHP ci sono i caratteri speciali che vanno fatti precedere da un simbolo di escape.
$num = 10;
$string = "Il numero è "$num"";
Chi pensa che l'output di tale codice sia 'Il numero è "10"' si sbaglia: a parte il fatto che, così come è scritto, lo script darebbe un errore di compilazione, le virgolette sono caratteri speciali, ma non per questo non è permesso utilizzarle; la sintassi corretta per il comando riportato sopra è questa.
$num = 10;
$string = "Il numero è \"$num\"";
Altri caratteri speciali sono i seguenti.
\n -> newline
\r -> carriage return
\t -> tabulazione
\\ -> backslash
\$ -> simbolo del dollaro
L'alternativa ai caratteri di escape, quando non ci siano contenuti da espandere, sono gli apici (')
$string = '$ è il simbolo del dollaro';
visualizzerà proprio ciò che è contenuto fra gli apici. Attenzione a non cadere nel più consueto degli errori.
$num = 10;
$string = 'Il numero è $num';
che non visualizzerà "Il numero è 10" bensì "Il numero è $num"! Quindi possiamo affermare che, con gli apici, il contenuto della string è riportato letteralmente, ossia com'è effettivamente scritto al suo interno.

Object

class visualizza {
function esegui_visualizza () {
echo "Visualizza un messaggio"; }
}
$obj=new visualizza;
$obj->esegui_visualizza();
Iniziamo definendo la classe "visualizza", che contiene la funzione "esegui_visualizza" che non fa altro che visualizzare un semplice messaggio a video. Con lo statement "new" inizializziamo l'oggetto "$obj" e richiamiamo la funzione visualizza con l'operatore -> su $obj.

OPERATORI

Gli operatori sono dei segni che ci permettono di svolgere le principali funzioni all'interno dei programmi PHP. Per esempio, il segno "=" ci permette di attribuire un valore ad una stringa come abbiamo visto negli esempi precedenti, (operatore di assegnazione) oppure il segno "+" ci permette di sommare due o più numeri. Nel PHP esistono vari tipi di operatori.

Aritmetici

Sono gli operatori più semplici che ci permettono di svolgere le operazioni matematiche all'interno degli script.
$a + $b // (+) La somma di $a e $b

```
$a - $b      // ( - ) La sottrazione fra $a e $b
$a * $b      // ( * ) Il prodotto di $a e $b
$a / $b      // ( / ) Il rapporto di $a e $b
$a % $b      // ( % ) Il resto della divisione di $a e $b
```

Fermiamoci per un attimo sugli ultimi due: il risultato della divisione non è approssimato all'intero più vicino, ma riporta tutto il numero risultante; il numero dei caratteri dopo il punto da considerare è definito nel file php3.ini alla riga.

```
precision = 14
```

Quindi, saranno riportati "solo" 14 numeri dopo la virgola, a patto che ci siano.

```
$a = 12;
$b = 5;
$c = $a / $b;
echo $c;
```

il risultato è 2.4 e sarà visualizzato proprio come 2.4, non come 2.40000000000000! Quindi, i 14 decimali sono visualizzati solamente se esistono, e non indiscriminatamente come inutili zeri. Il resto della divisione è riportato da solo, senza il risultato della divisione stessa: se nell'esempio precedente avessimo utilizzato "%" al posto di "/", il risultato sarebbe stato "2".

Relazionali

Gli operatori di confronto sono quelli che consentono di mettere in relazione tra loro due o più espressioni.

```
$a == $b      //Confronta l'uguaglianza del valore tra $a e $b (stesso valore)
$a != $b      //$a e $b sono diversi;
$a < $b       //$a è minore di $b;
$a <= $b      //$a è minore o uguale di $b;
$a > $b       //$a è maggiore di $b;
$a => $b      //$a è maggiore o uguale di $b;
$a ? $b : $c  //questo, da utilizzarsi più con le espressioni che con le variabili, valuta $b se
              //$a è vera, e valuta $c se $a è falsa;
++$a  //incrementa di uno $a e la restituisce;
$a++  //restituisce $a e la incrementa di uno
--$a
$a--//come i due precedenti, ma il valore è decrementato;
```

Logici

```
$a & $b      //operatore "And" ($a e $b);
$a && $b     //come sopra, ma con una precedenza più alta;
$a | $b      //operatore "Or" ($a oppure $b);
$a || $b     //come sopra, ma con una precedenza più alta;
$a ^ $b      //operatore "Xor" ($a oppure $b ma non entrambi);
!$a          //operatore "Not" (vero se $a non è vera);
```

ISTRUZIONI

Semplici: non contengono altre istruzioni; per esempio l'assegnazione.
Strutturate: sono composte da più istruzioni.

Sequenza

Le istruzioni sono eseguite nello stesso ordine in cui sono scritte.

Selezione

Unaria

Il costrutto *if* permette di eseguire delle determinate operazioni solo se si verificano determinate condizioni. Poniamo il caso di voler confrontare due variabili, solo se queste due variabili sono uguali stamperemo a video il loro valore.

```
<?php
$a=5; $b=5;
if( $a == $b) {
```

```
                print " Il valore di a è uguale a $a";
                print " Il valore di b è uguale a $b";
                }
?>
```

Vediamo la struttura del costrutto if.

```
if (condizione) {
                blocco-istruzioni da eseguire nel caso la nostra istruzioni risulti vera.
                }
```

Il costrutto if risulta molto semplice a parte qualche nuova regola; come potete notare non bisogna inserire alla fine della parentesi graffa di chiusura il segno di fine istruzioni perché il costrutto if non è una vera istruzione, ma solo uno strumento di controllo alle istruzioni inserite al suo interno.

Ma cosa succede se $a e $b fossero state diverse?

La risposta è alquanto semplice, poiché la condizione non risulta vera il codice compreso nelle parentesi graffe non sarà eseguito.

Binaria

Il costrutto if ci offre delle ottime varianti per controllare ancora più il nostro codice, il costrutto else .

```
<?php
$a=5;
$b=5;
if ( $a == $b) {
                print " I valori $a e $b sono uguali";
} else {
                print " I valori $a e $b sono diversi";
}
?>
```

Il costrutto else quindi ci consente di eseguire un secondo blocco d'istruzioni nel caso la condizione risulti falsa.

Nidificata

Esistono molte altre opportunità con il costrutto if, supponiamo di avere molte condizioni da eseguire una dietro l'altra. Supponiamo di voler stabilire se, sempre date due variabili $a e $b, $a è maggiore, minore o uguale a $b, potremo scrivere in questo modo.

```
If ($a<$b) {
                print "$a è maggiore di $b;
            }
If ($a>$b) {
                print "$a è minore di $b;
           }
If ($a==$b) {
                print "$a è uguale di $b;
            }
```

Questo metodo senz'altro funziona, ma è troppo macchinoso e porterebbe a grosse difficoltà qualora le condizioni risultassero più di tre. In questo caso il PHP ci viene incontro con un'altra variante : il costrutto elseif.

```
If ($a<$b) { print "$a è maggiore di $b";
}
elseif ($a>$b) { print "$a è minore di $b";
} elseif ($a == $b) { print "$a è uguale di $b";
}
```

In questo modo abbiamo utilizzato un nuovo costrutto che ci ha permesso di unire i tre cicli if visti poco sopra.

Il ciclo if è forse uno dei costrutti più utilizzati all'interno di qualsiasi script in PHP. Nel suo uso date particolare attenzione all'apertura e chiusura delle parentesi graffe, molti

programmatori utilizzano un piccolo trucco per evitare questa dimenticanza, aprire e chiudere subito le parentesi e scrivere l'istruzione al loro interno successivamente.

Multipla

Quando si hanno istruzioni mutuamente esclusive.

```
// La schedina!
switch($segno) {
case 1: echo "E' uscito il segno 1!";
break;
case 2: echo "E' uscito il segno 2!";
break;
default: echo "E' uscito il segno X.";
}
```

La sintassi dello switch è la seguente.

```
switch(espressione) {
case valore1: blocco-istruzioni1
case valore2:blocco-istruzioni2
...
}
```

Iterazione

Successione d'istruzioni eseguite ripetutamente.

Il programmatore specifica la condizione che determina la fine dell'iterazione.

Ciclo a condizione iniziale

Molto spesso, in alcune pagine vi è l'esigenza di ripetere l'esecuzione di determinate funzioni fino a quando non si verifica una determinata condizione.

```
While( condizione ) {
                    Istruzioni da eseguire
                    }
```

Traducendo while nel nostro linguaggio potremo definirlo così: finché una determinata condizione non diventa vera esegui le istruzioni comprese all'interno delle parentesi graffe.
Vediamo come possiamo ricavarci la tabellina del due con il costrutto while.

```
<?php
// definisco una variabile di comodo che chiameremo $a che rappresenterà uno dei
// due membri della moltiplicazione (l'altro è il 2)
// poiché la tabellina parte dal numero 1
$a = 1;
// introduciamo il costrutto while
while ($a != 10) {
// voglio sapere la tabellina del 2 fino a 10
/* tradotto sarà così: finché ( while ) $a è diverso ( != ) da 10 ( quindi finché la condizione
tra parentesi risulterà vera ) esegui l'operazione che sarà */
print "( $a * 2 ) ";
// prima di uscire dal costrutto while incrementiamo di un'unità la variabile $a altrimenti
// incorreremo nel più frequente errore nell'uso di questo ciclo, ossia un loop infinito
// che finirà solo dopo il timeout del PHP
// Nel costrutto while, finché $a è diverso da 10 eseguirà sempre l'istruzione
// che avete inserito nelle parentesi graffe, una volta che $a sarà uguale a 10 si
// uscirà dal ciclo e lo script continuerà ad eseguire le istruzioni presenti dopo
$a++;
// Il ++ è un altro operatore del PHP, questo è una forma abbreviata del classico:
// $a = $a + 1;
} ?>
```

L'output di questo semplicissimo programma sarà: 2 4 6 8 10 12 14 16 18 20.
Anche nel costrutto prestate particolare attenzione alla chiusura delle parentesi graffe ed assicuratevi che il ciclo non diventi infinito.

Ciclo a condizione finale
Nella sintassi tradizionale prima è valutata la condizione e solo se vera è eseguito il blocco istruzioni, nella variante do/while invece.
do
{
blocco-istruzioni
}
while (condizione)
è prima eseguito un blocco istruzioni e poi è valutata l'istruzione, per i restanti cicli il funzionamento è identico a quello del costrutto tradizionale.
Il numero d'iterazioni è noto a priori
```
for (condizione1, condizione2, condizione3 ) {
        blocco-istruzioni
}
```
Il costrutto for si differenzia dal costrutto while per via del fatto che il for è utilizzato quando noi conosciamo il numero di ripetizioni da eseguire. Il for ha un funzionamento particolare, le condizioni tra parentesi sono tre e non una come nel ciclo while. Analiticamente può essere spiegato in questo modo: per prima cosa è valutata la condizione uno (una solo volta), generalmente questa prima condizione è un'espressione, da qui inizia l'iterazione vera e propria: è valutata la condizione due, se falsa si esce dal ciclo, se vera si esegue il blocco istruzioni. Al termine di ogni operazione è infine valutata la condizione tre.
Esempio fatto con il while per la tabellina del due.
```
<?php
for ($a = 1; $a < 10; $a++ ) {
print "( $a * 2 ) ";
?>
```

ARRAY
Un array è una variabile che contiene più valori. Ecco un piccolo esempio.
$colori = array ("rosso", " verde", "blu");
Con questa riga di codice abbiamo appena creato un array ($colori), il quale non è altro che una variabile con più valori al suo interno. In PHP esistono due tipi di array: gli array ad indice numerico e gli array associativi. Nel nostro caso l'array $colori è un array ad indice numerico poiché il PHP assegna un numero univoco ai valori dell'array in base al loro inserimento. Per cui.
$colori[0] sarà uguale a "rosso"
$colori[1] sarà uguale a "verde"
$colori[2] sarà uguale a "blu"
e così via se avessimo altri valori all'interno dell'array.
Il primo valore nel PHP è sempre lo 0. Nel PHP l'utilizzo dell'array è molto semplice perché non occorre dichiarare preventivamente quanti elementi saranno contenuti al suo interno, ma anzi è possibile aggiungere elementi anche durante l'esecuzione dello script. Volendo aggiungere un quarto elemento all'interno del nostro array $colori basterebbe scrivere:
$colori[]= "viola";
Il PHP accoderà in automatico il valore viola agli altri presenti nell'array e possiamo visualizzare il suo valore scrivendo:
print $colori[3];
Per spazzolare gli elementi contenuti all'interno di un array ad indice numerico, impiegheremo il costrutto for:
```
<?php
// visualizzeremo i valori dell'array $colori precedentemente creato
$numero_elementi = count($colori);
// la funzione count conta il numero di elementi presenti all'interno di un array
```

```
for ($a = 0; $a< $numero_elementi; $a++) {
print $colori[$a]
// inizialmente $a sarà uguale a 0, quindi visualizzeremo il valore di $colori[0], poi
// $colori[1] e così di seguito finché non arriveremo alla fine dell'array
}
?>
```

L'output di questo script sarà: rosso , verde, blu, viola. Per quanto riguarda invece gli array associativi il discorso si semplifica ulteriormente perché in questo caso siamo noi a scegliere il nome dell'indice dei valori presenti all'interno dell'array. Utilizzando l'esempio dei colori potremo scrivere in questo modo il nostro array associativo.

```
$colori[primo_colore] = "rosso";
$colori[secondo_colore] = "verde";
```

e così via….

In questo caso per visualizzare il secondo colore ci basterà rischiare l'indice che noi stessi abbiamo attribuito al secondo valore, ossia:

```
print $colori[secondo_colore];
```

L'output sarà: verde.

Gli array, man mano che faremo degli script sempre più complessi fungeranno da veri e propri magazzini per i nostri dati. Pensate alla realizzazione di una rubrica personale, invece di definire una variabile per ogni voce della rubrica potremo semplicemente fare un array associativo che le comprenda tutte e dopo maneggiare i dati attraverso gli indici che noi stessi abbiamo attribuito alle varie voci. In PHP un array può contenere vari tipi di dato nello stesso array: lettere, numeri reali, numeri in virgola mobile ecc. Il PHP gestisce le stringhe di caratteri come veri e propri array ad indice numerico; se noi scriviamo:

```
$stringa =" Il mondo è bello ";
```

potremo raggiungere qualsiasi singolo carattere contenuto nella variabile $stringa. Per cui:
$stringa[0] sarà uguale al carattere I
$stringa[1] sarà uguale al carattere l
$stringa[2] sarà uguale al carattere "spazio"
e così via fino alla fine della stringa.

```
// Questo è un array di numeri interi lo creo esplicitamente usando array()
$primi = array( 2, 3, 5, 7, 11 );
// Questo array lo creo implicitamente
$pari[0] = 2;
$pari[1] = 4;
$pari[2] = 6;
// Questa è una tabella di hash
$bookmark["puntoinf"] = "punto-informatico.it"
$bookmark["latoserver"] = "www.latoserver.it"
```

A differenza di quanto avviene in altri linguaggi di programmazione, un array in PHP può contenere elementi di tipo diverso.

```
// Questo è un array valido! Contiene: un numero intero, una stringa, un numero in virgola
//mobile ed un altro array!
$mix = array( 1, "ciao", 3.14, array( 1, 2, 3 ) );
```

Gli array associativi si basano invece su coppie "name-value".

```
$a = array(
"nome" => "Mario",
"cognome" => "Rossi",
"email" => "mario@rossi.com",);
```

è interessante la possibilità della funzione array di annidare le entries, come nell'esempio che segue.

```
$a = array(
"primo" => array(
"nome" => "Mario",
```

"cognome" => "Rossi",
"email" => "mario@rossi.com",),
"secondo" => array(
"nome" => "Marco",
"cognome" => "Verdi",
"email" => "mario@verdi.com",)
);
Eseguire su questo array un comando del tipo.
<? echo $a["secondo"]["email"]; ?>
visualizzerà "mario@verdi.com".

FUNZIONI

La dichiarazione di proprie funzioni consente al programmatore di estendere il linguaggio utilizzato. Una funzione può essere interpretata in due modi.

1. Come funzione matematica: relazione tra un input (i parametri passati) ed un output (il risultato della funzione.
2. Come un meccanismo per aggiungere nuovi comandi al linguaggio: è possibile raggruppare assieme un blocco d'istruzioni PHP che potranno essere richiamate ed eseguite.

Già nel PHP ci sono varie funzioni che possiamo utilizzare per gli scopi più comuni, ad esempio la funzione strlen() calcola quanti caratteri sono presenti in una stringa, la funzione time() calcola l'ora.
Una funzione può essere definita come un comando per effettuare un'operazione: se $a contiene la stringa "Il mondo è bello" la funzione strlen($a) ci fornirà il numero di caratteri della stringa contenuta in $a. Tutti i comandi, una volta impartiti devono restituirci qualcosa e nell'esempio sopra riportato la risposta dello script sarà sedici. Il PHP ci offre delle funzioni di carattere generale, ma nel contempo ci permette di poter creare delle nostre funzioni. Poniamo che per ipotesi si debba fare in continuazione la somma di due numeri e visualizzare il risultato. Nel modo tradizionale dovrei fare in questo modo:

```
<?php
$a = 5;
$b = 5;
$c = $a + $b;
print $c;
?>
```

Certo lo script non è molto lungo, ma provate ad immaginare operazioni più complesse e più laboriose e scoprirete subito il vantaggio di poter creare delle vostre funzioni. La nostra funzione sarà così.

```
function somma ($a,$b) {
$c = $a + $b;
print $c;
}
```

Una volta dichiarata la funzione per eseguire la somma di due numeri e visualizzare il risultato sarà: somma(5,5); l'output del programma sarà dieci.
Per utilizzare la funzione appena dichiarata è sufficiente invocarla usando il nome ad essa attribuito e fornendo gli argomenti previsti.

```
// Uso della funzione `somma'
// Il valore di $risultato è 2
$risultato = somma(1,1);
```

Esaminiamo la sintassi della funzione.

```
function <nome-funzione> ( <argomenti> ) {
<corpo-della-funzione>
}
```

Con la parola function diciamo all'interprete PHP che stiamo creando una nuova funzione,

subito dopo function deve essere definito il nome della funzione (nel nostro caso somma) che sarà poi utilizzato per richiamare la funzione stessa all'interno del nostro programma. Dopo il nome seguono le parentesi tonde (obbligatorie), al loro interno dobbiamo inserire i parametri da passare alla funzione in modo che possa eseguire su di esse le istruzioni che abbiamo inserito tra le parentesi graffe. Attenzione, mentre le parentesi tonde sono obbligatorie, non è obbligatorio inserire delle variabili da passare alla funzione perché magari è la funzione stessa che ha la funzione di creare altre variabili. Prendete ad esempio una funzione per la generazione di un numero random compreso tra dieci e venti, scriveremo.

```
<?php
function random () {
$a = rand(10,20);
return $a;
}
?>
```

Questa funzione non necessita di nessuna variabile esterna, ma l'abbiamo costruita solo per generare a sua volta una variabile. L'istruzione return indica alla funzione di restituire il valore che noi vogliamo che ci restituisca, in questo caso $c ed uscire da essa riprendendo l'esecuzione dal punto dello script in cui era stata invocata. La logica delle funzioni personalizzate stimola la programmazione modulare (ogni funzione è uguale ad un modulo) in cui ogni problema è suddiviso in problemi più piccoli per poi assemblare il tutto.
All'interno di una funzione è possibile definire ed utilizzare variabili locali, che non sono accessibili al di fuori di essa.

```
<?
function prova() {
$numero = 3;
...
}
// Qui $numero non è definita!
?>
```

Le variabili globali, invece, sono definite all'esterno della funzione, nella parte principale dello script. La parola chiave global è utilizzata per indicare che $numero è accessibile anche all'interno della funzione.

```
<?
// $numero è una variabile globale
$numero = 3;
function prova() {
// Quando dico $numero intendo la variabile globale
global $numero;
echo $numero;
}
?>
<?
// $numero è una variabile globale
$numero = 3;
function prova() {
// Ho dimenticato global! questa istruzione non stampa nulla perché $numero è
//considerata locale
echo $numero;
}
?>
```

In alternativa all'uso di global è possibile, dall'interno della funzione, accedere ad una variabile globale utilizzando l'array associativo $GLOBAL. Ad esempio, dall'interno della

funzione, prova si poteva accedere alla variabile globale $numero, senza usare global, nel modo seguente: $GLOBAL [“numero”].
La funzione phpinfo() contiene informazioni sulla versione di PHP installata, sull’ambiente di esecuzione e le variabili predefinite che PHP mette a disposizione del programmatore.

```
<html>
<head><title>phpinfo()</title></head>
<body>
<?php
phpinfo();
?>
</body>
</html>
```

La funzione echo() serve per scrivere (o stampare) l’output che è inviato al browser del visitatore che accede al nostro script.

```
<html>
<head><title>echo</title></head>
<body>
<?php
echo "<h1>Benvenuto!</h1>";
?>
</body>
</html>
```

La pagina inviata al client, dopo l’elaborazione da parte dell’interprete PHP, sarà la seguente.

```
<html>
<head><title>echo</title></head>
<body>
<h1>Benvenuto!</h1>
</body>
</html>
```

Grazie ad echo() è possibile visualizzare il contenuto di variabili; per esempio, $HTTP_HOST è il nome del dominio del sito su cui lo script è eseguito.

```
<html>
<head><title>echo</title></head>
<body>
<?php
echo "<h1>Benvenuto su $HTTP_HOST!</h1>";
?>
</body>
</html>
```

Le funzioni exit() e die() entrambe producono il risultato di arrestare l’esecuzione dello script, con la differenza che die() consente anche di stampare un messaggio.

```
<html>
<head><title>exit</title></head>
<body>
<? exit(); ?>
<p>Questa frase non si vedrà</p>
</body>
</html>
```

Utili per gestire situazioni di errore che non consentono di proseguire l’esecuzione del nostro script PHP (fatal error). Per esempio, controlliamo il valore della variabile globale $n e mostriamo un messaggio di errore, bloccando l’esecuzione del programma, se questo è maggiore di uno.

```
<html>
```

```
<head><title>die</title></head>
<body>
<?
$n = 5;
if ($n > 1) die("<h1>\$n è maggiore di uno!!!</h1>");
?>
<h1>$n è minore o uguale ad uno!</h1>
</body>
</html>
```

Funzioni predefinite del linguaggio

abs: restituisce il valore assoluto di un numero.

```
$num = -3.4;
$aa = abs($a);
echo $aa, "\n";
```

restituirà 3.4

acos: restituisce l'arcocoseno dell'argomento.

```
$arg = 1;
$arc_cos = acos($arg);
echo "$arc_cos\n";
```

restituirà 0.

asin: restituisce il seno dell'argomento.

atan: restituisce l'arcotangente dell'argomento.

base64_decode: decodifica una stringa codificata in MIME base64.

base64_encode: codifica dati in MIME base64; ad esempio con:

```
$str = "Ciao, io sono pippo\n";
echo "$str\n";
$enc_str = base64_encode($str);
echo "$enc_str\n";
$dec_str = base64_decode($enc_str);
echo "$dec_str\n";
```

si passa allo script la stringa "$str" che è prima codificata e visualizzata, poi decodificata e nuovamente visualizzata.

basename: restituisce, dato un percorso, la componente di questo identificata da un nome di file.

```
$path = "/var/www/php/index.php3";
$base = basename($path);
echo "$base\n";
```

restituirà "index.php3";

bcadd: somma due numeri.

```
$num = bcadd(1.334, 4.44554, 2);
echo "$num\n";
```

restituirà 5.77; la funzione "bcadd" prende come primi due argomenti due numeri e, come terzi argomento opzionale, il numero di cifre da visualizzare dopo la virgola.

bccomp: compara due numeri: la funzione prende come argomento due numeri e, opzionalmente, un ulteriore numero che determina il numero di decimali da considerare dopo la virgola per considerare i due numeri uguali; restituisce "0" nel caso i due numeri siano uguali, "+1" se il numero di sinistra è maggiore di quello di destra e "-1" nel caso opposto.

```
$comp = bccomp(0.334, 0.301, 2);
echo $comp;
```

che restituirà "1"; ma se, al posto del "2" avessimo inserito uno oppure non avessimo inserito niente, il risultato sarebbe stato "0".

bcdiv: divide due numeri, con le stesse modalità descritte per "bcadd" e "bccomp".

bcmult: moltiplica due numeri, ed è possibile aggiungere un ulteriore parametro per

limitare il numero di cifre dopo la virgola.
$molt = bcmul(2.31, 3.21, 2);
echo "$molt\n";
restituirà 7.41.
bcpow: eleva a potenza due numeri, con la possibilità di specificare il numero di cifre dopo la virgola.
$pot = bcpow(2.3, 3, 2);
echo "$pot\n";
eleverà 2.3 alla terza potenza, approssimando il risultato alla seconda cifra decimale.
bcsqrt: calcola la radice quadrata di un numero, con la possibilità di approssimare il numero di cifre dopo la virgola aggiungendo un secondo elemento alla funzione (come avveniva per altre funzioni matematiche viste sopra;
bcsub: sottrae un numero da un altro, anche qui con la possibilità di approssimare le cifre dopo la virgola.
$num = bcsub(2, 5);
echo "$num\n";
restituirà "-3";
bin2hex: converte una stringa di dati dal formato binario a formato esadecimale.
ceil: restituisce il valore intero più alto riferito al numero passato come argomento alla funzione.
$num = ceil(3.22112);
echo "$num\n";
restituisce "4".
chdir: cambia la directory di lavoro.
$dir = "/var/www/";
chdir($dir);
restituisce TRUE se l'operazione ha successo, FALSO in caso contrario, ad esempio nel caso la directory non sia leggibile.
checkdate: controlla che una data sia valida; per considerarsi valida, una data deve avere: l'anno compreso fra "0" e "32767"; il mese compreso fra "1" e "12"; il giorno è compreso fra "0" ed il numero relativo al numero di giorni del mese a cui si fa riferimento.
chgrp: tenta di cambiare il gruppo di un file a "gruppo"; la funzione accetta come argomenti il nome del file cui si vogliono cambiare i permessi ed il nuovo gruppo di appartenenza.
chgrp(filename, gruppo);
Nei sistemi Windows non funziona ma restituisce sempre vero.
chmod: è equivalente al comando di sistema Unix "chmod" ed ha la stessa sintassi di chgrp.
chmop: rimuove i "whitespaces" da una stringa; spesso è utilizzato per eliminare i caratteri "\n" quando si riceve un argomento dallo standard input; il carattere eliminato può essere letto con:
$carattere = chop($string);
echo "$carattere\n";
chown: cambia l'owner di un file, come l'analogo comando di sistema Unix. Accetta come argomento il nome del file ed il nome del nuovo proprietario.
$file = "prova.txt";
chown($file, $user);
Nei sistemi Windows, non fa niente e restituisce sempre vero (ed è peraltro inutile inserire questa funzione all'interno di uno script che non supporta il comando "chown").
chr: restituisce il carattere ASCII specificato dal rispettivo numero.
$ascii= "0126";
$char = chr($ascii);
echo "$char\n";
chunk_split: divide una stringa in parti di "n" caratteri; il numero è passabile alla funzione

dopo la stringa da dividere. Se non impostato, di default è assunto come 76.
$string = "Questo è un corso per imparare il linguaggio php":
$split = chunk_split($string, 5);
restituirà.
Quest
o è u
n cor
so pe
r imp
arare
il l
ingua
ggio
php
La funzione è utile per l'encoding MIME base64 visto precedentemente con la funzione "base64_encode".
closedir: chiude una directory precedentemente aperta con la funzione opendir().
copy: crea la copia di un file.
$file = "prova.txt";
copy($file, "$file.bak");
cos: restituisce il valore del coseno dell'argomento.
count: conta gli elementi in una variabile.
$arr[0] = "abc";
$arr[1] = "def";
$arr[2] = "ghi";
$count = count($arr);
echo $count;
restituirà "3", visto che all'interno dell'array "$arr" sono presenti 3 elementi ($arr[0], $arr[1], $arr[2]).
crypt: cripta una stringa.
crypt(string, salt);
In pratica, dovremo passare alla funzione la stringa che dovrà essere criptata e, opzionalmente, il seme con sui criptarla; se questo non è passato alla funzione, sarà generato in maniera random dal PHP stesso. Un esempio di criptazione di una stringa.
$var = "Questa è una variabile";
$crypt = crypt($var, "aa");
echo $crypt;
che restituirà la stringa criptata.
current: restituisce il primo elemento di un array:
$arr[0] = "abc";
$arr[1] = "def";
$arr[2] = "ghi";
$current = current($arr);
echo $current;
visualizzerà "abc".
date: visualizza la data in formato che è possibile definire; la funzione riconosce come validi i seguenti formati.
a: am/pm;
A: AM/PM
d: giorno del mese in due cifre, da "0" a "31";
D: giorno del mese in formato testo, ad esempio "Mon";
F: mese, in formato testuale, ad esempio "March";
h: ora nel formato "01", "12";
H: ora nel formato "00", "23";

g: ora nel formato "1", "12";
G: ora nel formato "0", "23";
i: minuti, nel formato "00", "59";
j: giorno del mese nel formato "1", "31";
l: giorno della settimana, ad esempio "Monday";
L: specifica se l'anno è bisestile o meno ("1" oppure "0");
m: mese nel formato "01", "12";
n: mese nel formato "1", "12";
M: mese in formato testuale corto, ad esempio "Jan";
s: secondi da "00" a "59";
S: suffisso inglese per gli ordinali, "st", "nd", "rd", "th";
t: numero di giomi nel mese corrente, da "28" a "31";
w: giorno della settimana in formato numerico ("0"=domenica);
Y: anno in quattro cifre, ad esempio "2000";
y: anno in due cifre, ad esempio "00";
Ad esempio, si potrebbe scrivere.
echo (date("l d F y H:i:s a"));
per avere la data corrente
Friday 23 June 00 11:45:48 am
debugger_off: disabilita il debugger PHP.
debugger_on: abilita il debugger PHP.
decibin: converte un numero da decimale a binario; ad esempio, il numero "10" decimale è "1010" in formato binario.
$bin = decbin(10);
echo $bin, "\n";
che restituirà appunto "1010".
dechex: converte un numero da decimale a esadecimale.
decoct: converte un numero da formato decimale a formato ottale.
define: definisce una costante.
define("COSTANTE", "Questa è una costante");
echo COSTANTE;
L'esempio riportato sopra visualizzerà "Questa è una costante".
defined: controlla che una certa costante esista.
define("COSTANTE", "Questa è una costante");
if (defined("COSTANTE")) {
echo "La costante è definita\n";} else {
echo "La costante non è definita\n";}
che visualizza un messaggio a seconda che la costante sia o meno definita.
die: visualizza un messaggio ed esce dal programma.
if (defined($num)) {
echo "\$num è definito\n";} else {
die ("\$num non è definito; impossibile proseguire\n");}
dirname: quando si specifica un path, riporta il path senza il nome del file finale: se ad esempio il path è "/home/yourname/public_html/index.php3" la funzione restituirà solamente "/home/yourname/public_html".
$path = "/home/yourname/public_html";
echo(dirname($path));
Fa l'esatto contrario della funzione "basename()"; combinando le due funzioni, si può avere il path completo di un file, compreso il suo nome.
diskfreespace: restituisce lo spazio di disco libero; se volessimo ad esempio vedere quanto spazio rimane nella directory root della macchina, potremo scrivere.
echo(diskfreespace("/"));
each: restituisce il primo valore di un array utilizzando le keys 0, 1, key e value; se l'array è associativo.

$array = array ("nome" => "valore", "nome2" => "valore2");
while (list($key, $val) = each ($array)) {
echo "$key => $val\n";}
che restituirà
nome => valore
nome2 => valore2
Se invece l'array non è associativo, il codice sarà.
$array = array ("nome", "valore", "nome2", "valore2");
while (list($key, $val) = each ($array)) {
echo "$key => $val\n";}
ed il risultato
0 => nome
1 => valore
2 => nome2
3 => valore2
La funzione "each()" è spesso utilizzata insieme a "list()".
echo: visualizza una o più stringhe.
ereg_replace: sostituisce un'espressione regolare con determinati valori; alla funzione devono essere passati tre argomenti: il primo indica il testo da sostituire, il secondo è il testo utilizzato per la sostituzione ed il terzo è la stringa da modificare.
$stringa = "Questo è un corso su ASP";
echo ereg_replace("ASP", "PHP", $stringa);
Notate che si sarebbe potuto scrivere anche
echo ereg:replace("ASP", "PHP", "Questo è un corso su ASP");
che non avrebbe avuto molto senso, comunque.
ereg: esegue il matching di un'espressione regolare.
if (ereg("([0-9]{4})-([0-9]{1,2})-([0-9]{1,2})", $date, $regs)) {
echo "$regs[3].$regs[2].$regs[1]"; } else {
echo "Invalid date format: $date";}
Tutto questo ciclo è fatto per controllare che una data sia in formato corretto. Vediamo il significato di "[0-9]{4})-([0-9]{1,2})-([0-9]{1,2}". Per chi conosca le espressioni regolari, non sarà difficile tradurre quanto sopra con "un numero da 0 a 9 ripetuto quattro volte seguito da un '-', da un numero da 0 a 9 ripetuto una o due volte, da un '-' e da un numero da 0 a 9 ripetuto una o due volte". Come spesso accade, leggere un'espressione regolare è molto più semplice che tradurla nel linguaggio parlato.
eregi_replace: funziona esattamente come "ereg_replace()", solamente che in questo caso l'espressione regolare è sostituita in maniera "case insensitive", ossia ignorando i caratteri maiuscoli e minuscoli.
eregi: funziona esattamente come "ereg()", solamente che in questo caso l'espressione regolare è sostituita in maniera "case insensitive".
error_log: invia un messaggio di errore al file di log del webserver, direttamente alla porta TCP dalla quale è arrivata la richiesta o in un file.
error_log(message, mesage_type, destination);
Message_type è un numero che specifica dove deve arrivare il messaggio. Può essere:
1. il messaggio è inviato al logger del PHP o nel file specificato da "error_log";
2. il messaggio è inviato per e-mail al parametro specificato in "destination";
3. il messaggio è inviato al debugger;
4. il messaggio è inserito in append al parametro specificato in destination".

escapeshellcmd: se si richiama un comando esterno da una shell, con questo comando si fa in modo che i metacaratteri della shell siano fatti precedere da un carattere di escape per evitare che il comando produca degli errori.
exec: esegue un programma esterno.
exit: esce da uno script; il comando "exit()" è utile nei casi si voglia fermare uno script in caso qualcosa non soddisfi determinate condizioni.

if (condizione) { esegui il blocco; } else { exit;}
"exit()" non riporta un messaggio di errore come fa "die()". Non è possibile scrivere:
exit "Esco dal programma\n";
o meglio, è possibile ma non ha alcun effetto se non quello di uscire.
exp: eleva "e" (2.71828.....) alla potenza riportata come argomento della funzione.
echo exp(3);
restituirà: 20.0855369...
explode: divide una stringa secondo un determinato pattern. Ad esempio, volendo dividere una stringa contenente tre nomi separati da virgole.
$nomi = "Tizio,Caio,Sempronio";
list ($nome1, $nome2, $nome3) = explode (",", $nomi);
echo "$nome1\n$nome2\n$nome3\n";
che restituirà
Tizio
Caio
Sempronio
Explode() è una versione semplificata di "split(). Entrambe le funzioni, inoltre, sono molto utili nel caso ci sia la necessità di leggere determinati file contenenti delle liste.
fclose: chiude un puntatore ad un file precedentemente aperto con fopen().
feof: testa un puntatore ad un file per vedere se si è alla fine dello stesso.
fgetc: restituisce il primo carattere del puntatore precedentemente aperto con fopen(); se ad esempio il puntatore $file punta al file "/tmp/prova.txt" che contiene solamente la riga "Ciao", un codice come il seguente:
$char = fgetc($file);
echo "$char\n";
restituirà "C" (ovviamente senza virgolette!).
file_exists: controlla se un file esiste, riportando TRUE in caso positivo o FALSE in caso negativo.
if (file_exists($file)) {
print "$file esiste;}
Può essere molto utile utilizzare questa funzione nel caso sia necessari agire su uno o più file, in modo da agire sullo stesso solo nel caso questo esista senza rischiare d'incorrere in inspiegabili "anomalie" dello script.
filegroup: restituisce il gruppo al quale appartiene il file.
$filename = "/tmp/prova.txt";
$group = filegroup($filename);
echo "$filename appartiene al gruppo $group\n";
Ovviamente, la funzione è implementata nei soli sistemi multiuser.
filesize: restituisce la grandezza di un file.
$filename = "/tmp/ptova.txt";
$size = filesize($filename);
echo "$filename -> $size\n";
filetype: determina il tipo di file; i valori possibili sono: fifo, char, dir, block, link, file e unknown.
flock: applica il locking ad un file; specificamente, flock() opera su un puntatore ad un file precedentemente aperto e le operazioni possibili sono:
1. per il lock in lettura;
2. per il lock in scrittura;
3. per rimuovere il lock, di qualsiasi tipo sia;
4. per impedire che flock() blocchi un file mentre applica il lock.

Ad esempio, per applicare flock() ad un puntatore "$file" precedentemente definito occorrerà scrivere.
flock($file, 2);
/ Per impedire che il file sia letto*/*

.....
/ Codice per lavorare sul file */*
flock($file, 3);
/ Per rimuovere il flock */*
fopen: apre un file oppure un'URL.
fopen(filename, mode);
Ovviamente a "filename" corrisponde il nome del file o l'URL dello stesso, ed a "mode" la modalità con il quale questo deve essere aperto: si ha qui la possibilità di scegliere fra:
r -> apre il file in sola lettura;
r+ -> apre il file in lettura ed in scrittura;
w -> apre il file in sola scrittura;
w+ -> apre il file in lettura e scrittura;
a -> apre il file in sola scrittura e inserisce il puntatore alla fine del file ("w" lo inserisce alla fine);
a+ -> apre il file in lettura e scrittura inserendo il puntatore alla fine del file.
Ad esempio, per aprire un file locale in sola lettura.
$file = fopen("/tmp/prova.txt", "r");
Per un URL, invece.
$file = fopen("http://www.myhost.com/index.html", r");
Per tutte le successive operazioni sul file, poi, dovremo agire direttamente sul puntatore ($file) e non direttamente sul file.
header: invia un qualsiasi header http.
header("Pragma: no-cache");
Questa funzione è molto utile in diversi casi: ad esempio, per forzare un'autorizzazione, si può inviare un "401".
hexdec: restituisce il valore decimale di un numero esadecimale.
implode: è l'opposto di "explode": la sintassi è identica, ma in questo caso restituisce una stringa con i valori separati dal primo argomento della funzione.
in_array: restituisce valore vero se in un array è presente un determinato valore.
$numeri = array("1", "2", "3", "4", "5");
$num = 2;
if (in_array($num, $numeri)) {
print "$num è presente nell'array \$numeri\n";}
is_array: controlla se una data variabile è un array.
if (is-array($var)) {
echo "\$var è un array\n";}
La stessa cosa è fatta, con ovviamente la differenza dell'argomento, dalle funzioni.
1. is_dir;
2. is_double;
3. is_executable;
4. is_file;
5. is_float;
6. is_int;
7. is_integer;
8. is_link;
9. is_long;
10. is_object;
11. is_readable;
12. is_real;
13. is_string;
14. is_writeable.

isset: restituisce TRUE nel caso la variabile esista, falso nel caso opposto; ad esempio, per vedere se esiste o meno una variabile.
$a = 10;

echo isset($a), "\n";
echo isset($b), "\n";
che restituirà 1 e 0; ricordiamo che 1 è considerato valore di successo (TRUE), 0 di insuccesso (FALSE);
join: unisce gli elementi di un array con una determinata stringa; l'uso è identico a quello d'implde().
key: prende una chiave da un array associativo.
$array = array("1" => "uno");
$chiave = key($array);
echo "$chiave\n";
che restituirà "1". Questa funzione è utile per estrarre tutte le chiavi di array associativi complessi.
link: crea un hard link.
link(target, link);
Le informazioni sui link possono essere visualizzate con linkinfo().
list: assegna delle variabili come se fossero parti di un array.
$array = array ("nome" => "valore", "nome2" => "valore2");
while (list($key, $val) = each ($array)) {
echo "$key => $val\n";
}
In questo caso, list() è utilizzato per "stilare" una lista di variabili che saranno estratte dall'array, senza ovviamente dare loro un valore ma lasciando alle parti successive del codice l'assegnazione dei loro valori. È inoltre utile notare che le variabili create da list() non assumono un solo valore, ma per ogni chiamata assumono un diverso valore, secondo gli elementi presenti nell'array.
mail: funzione per l'invio di e-mail.
mail(To, Subject, Message, Altri_headers);
Supponendo di voler inviare un'e-mail a "nome@host.com" con subject "Prova" e volendo specificare il nome del mittente.
mail("nome@host.com", "Subject", "Questo è il corpo dell'email",
"From: mittente <mittente@host.net>);
Ovviamente, nel file di configurazione, dovrete aver specificato la locazione di sendmail (o analogo programma per l'invio delle e-mail).
max: restituisce il valore più alto di una serie di variabili.
$num = 1;
$num2 = 23;
$num3 = 0.3;
$max = max($num, $num2, $num3);
echo $max, "\n";
restituirà "23";
Opposto a max() è min(), che adotta la stessa sintassi di max().
mkdir: crea una directory, di cui si deve specificare il percorso ed i permessi.
mkdir("/tmp/prova", 0777);
creerà la directory "/tmp/prova" con permessi impostati a 0777.
opendir: apre una directory, della quale sarà possibile leggere gli elementi con readdir() e, poi, chiuderla con closedir().
phpinfo: visualizza moltissime informazioni sul PHP stesso.
phpinfo();
phpversion: visualizza la versione di PHP che si sta utilizzando.
popen: apre un puntatore ad un processo che deve essere chiuso con pclose().
print: visualizza una stringa a video come echo().
rand: genera un valore numerico in maniera casuale; se si volesse un valore compreso fra 10 e 20, si potrebbe scrivere:
$random = rand(10, 20);

range: crea un array contenente un range di valori interi specificato; ad esempio, per creare un array con valori da 1 a 10.
$array = range(1, 10);
rename: rinomina un file.
rename("oldname", "newname");
per rinominare "oldname" come "newname".
rmdir: analogo comando Unix, rimuove una directory; questo può essere fatto solo se: la directory è vuota e i permessi sulla directory lo consentono.
round: arrotonda un numero.
$numero = round(2,3); / restituisce 2 */*
$numero = round(2.5); / restituisce 3 */*
$numero = round(2.6); / restituisce 3 */*
I decimali da 0 a 4 sono approssimati all'intero precedente, da 5 a 9 all'intero successivo.
shuffle: ordina in modo casuale gli elementi di un array; ad esempio, per poter visualizzare gli elementi di un array in maniera casuale.
$num = range(0,10);
shuffle($num);
while (list(,$numero) = each($num)) {
echo "$numero ";
}
sin: restituisce il seno dell'espressione.
sizeof: calcola il numero di elementi presenti in un array. Se ad esempio si volesse calcolare il numero di elementi in un array.
$array = array("1", "2", "3", "4", "5");
$size = sizeof($array);
if ($size <= 10) {
echo "L'array contiene meno di 10 elementi\n";
} else {
echo "L'array contiene più di 10 elementi\n";
}
sleep: mette lo script in pausa per un determinato numero di secondi, specificato come argomento della funzione; ad esempio, "sleep(10)" farà in modo che lo script sia sospeso per 10 secondi, per poi continuare normalmente.
split: divide una stringa secondo un determinato pattern.
$linea = "tizio||caio||sempronio";
list ($uno, $due, $tre) = split("\|\|", $linea, 3);
print "1 => $uno\n2 => $due\n3 => $tre\n";
è stato necessario inserire un carattere di escape (\) prima di ogni "|" nell'espressione da utilizzare per splittare la riga.
sqrt: restituisce la radice quadrata dell'argomento.
strcmp: esegue una comparazione su due stringhe.
$cmp = strcmp("Ciao", "Ciao a tutti");
if ($cmp == "0") {
print "Le stringhe sono identiche\n";
} elseif ($cmp < 0) {
print "La seconda riga è più lunga della prima\n";
} elseif ($cmp < 0) {
print "La prima riga è più lunga della prima\n";
}
restituisce "La seconda riga è più lunga della prima". La funzione, infatti, restituisce "0" se le stringhe sono uguali, un valore minore di zero se la seconda è più lunga della prima e maggiore di zero se la prima è più lunga della seconda.
system: esegue un programma di sistema, ne restituisce l'output e ritorna allo script.
tan: restituisce la tangente dell'argomento.

unset: elimina il valore di una variabile.
usleep: come sleep(), ma questa funziona blocca lo script per N microsecondi.
Funzioni legate ad Apache
apache_lookup_uri: questa funzione esegue una richiesta per un determinato URI (Uniform Resource Identifier: un indirizzo web) e riporta i risultati di tale richiesta: i risultati sono contenuti in un array.
$array = apache_lookup_uri($url);
Con questa funzione avremo informazioni su:
status (che è poi un codice numerico);
the_request: il tipo di richiesta, il metodo, il file richiesto ed il protocollo utilizzato;
method: il metodo utilizzato;
uri: l'uri relativo alla richiesta;
filename: il nome del file con il path locale;
path_info: informazioni sul path;
no_cache;
no_local_copy;
allowed;
sent_bodyct;
bytes_sent;
byterange;
clenght;
unparsed_uri;
request_time.
Ad esempio, utilizzando un codice come il seguente.
```
$uri = "http://localhost";
$array = apache_lookup_uri("$uri");
while ( list ( $header, $valore ) = each( $array ) ) {
echo "$header: $valore<br>\n";
}
```
si potrebbe ottenere qualcosa di simile a
```
status: 200
the_request: GET /php/prova.php3 HTTP/1.0
method: GET
uri: /php/localhost
filename: /var/www/php/localhost
path_info:
no_cache: 0
no_local_copy: 1
allowed: 0
sent_bodyct: 0
bytes_sent: 0
byterange: 0
clength: 0
unparsed_uri: /php/localhost
request_time: 965150868
```
getallheaders: questa funzione è possibile richiedere tutti gli headers relativi ad una richiesta: questi headers con molti altri possono essere letti anche con la funzione "phpinfo()".
$array = getallheaders();
e, utilizzando un codice
```
$headers = getallheaders();
while (list($header, $value) = each($headers)) {
echo "$header: $value<br>\n";
}
```

si può ottenere qualcosa di simile a
*Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */**
Accept-Charset: iso-8859-1,,utf-8*
Accept-Encoding: gzip
Accept-Language: en
Connection: Keep-Alive
Host: localhost
Pragma: no-cache
User-Agent: Mozilla/4.72 [en] (X11; I; Linux 2.2.14 i586)
Virtual: è l'equivalente di <!--# include virtual="/file.txt"-->, chiamata relativa ai Server Side Includes. Con questa funzione, potremo includere un qualsiasi file in una pagina generata dinamicamente con il PHP.
virtual(file);
Il file da specificare può essere qualunque file di testo e, di norma, è riferito alla directory radice dell'host; quindi, se abbiamo un file visibile via web con "http:/www.dominio.com/file.txt", la sintassi da utilizzare con "virtual()" è
virtual("/file.txt");
Funzioni relative alla criptazione
PHP offre agli sviluppatori una serie di funzioni relative alla criptatura, legate alla libreria mcrypt; tale libreria supporta moltissimi algoritmi di criptazione, alcuni più utilizzati ed altri meno. Gli algoritmi sono: DES, TripleDES, Blowfish, 3-WAY, SAFER-SK64, SAFER-Sk128, TWOFISH, TEA, RC2, GOST, RC6, IDEA.
Per funzionare con tale libreria, il PHP deve essere stato compilato con l'opzione "--with-mcrypt". I comandi fondamentali sono quattro, tutti con la medesima sintassi:
mcrypt_cfb(): cipher feedback; codifica byte per byte;
mcrypt_cbc(): cipher block chaining: utile per l'encoding dei file con un gran margine di sicurezza;
mcrypt_ecb(): electronic codebook: utilizzata per dati random, dove il livello di sicurezza non è altissimo;
mcrypt_ofb(): output feedback: simile a cfb, ma è data maggiore attenzione agli errori.
$encrypted = mcrypt_XXX(algoritmo, chiave, input, encode/decode)
dove:
XXX è il metodo che s'intende utilizzare (cfb, cbc, cfb o ofb);
algoritmo è l'algoritmo che s'intende utilizzare, con la sintassi
MCRYPT_ALGORITMO
Ad esempio, si potrebbe utilizzare
MCRYPT_BLOWFISH
oppure
MCRYPT IDEA
chiave altro non è che la chiave con cui si andranno a criptare i dati;
input sono i dati da criptare;
encode/decode indica alla funzione se si devono criptare o decriptare i dati; per questo, si usano rispettivamente
MCRYPT_ENCRYPT
e
MCRYPT_DECRYPT
Volendo criptare una semplice stringa di testo con chiave di criptatura "La mia chiave" utilizzando CFB con l'algoritmo IDEA.
$stringa = "Una semplice stringa di testo";
$chiave = "La mia chiave";
$encrypted = mcrypt_cfb(MCRYPT_IDEA, $chiave, $stringa, MCRYPT_ENCRYPT);
Chiunque voglia poi leggere i dati criptati ($encrypted) dovrà ovviamente conoscere la chiave, il metodo e l'algoritmo utilizzati; quindi potrebbe scrivere qualcosa del tipo:
$chiave = "La mia chiave";

$stringa = mcrypt_cfb(MCRYPT_IDEA, $chiave, $encrypted, MCRYPT_DECRYPT);
La sua variabile "$stringa", quindi, conterrà "Una semplice stringa di testo".

Funzioni legate al protocollo FTP

Fra i vari protocolli, PHP ci mette a disposizione una vasta libreria di funzioni legate al protocollo FTP (FILE TRANSFER PROTOCOL), per il trasferimento di file da un computer all'altro in una rete.

ftp_connect: questa è la funzione "principale" nel senso che ci permette di stabilire una connessione FTP fra la nostra macchina ed il server FTP remoto.

$stream = ftp_connect(host, port);

dove host è il nome del server cui intendiamo connetterci e port (opzionale) è la porta alternativa alla quale ci si vuole connettere; se questa non è specificata, è utilizzata la porta di default per il protocollo FTP, ossia la 21. Nella variabile $stream, inoltre, è immagazzinato appunto lo stream di dati che il client (in questo caso il PHP) riceve dal server, ossia i messaggi di connessione accettata (con i vari dettagli) o di connessione rifiutata. Ad esempio, per connetterci alla porta di default del server FTP "ftp://ftp.host.com" utilizzeremo.

$stream = ftp_connect("ftp://ftp.host.com");

ftp_login: dopo la connessione, abbiamo bisogno di identificarci in modo che il server ci permetta lo scambio dei dati. Molti saranno abituati a non vedere tale fase visto che, con i più diffusi client FTP grafici essa è svolta in automatico utilizzando le informazioni di login (username e password) inseriti come opzioni per il collegamento, ma questa è una fase di vitale importanza per la connessione.

$login = ftp_login(stream, username, password);

Se ad esempio in precedenza ci eravamo collegati all'host "ftp.host.com", utilizzando la variabile "$stream", adesso potremo procedere al login vero e proprio con

$login = ftp_login($stream, "utente", "password");

La variabile $login ci servirà per capire se il login è andato o meno a buon fine e conterrà il valore "1" per il successo, "0" altrimenti. Ad esempio, per vedere se continuare lo scambio di dati in seguito all'autorizzazione potremo utilizzare il valore assegnato a tale variabile e scrivere.

```
if ($login == "1") {
... # Fai il resto delle operazioni
} else {
echo "Autorizzazione non riuscita\n";
}
```

Una volta connessi, potremo sapere su che macchina stiamo lavorando con la funzione "ftp_systype()".

$system = ftp_systype($stream);

ftp_pwd: questa funzione invoca il comando "pwd", ovvero "Print work directory", che potremo tradurre come "Visualizza la directory corrente". Per vedere a che directory veniamo connessi dopo il login.

$directory = ftp_pwd($stream);

dove $stream è sempre la variabile che abbiamo utilizzato per la connessione con "ftp_connect()".

ftp_cdup e ftp_chdir: queste due funzioni servono rispettivamente a muoversi alla directory superiore e a muoversi in una determinata directory all'interno del server.

$var = ftp_cdup($stream);

$newdir = ftp_chdir($stream, "nuova_directory");

Se ad esempio al login siamo nella directory "/" e volessimo spostarci in "/var/wwwdata" potremo scrivere.

$newdir = ftp_chdir($stream, "/var/wwwdata");

ftp_mkdir e ftp_rmdir: queste due funzioni invocano il comando "mkdir "(crea una directory) e "rmdir" (rimuovi una directory). La prima restituisce il nome della nuova directory, la seconda solamente i valori true o false.

$mydir = ftp_chdir($stream, "/var/wwwdata/");
 # Posizioniamoci in "/var/wwwdata".
$newdir = ftp_mkdir($stream, "prova")
 # Creiamo la directory "prova" come sottodirectory di "/var/wwwdata"
$deleted_dir = ftp_rmdir($stream, $newdir);
 # Cancelliamo la directory appena creata!
 # Possiamo ora controllare il tutto con:
if ($deleted_dir == "1") {
print "Operazione completata con successo.\n";
} else {
print "Qualcosa non è andato per il verso giusto.\n";
}

ftp_nlist: questa funzione è analoga al comando "dir", ossia il comando utilizzato per vedere i nomi dei file presenti in una directory.
$list = ftp_nlist($stream, directory);
Ad esempio, possiamo portarci nella directory "/var/wwwdata" e leggerne i file con
$newdir = "/var/wwwdata";
$list = ftp_nlist($stream, $newdir);
I risultati sono contenuti in un array, quindi un 'echo "$list"' non avrebbe alcun senso.
ftp_get: funzione che richiama il comando GET, per scaricare un file dal server remoto. Dobbiamo specificare per la funzione, oltre al solito stream, il nome del file locale, il nome del file remoto e la modalità di trasferimento (FTP_ASCII o FTP_BINARY.
$file = ftp_get($stream, local_filename, remote_filename, mode);
Ad esempio, volendo scaricare dal server il file "data.txt" (supponiamo di essere già all'interno della directory che lo contiene) inserendolo nella directory "/tmp" con nome "file.txt" in ASCII mode.
$file = ftp_get($stream, "/tmp/file.txt", "data.txt", FTP_ASCII);
Per vedere se l'operazione ha avuto o meno successo, possiamo operare in due modi: controllare se effettivamente il file c'è nel nostro disco oppure controllare il valore della variabile $file: se ha valore "1" allora l'operazione è stata completata, se ha valore "0" nessun file sarà stato scaricato sul nostro disco.
ftp_put: questa funzione fa esattamente il contrario di "ftp_put()", ossia carica un file sul server.
$file = ftp_put($stream, remote_filename, local_filename, mode);
Le opzioni sono identiche alle precedenti, quindi possiamo fare l'esempio contrario del precedente: carichiamo il file locale "/tmp/file.txt" nella directory remota (siamo già in questa directory) con il nome "data.txt". Tutto in ASCII mode, ovviamente.
$file = ftp_put ($stream, "data.txt", "/tmp/file.txt", FTP_ASCII);
Anche qui, possiamo controllare in due modi: valutando il valore di $file oppure invocando la funzione "ftp_nlist()" per vedere se fra i file c'è anche "data.txt".
ftp_size: restituisce le dimensioni di un dato file.
$size = ftp_size($stream, remote_filename);
Per tornare agli esempi precedentemente fatti, vediamo di conoscere la grandezza del file "data.txt", che si trova nella directory in cui siamo al momento.
$size = ftp_size($stream, "data.txt");
in modo che la variabile $size contenga le dimensioni del file "data.txt".
ftp_mdtm: restituisce la data di ultima modifica di un file, restituendola come Unix timestamp.
$date = ftp_mdtm($stream, remote_filename);
Ad esempio, volendo sapere la data di ultima modifica del file "data.txt" possiamo scrivere.
$date = ftp_mdtm($stream, "data.txt");
Anche in questo caso, la variabile "$data" conterrà la data di ultima modifica del file oppure il valore "-1" in caso di insuccesso (file inesistente o casi del genere).
ftp_rename e ftp_delete: queste due funzioni servono per rinominare un file e per

cancellarlo.
$name = ftp_rename($stream, oldname, newname);
dove "oldname" è il nome originario del file e "newname" è il nuovo nome che vogliamo assegnare al file.
Ad esempio, per rinominare il file "data.txt" in "dati.dat" possiamo scrivere.
$name = ftp_rename($stream, "data.txt", "dati.dat");
La variabile $name conterrà "1" se l'operazione ha avuto successo, "0" altrimenti (file inesistente o casi simili).
La funzione "ftp_delete(),invece, si utilizza con sintassi.
$delete = ftp_delete($stream, file);
Ad esempio, per eliminare il file "dati.dat" presente nella "current-directory" possiamo scrivere.
$delete = ftp_delete ($stream, "dati.dat");
Anche in questo caso la variabile può contenere valore "1" (il file è stato eliminato) o "0" (qualcosa non è andato per il verso giusto).
ftp_quit: il lavoro sul server è terminato e possiamo disconnetterci utilizzando la funzione "ftp_quit()".
$quit = ftp_quit($stream).
È sempre consigliato invocare questa funzione invece di chiudere il programma in esecuzione, più che altro per una questione di rispetto verso il server.
Funzioni relative al networking
gethostbyaddr e gethostbyname: queste due funzioni permettono, rispettivamente, di ricavare il nome di un host partendo dal suo indirizzo IP e di ricavare l'indirizzo IP associato ad un nome di dominio.
$hostname = gethostbyaddr(IP_address);
$hostaddress = gethostbyname(hostname);
Ad esempio, poniamo che "www.server.com" abbia indirizzo IP 123.456.78.9 che noi sappiamo: ma vogliamo che sia uno script PHP a dircelo. Scriveremo quindi.
$hostname = gethostbyaddr("123.456.78.9");
$hostaddress = gethostbyname("http://www.host.com");
print "$hostname ha indirizzo IP $hostaddress.\n";
getservbyname: questa funzione ci permette di ricavare il numero della porta alla quale è associato un determinato servizio.
$porta = getservbyname(service, protocol);
Ad esempio, volendo sapere a che porta è associato il protocollo ftp, possiamo scrivere.
$ftp_port = getservbyname("ftp", "tcp");
che con grande probabilità ci restituirà la porta 21 (quella di default per il FTP).
Standard POSIX
Il PHP mette a disposizione molte funzioni POSIX-compliant, come definito nello standard document IEEE 1003.1; tali funzioni ci permetteranno di innestare una completa interazione con il sistema, per un'ottimale scrittura degli script.
posix_kill: invia un segnale ad un determinato PID, restituendo FALSE in caso il PID non sia associato ad alcun processo, vero nel caso opposto.
posix_kill(PID, SIG)
dove PID è il pid del processo cui vogliamo inviare il segnale e SIG è, appunto, il segnale.
Ad esempio, volendo inviare un segnale di kill a Netscape (pid 1234), potremo scrivere.
$pid = "1234";
$signal = posix_kill($pid, KILL);
posix_getpid: utilizzare la funzione sopra descritta lavorando da uno script è assai duro per un semplice motivo: non avendo accesso alla shell, non possiamo sapere il PID associato ad una determinata applicazione. Ma il PHP ci viene incontro anche in questo con la funzione posix_getpid, che permette d'inserire in una variabile il PID di un processo.
$pid = posix_getpid($application);
dove application è ovviamente l'applicazione delle quale ci interessa il PID. Utilizzando

questa funzione con la precedente, potremo scrivere qualcosa del tipo.
$pid = posix_getpid("netscape");
$signal = posix_kill($pid, KILL);
posix_getppid: la funzione posix_getppid() restituisce il padre del processo in uso.
$parent = posix_getppid();
echo $parent;
avremo visualizzato a video il padre del processo che abbiamo appena lanciato: nel nostro caso, visto che abbiamo lanciato il comando da una shell, il PID sarà proprio legato alla shell.
posix_getuid, posix_geteuid, posix_getgid, posix_getegid: queste quattro funzioni sono molto simili e restituiscono rispettivamente:
Real user ID del processo;
Effective user ID del processo;
Real GID del processo;
Effective GID del processo.
$UID = posix_getuid();
echo "UID = $UID\n";
$EUID = posix_geteuid();
echo "EUID = $EUID\n";
$GID = posix_getgid();
echo "GID = $GID\n";
$EGID = posix_getegid();
echo "EGID = $EGID\n";
posix_getlogin: questa funzione ci permette di recuperare il nome di login dell'utente che esegue lo script.
$login = posix_getlogin("/usr/sbin/apache");
echo $login;
per fare in modo di aver visualizzato a video il nome dell'utente che esegue lo script.
posix_uname: questa funzione è simile ad "uname" dei sistemi Posix-compliant e restituisce informazioni sul sistema. Tali informazioni sono organizzate in un array che contiene le keys: sysname, nodename, release, version, machine.
Ad esempio, lanciando uno script del tipo.
$uname = posix_uname();
while (list($key, $value) = each ($uname)) {
echo "$key -> $value\n";
}
si otterrà come risultato.
sysname -> Linux
nodename -> scatolinux
release -> 2.2.14
version -> #11 mar lug 4 00:06:07 CEST 2000
machine -> i586
posix:getpwname: questa funzione ci permette di avere determinate informazioni su un utente del sistema partendo dal suo ID; allo stesso modo, la funzione "posix_getpwuid()" riporta le stesse informazioni partendo dal suo ID.
$user = posix_getpwnam("edo");
while (list($info, $value) = each($user)) {
echo "$info -- $value\n";
}
che restituisce
name -- nome_utente
passwd -- x
uid -- NUM
gid -- NUM

gecos -- ,,,
dir -- /home/nome_utente
dove "nome_utente" è il nome dell'utente e "/home/nome_utente" è la sua home directory, NUM sono i due attributi numerici associati all'UID ed al GID e gecos sono informazioni addizionali sull'utente.
posix_getrlimit: questa funzione restituisce un array contenente i limiti delle risorse del sistema.
$resources = posix_getrlimit();
while (list($key, $value) = each ($resources)) {
echo "$key -> $value\n";}
che restituirà un output del tipo

```
soft  core      ->  0
hard  core      ->  unlimited
soft  data      ->  unlimited
hard  data      ->  unlimited
soft  stack  -  >   8388608
hard  stack     ->   unlimited
soft  totalmem  ->  unlimited
hard  totalmem  ->  unlimited
soft  rss       ->   unlimited
hard  rss       ->   unlimited
soft  maxproc   ->  256
hard  maxproc   ->  256
soft  memlock   ->   unlimited
hard  memlock   ->  unlimited
soft  cpu       ->  unlimited
hard  cpu       ->  unlimited
soft  filesize  ->   unlimited
hard  filesize  ->  unlimited
soft  openfiles ->   1024
hard  openfiles ->  1024
```

Le estensioni
La differenza fra le funzioni di libreria e le estensioni è sostanziale: le prime sono presenti direttamente all'interno del motore (built-in), le seconde sono presenti in librerie aggiuntive che devono essere installate sul sistema e richiamate in maniera particolare. Prima di tutto, vediamo di capire il meccanismo di caricamento dinamico di queste librerie: aprendo il file "php3.ini" vedremo un paragrafo dedicato alle "Extension", ossia estensioni nel linguaggio stesso: sono un insieme di librerie che sono richiamate al momento dell'esecuzione di uno script come avviene, in maniera analoga, per il caricamento dei moduli con un server web. La sintassi per il caricamento di queste estensioni di linguaggio è molto semplice: una volta che avremo installato la libreria sul sistema, non ci resta che aggiungere nel file php3.ini la riga.
extension=libreria
È qui necessario un discorso mirato per i sistemi Unix ed i sistemi Windows: per entrambi la sintassi è identica, ma ovviamente il nome delle librerie e la loro estensione no! Nei sistemi Windows, una libreria si riconosce dall'estensione ".dll", mentre per Unix questa è ".so": quindi, secondo il sistema, dovremo utilizzare il corretto nome per la libreria e, soprattutto, la corretta estensione. Nello scrivere il nome della libreria che ci interessa caricare, non dobbiamo soffermarci sul percorso completo, ma è necessario solamente il nome della stessa, ad esempio "pgsql.so" per i database Postgres. Questo perché, nello stesso file, è presente un'altra linea di configurazione che definisce in quale directory sono presenti queste librerie: leggendo il file php3.ini potrete trovare la riga "extension_dir = directory" che istruirà il motore sulla locazione standard delle librerie. Quindi, quando specifichiamo con "extension" una libreria che vogliamo sia caricata per l'esecuzione di

uno script, sono di fatto uniti "extension_dir" e "extension": se ad esempio "extension_dir" è "/usr/lib/php3" e abbiamo impostato "extension=pgsql.so", il PHP saprà che per caricare la libreria "pgsql.so" dovrà cercarla in "/usr/lib/php3/pgsql.so". Inoltre, con "extension= ...." è possibile specificare non solo una libreria ma tutta la seri di librerie che ci possono fare comodo.

extension=pgsql.so
extension=mysql.so
extension=gd.so
extension=imap.so
extension=ldap.do
extension=xml.so

e via dicendo; come noterete dalla seconda riga, poi, è possibile specificare anche due o più librerie per uno stesso "campo" in questo caso, ci sono due librerie per due database (Postgres e MySQL) che, oltre a non entrare in conflitto l'una con l'altra, potrebbero teoricamente anche essere utilizzate contemporaneamente. Nel caso si cerchi di richiamare una funzione non built-in all'interno di uno script, lo script visualizza una messaggio d'errore che ci avverte che stiamo cercando di utilizzare una funzione non riconosciuta.

FORM HTML

Un form (modulo) HTML è costituito da un certo numero di oggetti quali caselle di testo, menu, caselle di spunta e così via. Tali oggetti consentono al visitatore della pagina d'inserire delle informazioni esattamente come nel caso della compilazione di un questionario cartaceo. Una volta inseriti i dati richiesti questi sono inviati, tramite il tasto Submit, ad uno script o ad un'applicazione CGI che li elabora. La URL dell'applicazione è specificata con l'attributo action del tag form.

```
<!-- file form.html -->
...
<form action="/scripts/elabora.php" method="get">
<input type="text" name="campione">
<input type="submit" name="bInvia" value="Invia i dati">
</form>
```

L'attributo method specifica il metodo da utilizzare per l'invio delle informazioni e può essere:

1. GET, le informazioni sono incluse nell'indirizzo URL e pertanto sono visibili nella barra degli indirizzi del browser, ma soprattutto sono vincolate dalla lunghezza massima di una URL, 256 caratteri.
2. POST, le informazioni sono scritte sullo standard input dall'applicazione destinatarie, non sono visibili nella barra degli indirizzi del browser e non ci sono limiti sulla quantità di dati inviata.

Dal punto di vista del programmatore PHP i due metodi sono di fatto equivalenti: l'interprete PHP analizza sia le informazioni passate tramite URL (metodo GET), sia quelle ricevute sullo standard input (metodo POST) e le rende immediatamente disponibili nello script di destinazione sotto forma di variabili globali.

In alternativa, si può accedere alle informazioni inviate con i metodi GET e POST utilizzando gli array associativi $HTTP_GET_VARS e $HTTP_POST_VARS. Ad esempio, se tramite il metodo GET inviamo ad uno script PHP un parametro di nome pagina e di valore pari a uno, all'interno dello script stesso potremo accedere a tale informazione sia usando la variabile globale $p, sia accedendo a $HTTP_GET_VARS["p"], in pratica all'elemento dell'array associativo.

Riprendiamo il form dell'esempio precedente e supponiamo di scrivere nella casella di testo la parola "Cipollini" e di premere il pulsante Invia i dati. Il browser si sposterà all'indirizzo.

http://www.miosito.tld/scripts/elabora.php?campione=Cipollini

Cosa è successo? Poiché il form usa il metodo GET per trasferire i propri dati, questi sono codificati nell'indirizzo dello script cui devono essere inviati. Nello script elabora.php adesso troveremo definita una variabile globale di nome $campione il cui valore è la stringa "Cipollini".

```
// Nel file `elabora.php' ...questo stampa "Cipollini"
echo $campione;
```

A questo punto è utile accennare al modo in cui le informazioni provenienti da un form HTML sono codificate per essere trasmesse con il metodo GET. Partiamo dall'indirizzo URL dello script cui vogliamo inviare i tali dati; ad esso aggiungiamo a destra un punto interrogativo che separerà la URL vera e propria a sinistra dalla query string che andiamo a costruire. La struttura della query string consiste in una serie di coppie nome/valore separate da una "&"; i caratteri non ammissibili in un indirizzo URL, ad esempio gli spazi, sono sostituiti dal simbolo "%" seguito dal corrispondente codice ASCII in esadecimale. Adesso che sappiamo come funziona il metodo GET possiamo sfruttarlo per passare parametri ad uno script PHP. Supponiamo di avere uno script di nome news.php che estrae notizie ed articoli da un database, ad esso vogliamo passare un parametro $argomento, che determinerà il tipo di notizie che ci saranno mostrate. Ad esempio, per ottenere notizie sportive, invocheremo.

```
http://www.miosito.tld/news.php?argomento=Sport
```

Se c'interessano le notizie di attualità e cultura, esiste una funzione PHP urlencode(), che ci permette di codificare una stringa in una forma ammissibile per una URL.

```
<?
echo '<a href="http://www.miosito.tld/news.php?argomento=';
echo urlencode("Attualità e Cultura");
echo '">Clicca qui</a>';
?>
```

INCLUSIONE DI FILE ESTERNI

Possono essere sia dei semplici frammenti di codice HTML sia script PHP: nel primo caso avremo una situazione simile all'uso delle SSI (Server Side Includes); nel secondo, invece, saremo in grado di condividere codice PHP tra più script.
La sintassi è mostrata nell'esempio.

```
// Questa istruzione include ed esegue il file
// 'libreria.php3' contenuto nella directory corrente
require "libreria.php3";
// Usando 'include' avrei scritto
include "libreria.php3";
```

È importante osservare che nel momento in cui PHP inizia ad analizzare il file incluso l'interprete si pone in modalità HTML, quindi eventuali frammenti di codice PHP, per essere correttamente riconosciuti ed eseguiti, dovranno essere racchiusi dai consueti demarcatori. Terminata l'analisi del file esterno si torna in modalità PHP e si continua l'elaborazione dello script principale.
Supponiamo di voler realizzare un sito in cui tutte le pagine devono contenere la classica barra di navigazione con i link alle varie sezioni del sito stesso. Per prima cosa isoliamo il frammento di HTML corrispondente alla barra di navigazione e lo salviamo in un file.

```
<!-- file intestazione.html === INIZIO -->
<body color="black" bgcolor="white">
<a href="index.php3">Home page</a> |
<a href="pagina2.php3">Pagina 2</a> |
<hr size="1">
<!-- file intestazione.html === FINE -->
```

Si noti che c'è anche il tag BODY, in questo modo, tutte le pagine utilizzeranno lo stesso insieme di colori per il testo, lo sfondo.
Realizziamo adesso le varie pagine del sito. Ogni pagina sarà uno script PHP. Una volta

completata una pagina andremo ad inserire il codice PHO necessario per includere il file con la barra di navigazione.

```
<!-- file index.php3 -->
<html>
<head>
<title>Pagina che include un file esterno</title>
</head>
<? require "intestazione.html" ?>
<h1>Pagina che include un file esterno</h1>
(...Contenuto della pagina...)
</body>
</html>
```

Nella pagina index.php3 è stato rimosso il tag BODY in quanto già presente nella barra di navigazione inclusa.
In tutte le pagine PHP costruite in questo modo, l'istruzione

```
<? require "intestazione.html" ?>
```

sarà rimpiazzata dinamicamente dal contenuto del file intestazione.html. Qualunque modifica apportata a tale file sarà immediatamente riflessa in tutte le pagine del sito che la includono rendendone semplice la manutenzione.

UNA BARRA DI NAVIGAZIONE INTELLIGENTE

Costruire una barra intelligente, in grado di evidenziare automaticamente la voce corrispondente alla pagina corrente.
Costruiamo la struttura dati appropriata; in pratica un array associativo in cui avremo tanti elementi quanti sono i link che vogliamo inserire nella barra di navigazione. Ogni elemento consisterà di una chiave e di un valore: la chiave sarà l'indirizzo della pagina web corrispondente, il valore una sua breve descrizione.

```
// Struttura dati: array con link e descrizioni
$links = array( "index.php3" => "Home page",
"pagina2.php3" => "Pagina 2",
"pagina3.php3" => "Pagina 3"
);
```

Quando la barra di navigazione è visualizzata, deve essere evidenziata la voce corrispondente alla pagina in cui ci troviamo. Ci sono due possibilità: possiamo essere noi a stabilire di volta in volta quale voce evidenziare oppure, fare in modo che ciò avvenga in modo automatico, come nel nostro caso. Nell'esempio, confrontiamo il nome del file della pagina corrente con quelli contenuti nell'array $links: la pagina corrente è quella per cui il confronto ha esito positivo. Il percorso dello script PHP corrente è contenuto nella variabile globale $PHP_SELF; poiché c'interessa soltanto il nome del file e non il suo pathname, utilizziamo la funzione "basename" per estrarlo, nel modo seguente.

```
// Nome del file dello script corrente
$pagina_corrente = basename($PHP_SELF);
```

A questo punto scriviamo il codice PHP necessario alla visualizzazione della barra di navigazione, si tratta di spazzolare l'array confrontando ogni indirizzo con quello della pagina corrente e in base all'esito del confronto, rappresentare la voce tra parentesi quadre, con la sola eccezione di quello corrispondente alla pagina corrente, visualizzato in grassetto e senza collegamento ipertestuale.

```
// Visualizzazione barra di navigazione
// Riga orizzontale prima dei link
echo "<hr>\n";
// Inizio a scorrere l'array con list ed each, se si usa PHP4 si può usare foreach
while (list($url,$desc)=each($links)) {
if ($url==$pagina_corrente) {
// Pagina corrente
```

```
echo "<b>$desc</b> ";
} else {
// Altre pagine
echo "[<a href=\"$url\">$desc</a>] ";
}
}
// Riga orizzontale dopo i link
echo "<hr>\n";
```

Per concludere, inseriamo la barra di navigazione nelle pagine del sito.

```
<!-- Questo è il file index.php3 -->
<html>
<head>
<title>Barra di navigazione intelligente</title>
</head>
<body>
<? require "barra.php3" ?>
<h1>Barra di navigazione intelligente</h1>
<h2>Prima pagina</h2>
<p>Questa &egrave; la home page; in alto dovrebbe
essere visibile la barra di navigazione...
</body>
</html>
```

LE DATE

In PHP le date sono rappresentate sotto forma di timestamp; che è un numero intero che corrisponde al numero di secondi trascorsi dalla Unix epoach. Ad esempio, le ore 00:00:00 del primo gennaio 2001 corrispondono al timestamp 978303600.

Per conoscere il timestamp corrispondente ad una certa data basta invocare la funzione mktime() passandole i parametri: ore, minuti, secondi, mese, giorno, anno. Un argomento opzionale tiene conto dell'ora legale.

```
// Timestamp delle ore00:00:00 del primo gennaio 2001
echo mktime(0, 0, 0, 1, 1, 2001);
```

PHP mette a disposizione una forma più leggibile con al funzione date(), che vuole due argomenti: il primo, obbligatorio, è una stringa che rappresenta il formato in cui codificare la data; il secondo, facoltativo, è il timestamp da formattare, se è omesso è considerato il timestamp corrente.

```
// Questa istruzione stampa la data corrente nel formato gg/mm/aaaa
echo "Data di oggi " . date("d/m/Y");
// Come sopra ma senza lo zero prima di giorni e/o mesi di una sola cifra
echo "Data di oggi " . date("j/n/Y");
echo "Sono trascorsi ".date("z")." giorni dall'inizio dell'anno.";
```

Rappresentazione della data corrente in modo testuale.

```
// Nomi dei giomi della settimana
$giomi = array( "Dom", "Lun", "Mar", "Mer", "Gio", "Ven", "Sab" );
echo "Oggi e': " . $giorni[date("w")];
```

La funzione getdate() restituisce informazioni sulla data e ora correnti in un array associativo.

Vediamo come sia possibile validare una data, in pratica accertarsi che si tratti di una data valida; per esempio, sono date non valide il 31 aprile o il 29 febbraio di un anno non bisestile. La funzione usata è checkdate().

```
// Verifichiamo una data (31 aprile 2001!?)
$giomo = 31;
$mese = 4;
$anno = 2001;
```

```
echo "La data $giorno/$mese/$anno ";
if (checkdate($mese,$giorno,$anno)) echo "è corretta.";
else echo "non è valida!";
```
Supponiamo di voler confrontare due date per verificare se la prima è antecedente alla seconda: basta convertire il tutto in timestamp.
```
// Data n° 1: ancora il primo gennaio 2001
$data1 = mktime(0, 0, 0, 1, 1, 2001, 0);
// Data n° 2: il 29 luglio 2001
$data2 = mktime(0, 0, 0, 7, 29, 2001, 0);
echo "La prima data è ";
if ($data1 < $data2) echo "precedente";
else echo "successiva";
echo " alla seconda.";
```
Calcolare il numero di giorni che separa due date arbitrarie.
```
// Data n° 1: ancora il primo gennaio 2001
$data1 = mktime(0, 0, 0, 1, 1, 2001, 0);
// Data n° 2: il 29 luglio 2001
$data2 = mktime(0, 0, 0, 7, 29, 2001, 0);
echo "Tra le due date ci sono ";
echo ($data2 - $data1)/(60*60*24);
echo " giorni.";
```
La differenza tra i due timestamp, espressa in secondi, può essere convertita in giorni dividendo per 24 ore al giorno, 60 minuti all'ora, 60 secondi al minuto. L'unica particolarità riguarda l'ultimo parametro fornito a mktime(), impostato a zero in modo da ignorare l'ora legale.

I COOKIES (BISCOTTI)

Meccanismo mediante il quale le applicazioni lato server possono memorizzare e recuperare informazioni sul lato client (rappresentato dal browser).
Tutte le operazioni di scrittura, modifica o cancellazione di cookies in PHP avvengono mediante una funzione: setcookie(), questa funzione deve obbligatoriamente essere invocata prima che qualsiasi contenuto sia inviato al browser; i cookies, infatti, sono trasmessi tra client e server sotto forma d'intestazione (header) Http.
```
setcookie(Nome, Valore, Espirazione, Percorso, Dominio, Secure);
```
Vediamo di chiarire le opzioni che si possono passare alla funzione:

1. Nome è il nome del cookie, che può essere arbitrariamente scelto;
2. Valore è il valore, anch'esso arbitrario, da assegnare al cookie;
3. Espirazione è la data di espirazione del cookie;
4. Percorso è la directory, a partire dal dominio (vedi sotto) per la quale il cookie è valido;
5. Dominio è il dominio per il quale il dominio è valido;
6. Secure è un valore che imposta se il cookie debba essere inviato tramite una connessione HTTPS.

Poniamo di voler inviare dalla nostra pagina un cookie chiamato "Test", con valore "Prova per il cookie Test", con espirazione di un minuto dal momento dell'invio, per la directory "/nomeutente" del dominio "http://www.dominio.com" senza utilizzare una connessione HTTPS: i parametri da passare a setcookie sono:
```
setcookie("Test", "Prova per il cookie Test", time()+60, "/nomeutente", ".dominio.com", 0);
```
Le cose interessanti da esaminare sono due: il tempo di espirazione e la connessione HTTPS. Per la seconda, impostiamo con "0" che la connessione deve essere una normale connessione HTTP; se volessimo utilizzare il protocollo sicuro HTTPS, dovremo inserire il valore "1". Il tempo di espirazione non può essere impostato come "Tre minuti", poiché la cosa non ha tempo: dobbiamo invece impostare il momento di espirazione a partire dal momento in cui il cookie è inviato all'utente; per questo, utilizziamo la funzione interna time() alla quale aggiungiamo il numero di secondi dopo i quali il cookie deve scadere. Ad

esempio, se il cookie è inviato alle 13:20:00 e vogliamo che esso espiri dopo 30 minuti (60x30=1800 secondi) possiamo scrivere come espressione di expire:
time()+1800
In questo modo, time() riporta le 13:20:00 e time()+1800 sarà 13:50:00
La funzione prevede solo due argomenti obbligatori: il nome da assegnare al cookie ed il suo valore. Ad esempio, se vogliamo memorizzare nel browser di un visitatore un cookie, che chiameremo $nomeutente, contenente una stringa "latoserver"", l'istruzione da utilizzare sarà la seguente.

```
// Imposto un cookie: $nomeutente = "latoserver.it";
setcookie( "nomeutente", "latoserver.it" );
```

È possibile specificare anche altri argomenti (facoltativi); nell'ordine abbiamo: scadenza del cookie, percorso, dominio, e secure. Quando un cookie è stato impostato, il suo valore può essere modificato richiamando nuovamente la stessa funzione ed associando allo stesso nome il nuovo valore. La cancellazione di un cookie, infine, può avvenire in due modi: assegnandogli un valore nullo o impostando la scadenza ad una data passata. In quale modo si possono leggere da uno script PHP le informazioni memorizzate nei cookies: l'interprete PHP analizza automaticamente i cookies inviati dal browser e li rende disponibili ad altrettante variabili globali e nell'array associativo $HTTP_COOKIE_VARS.
Progettiamo una pagina PHP che ricorda la data e l'ora dell'ultimo accesso del visitatore. A tale scopo impostiamo sul browser un cookie $ultimavisita, non è stata specificata alcuna scadenza, in modo da far scomparire il cookie alla chiusura del browser. Il valore assegnato di volta in volta al cookie $ultimavisita è il risultato della funzione time() e consiste nel numero di secondi trascorsi dalla Unix epoch (01/01/1970).

```
<?php
// file `saluto.php'
// Il saluto predefinito
$saluto = "Benvenuto!";
// Controllo se esiste il cookie...
if (isset($HTTP_COOKIE_VARS["ultimavisita"])) {
// Cambio il saluto con uno più appropriato
$saluto = "Bentornato!";
}
// Imposto il cookie relativo a questa visita
setcookie( "ultimavisita", time() );
?>
<html>
<head>
<title><? echo $saluto ?></title>
</head>
<body>
<h1><? echo $saluto ?></h1>
<?php
if (isset($HTTP_COOKIE_VARS["ultimavisita"])) {
// Stampo la data dell'ultima visita
echo "L'ultima volta sei stato qui il " . date( "d/m/Y");
echo " alle ore " . date( "H:i:s.", $ultimavisita );
// Link per cancellare il cookie
echo "<p><a href=\"cancella.php\">Cancella il cookie</a>";
} else {
echo "Non sei mai stato qui prima?";
}
?>
</body>
</html>
```

Lo script cancella.php, richiamabile cliccando sul link “Cancella il cookie”, non fa altro che cancellare il cookie $ultimavisita, assegnandogli un valore nullo, e dirottare il browser di nuovo alla pagina precedente.

```
<?
// file `cancella.php'
setcookie( "ultimavisita", "" );
header( "Location: saluto.php" );
?>
```

Per finire, vediamo come leggere un determinato cookie da uno script: ricordando che un cookie è inviato come un array di dati, possiamo ricorrere alle funzioni relative agli array.

```
if (isset ($HTTP_COOKIE_VARS) )
{ while (list ($nome, $valore) = each ( $HTTP_COOKIE_VARS ) )
{ echo "$nome = $valore\n";
}
}
```

Quindi, per tutte le informazioni inserite nel cookie, sarà visualizzata una coppia "nome = valore": ovviamente, il valore è il valore che avete impostato nel cookie, i "nome" sono: cookie, expires, path, domain e secure (quest'ultimo è particolare, visto che se impostato su 1 fa apparire "secure" nel cookie, altrimenti non fa apparire alcunché). Per il cookie di esempio, leggeremo qualcosa del genere.

```
cookie=Test; epires=Thursday, expires=Monday, 31-Jul-00 11:50:00 GMT;
path=/nomeutente; domain=.dominio.com
```

Se avessimo impostato 1 anziché 0 per il protocollo HTTPS, sarebbe apparso anche "secure" alla fine della stringa. La data è gestita di default secondo l'orario GMT: se notate, infatti, l'espirazione è impostata per le 11:50:00 GMT, che sono le 13:50:00 locali.

LE SESSIONI

Il meccanismo dei cookies presenta delle limitazioni; per esempio, mantenere uno stato anche nell’eventualità in cui il browser del visitatore non supporti i cookies o li abbia disabilitati.

Una sessione consiste in una serie di accessi alle nostre pagine PHP, effettuati in un determinato arco di tempo durante il quale è mantenuto uno stato. Ogni sessione è individuata da un identificatore, univoco, utilizzato per l’associazione tra client e relativa sessione. Per utilizzare le sessioni in PHP 4 si usano tre funzioni: session_start(), session_register(), session_destroy(). Con le versioni precedenti si usa la libreria PHPLIB.

La prima funzione, session_start() è invocata per creare una nuova sessione o per ripristinarla se creata in precedenza; tenta anche d’impostare , nel browser, un cookie contenente l’identificativo di sessione, per cui è necessario che sia invocata all’inizio degli script, come setcookie().

La funzione, session_register() è usata per registrare delle variabili come variabili di sessione. Ad esempio, se abbiamo una variabile $nomeutente e vogliamo renderla persistente per tutta la durata della sessione si lavora nel modo seguente.

```
// $nomeutente diventa variabile di sessione
session_register("nomeutente");
```

La funzione, session_destroy() è invocata per distruggere i dati relativi alla sessione, al log out.

Il programmatore deve prestare attenzione alla propagazione del SID (Identificatore di Sessione). In generale, è sufficiente memorizzare tale valore in un cookie nel browser del visitatore, PHP lo fa automaticamente, ma nel caso in cui i cookies non possono essere utilizzati perché il browser non li supporta o è stato configurato per rifiutarli, è il programmatore che deve farsi carico della propagazione del SID, modificando i link tra i vari script che condividono la sessione e passandolo come parametro.

```
<!--
Un esempio di link che propaga l'identificativo di sessione senza richiedere cookies
```

```
-->
<a href="altroscript.php?<?= SID ?>">Altro script</a>
```
Progettare uno script che propone al visitatore di cliccare su uno dei tre colori disponibili (bianco, rosso e verde), mostrando al contempo la sequenza delle selezioni effettuate fino a quel momento.
La sequenza dei clic sui vari colori è memorizzata in un array, registrato come variabile di sessione. La prima cosa da fare è l'avvio della sessione.
```
// Attivo (o ripristino) la sessione
session_start();
```
Il secondo passo è la registrazione dell'array $clicks come variabile di sessione.
```
// 'clicks' è una variabile di sessione: devo registrarla
session_register("clicks");
```
Passiamo ai parametri passati dall'utente. Per prima cosa verifichiamo se è stato cliccato il link "Ricomincia da capo", che comporta l'azzeramento della sequenza (dell'array).
```
// Devo azzerare?
if ($azzera) {
$clicks = array();
}
```
Se l'utente ha cliccato su uno dei colori lo aggiungiamo alla sequenza, inserendo un nuovo elemento in coda all'array.
```
if ($click) {
$clicks[] = $click;
}
```
Infine, arriviamo alla visualizzazione della sequenza dei colori. Se l'array che rappresenta la sequenza contiene almeno un elemento, quindi se è stato scelto almeno un colore, sono visualizzati tutti i suoi elementi, diversamente è mostrata la scritta "sequenza vuota".
```
if (count($clicks)) {
foreach ($clicks as $colore) { echo "$colore "; }
} else {
echo "(sequenza vuota)";
}
```

AUTENTICAZIONE

Con il PHP e l'ausilio della sua funzione "header" è possibile gestire senza la necessità di utilizzare i file .htaccess o comunque le autorizzazioni del server le pagine protette. Basterà semplicemente inviare un header preciso al browser prima di passargli l'output per far si che, all'interno di una pagina, siano richiesti username e password per l'accesso. Intanto, copiate questo codice in un file e richiamatelo via browser.
```
$username = "pippo";
$pwd = "segreta";
if(!isset($PHP_AUTH_USER)) {
Header("WWW-Authenticate: Basic realm=\"Zona protetta\"");
Header("HTTP/1.0 401 Unauthorized");
echo "Impossibile eseguire l'autorizzazione\n";
exit;
} else {
if (($PHP_AUTH_USER == $username) && ($PHP_AUTH_PW == $pwd)) {
echo "Autorizzazione riuscita per $username.";
} else { echo "Autorizzazione fallita.";}
}
```
In questo modo, impostiamo "pippo" come username e "segreta" come password: se i dati inseriti dall'utente nella maschera di autorizzazione coincidono con questi, l'autorizzazione riesce, altrimenti fallisce.
Ma come avviene tutto questo?

Vediamo come funziona lo script appena presentato. Prima di tutto, questo controlla se sia impostata la variabile PHP_AUTH_USER, ossia l'username dell'utente che accede alla pagina protetta. Se questa esiste, probabilmente l'utente si è precedentemente autorizzato e, per questo motivo, essa è "ricordata"; In ogni caso, sia che l'utente si sia precedentemente autorizzato sia che sia la sua prima visita alla pagina, è controllato che l'username e la password precedentemente inserite o inserite al momento nella maschera di autorizzazione siano esattamente $username e $pwd: se combaciano, è possibile visualizzare l'output per l'avvenuta autorizzazione (quello che abbiamo impostato come "Autorizzazione riuscita per $username"); nel caso contrario, è visualizzato un messaggio d'errore (nel nostro caso "Autorizzazione fallita). Se invece non esiste $PHP_AUTH_USER, è inviato un header per l'autorizzazione (il codice 401) che presenta all'utente la maschera che gli richiede username e password: una volta inserite, sarà eseguito il blocco descritto sopra e, se l'utente decide di non autorizzarsi, gli è presentata una pagina contenente il testo "Impossibile eseguire l'autorizzazione", che è comunque personalizzabile. La funzione che controlla tutto questo meccanismo è "header()", utilizzata per mandare un header al browser che ha inviato la richiesta; essendo quello che è inviato un header, è necessario che questo sia inviato prima di qualsiasi altro output. La funzione è molto semplice, vediamone degli esempi: se volessimo inviare un header che rediriga il browser ad una determinata locazione.

header("Location: http://www.html.it");

Se invece volessimo inviare un messaggio di pagina inesistente oppure un'istruzione per fare in modo che il browser non utilizzi la cache, potremo scrivere.

header("http:/1.0 404 Not Found");
header("Pragma: no-cache");

PHPMYADMIN

INTRODUZIONE

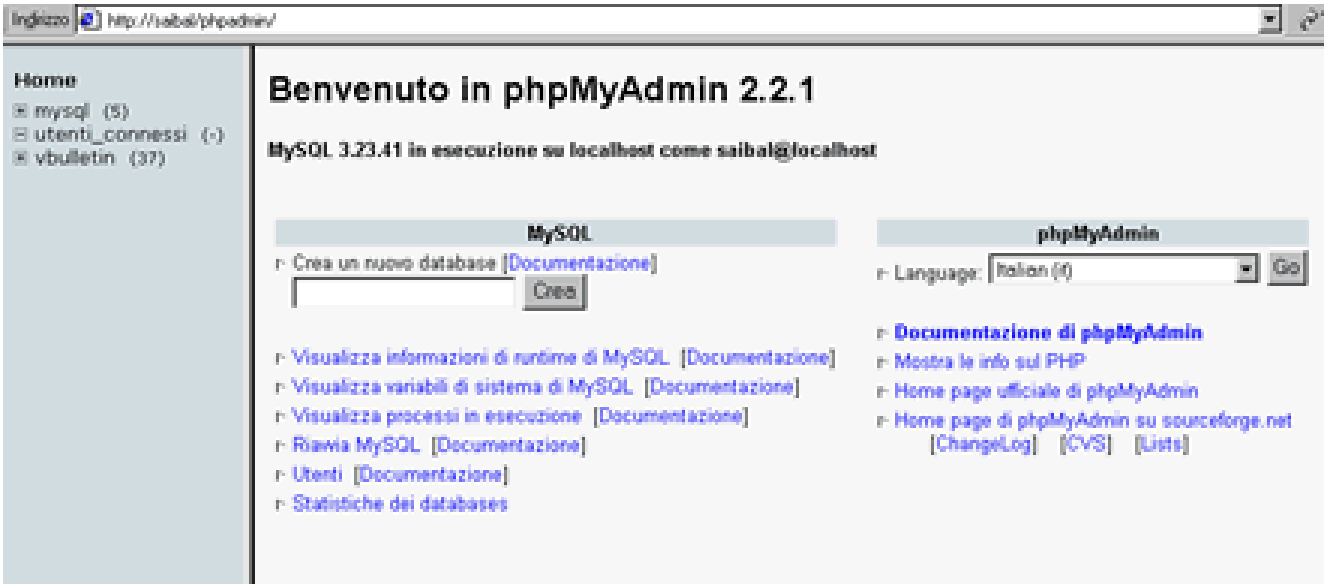
Guida pratica all'utilizzo di **PHPMyAdmin** per installare uno script che richieda MySQL. Prima di cominciare vediamo di cosa abbiamo bisogno:

- un server web
- il modulo PHP
- MySQL
- PHPMyAdmin

PHPMyAdmin non è altro che un'interfaccia grafica che permette di amministrare MySQL, un tipo di database che immagazzina qualsiasi tipo di dati in strutture chiamate tabelle; con PHPMyAdmin, in pratica, possiamo visualizzare il contenuto del database; creare, modificare, cancellare intere tabelle o singoli record; fare un backup dei dati contenuti; visualizzare informazioni interessanti sul DB.
Le tabelle sono dei "contenitori" di dati (chiamati record); che possono avere diversi tipi di struttura, si trovano all'interno di archivi molto grossi (i singoli database appunto); ogni database, infine, può contenere diverse *tables* purché queste non abbiano lo stesso nome.

INTERFACCIA DI PHPMYADMIN

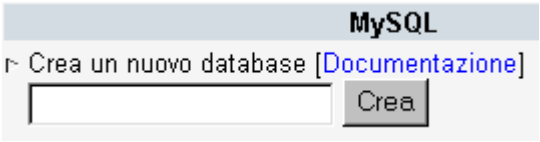
Attiviamo Apache, il database MySQL ed accediamo a PHPMyAdmin digitando l'URL corrispettivo:
http://localhost/PHPadmin/index.php
Ci troviamo di fronte ad una pagina composta da due frames; nella colonna di sinistra, sotto la scritta **Home**, ci sono i nomi di tutti i database creati; se è la prima volta che attivate MySQL dovreste visualizzarne solo due: *mysql* e *test* (**MySQL non va assolutamente toccato perché contiene dati importanti per il funzionamento del DB**).
Nella pagina centrale ci sono le risorse principali; abbiamo, ad esempio, il form per creare un nuovo DB; la scritta **Utenti** per impostare nuovi users; il collegamento per riavviare MySQL ed una serie di link per visualizzare alcune informazioni statistiche; sono presenti, infine, interessanti collegamenti alla documentazione ufficiale.



CREARE UN DATABASE

Se si ha la possibilità di farlo, perché il nostro host lo permette, si potrebbe creare un DB per ogni nuovo codice installato in modo da avere tutti i dati suddivisi ed ordinati; sappiate

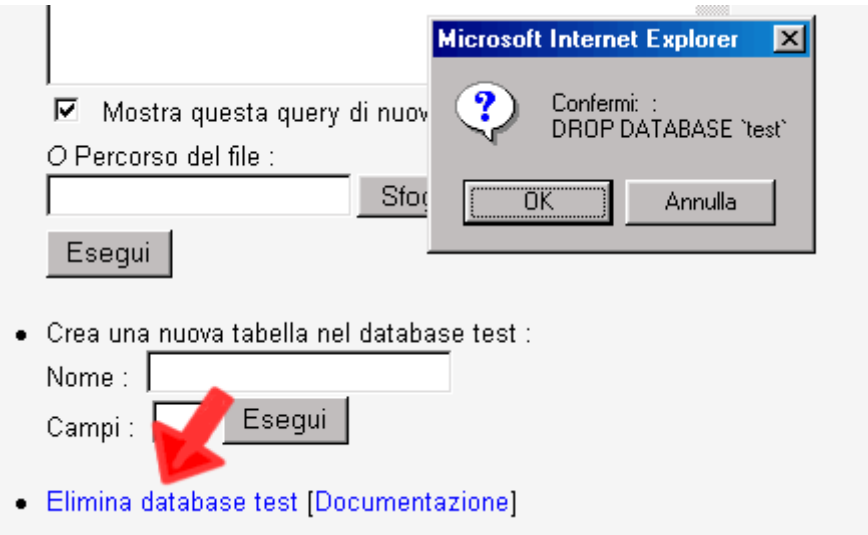
in ogni modo che è possibile installare diversi script sullo stesso DB purché non ci siano tabelle con lo stesso nome.
Ci dovremo abituare a questa situazione quando agiremo sul nostro spazio remoto dato che la maggior parte dei provider offrono, per ciascun utente, un solo database su cui lavorare. Trovandoci in locale, invece, non abbiamo questa restrizione.
Finita questa piccola premessa continuiamo con le operazioni di creazione. Facciamo riferimento al frame centrale: nel campo di testo sotto la scritta **Crea un nuovo database [Documentazione]** possiamo inserire il nome del nuovo DB. In questo caso, data la tipologia dello script che installeremo, suggerisco **utenti_connessi** (non dobbiamo mai usare nomi staccati).



Una volta scritto il titolo possiamo premere sul bottone **Crea** e, "magicamente" (nel frame di sinistra), apparirà il DB che abbiamo appena costruito; noterete anche la presenza di un segno **-** (meno) che sta ad indicare l'assenza di tabelle all'interno di **utenti_connessi**. Viceversa il segno **+** indica la presenza di tabelle all'interno di un database, come possiamo facilmente vedere facendo riferimento a quello chiamato **MySQL**.

CANCELLARE UN DATABASE

Operazione inversa alla precedente. Clicchiamo una volta sul nome del DB da cancellare; se ci fossero delle tabelle al suo interno, queste sarebbero visualizzate sia nel frame di sinistra che nella pagina centrale; in questo caso, invece, *test* è vuoto. Avrete sicuramente notato come, una volta selezionato il DB, sia cambiato il contenuto del frame principale; in fondo a questa stessa pagina troverete un link molto interessante: **Elimina database test [Documentazione]**. È facile intuire la funzione di questo comando; clicchiamo senza paura sul link, diamo la conferma quando ci sarà richiesto e *test* sarà cancellato definitivamente. Ovviamente, per non completare l'operazione, clicchiamo su "**ANNULLA**".



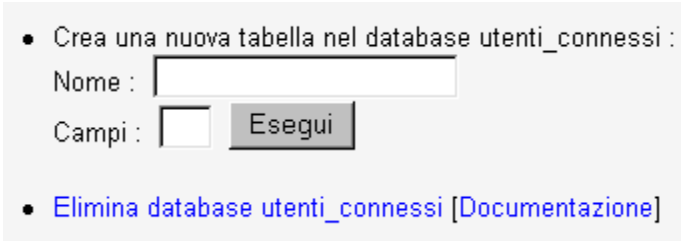
CREAZIONE DI UNA TABELLA

Abbiamo detto che ogni DB contiene delle tabelle dove sono immagazzinati i dati. Le

tabelle, che dobbiamo creare noi, possono avere numerosi campi al loro interno ed ogni campo ha specifiche caratteristiche, ad esempio in base alla grandezza dei record da immettere.
Per creare le tabelle con PHPMyAdmin esistono fondamentalmente due modi:

1. manuale in cui dovremo impostare ogni singolo campo a mano;
2. basterà creare un file con estensione **.sql** e caricarlo sul server.

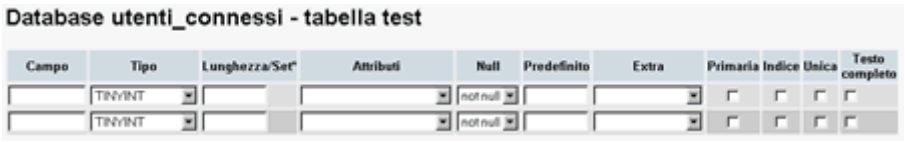
Inizialmente studieremo il primo procedimento: clicchiamo sul database **utenti_connessi**; nel frame centrale, a fondo pagina, avremo: **Crea una nuova tabella nel database utenti_connessi:**



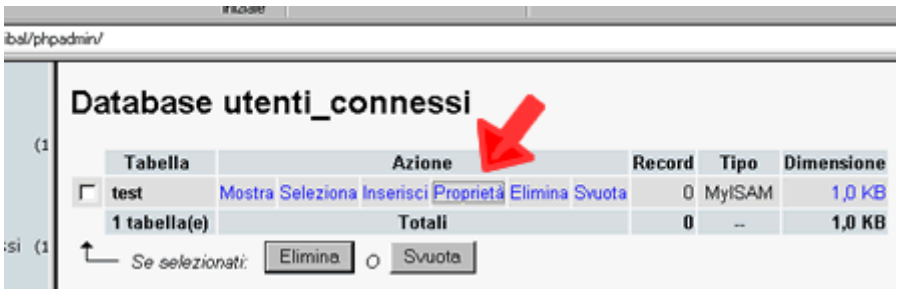
I due text field richiedono il nome da dare alla tabella e il numero di campi che questa tabella dovrà avere. Supponiamo di voler creare un esempio di prova. Immettiamo i seguenti dati:

- Nome : **test**
- Campi : **2**

Clicchiamo su **Esegui** e, se tutto è andato come dovrebbe, PHPMyAdmin visualizzerà la struttura dei campi (2) della tabella (test):

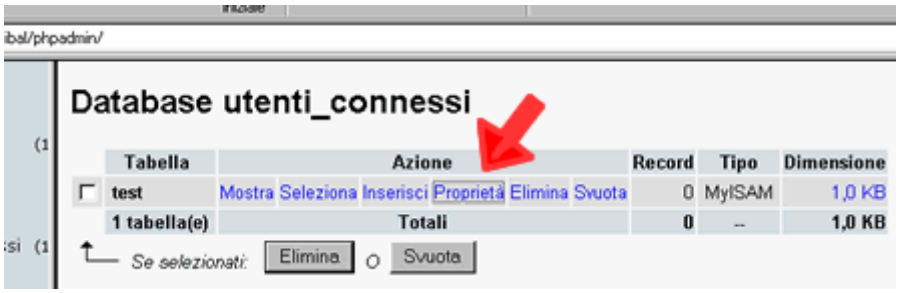


Visto che stiamo facendo delle prove riempiano le voci "**Campo**", nella schermata poco fa visualizzata, con due nomi a piacere: **prova1** e **prova2**. Fatto questo clicchiamo su **Salva** per completare l'operazione.
Per avere un riassunto della situazione selezioniamo (nel frame di sinistra) il nostro DB (utenti_connessi); nel frame centrale avremo tutte le tabelle presenti all'interno del DB selezionato (in questo caso una sola); clicchiamo su **Proprietà** per vedere in dettaglio la composizione della tabella *test*:



CANCELLARE O MODIFICARE UNA TABELLA

Facciamo riferimento sempre all'immagine successiva; notiamo che accanto alla voce "**Proprietà**" ci sono altri link per la gestione delle tabelle: analizziamoli velocemente.



- **Mostra**: mostra il contenuto della tabella selezionata
- **Seleziona**: permette delle operazioni sui campi selezionati dalla tabella: ad esempio possiamo cambiare l'ordine di visualizzazione dei campi
- **Inserisci**: dà la possibilità di inserire manualmente i record nei campi della tabella
- **Proprietà**: mostra i singoli campi della tabella dando la possibilità di cancellare o modificare il loro contenuto
- **Elimina**: elimina la tabella selezionata
- **Svuota**: cancella i record all'interno della tabella senza eliminarla.

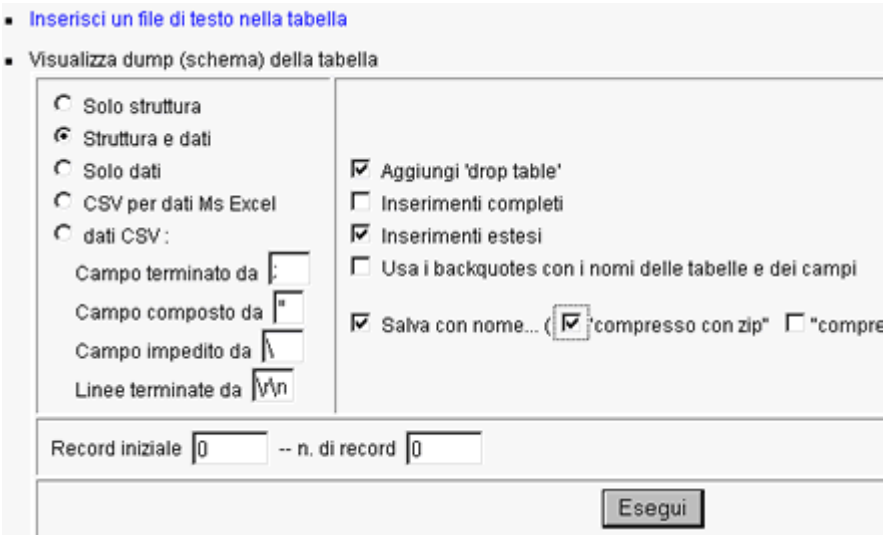
CANCELLA O MODIFICARE I DATI DI UN CAMPO

Nel caso avessimo commesso uno sbaglio nel compilare un campo abbiamo la possibilità di correggere gli errori senza dover ripartire da zero. Supponiamo di voler cambiare il nome del primo campo da **prova1** a **prova**. Clicchiamo, ancora una volta, sul nome del DB nel frame sinistro; dopo che il menu si sarà espanso, mostrando le tabelle presenti all'interno di *utenti_connessi*, selezioniamo la tabella interessata (in questo caso *test*). Avremo davanti una schermata di questo tipo:

Selezioniamo **Modifica** riferito a *prova1* e, nella schermata successiva, trasformiamo il contenuto di "**Campo**" da *prova1* a *prova*; premiamo "**Salva**" per aggiornare le modifiche. Questo stesso procedimento può essere compiuto anche per le modifiche agli altri campi della tabella (non solo al nome).

EFFETTUARE UNA COPIA DI BACKUP DEL DATABASE

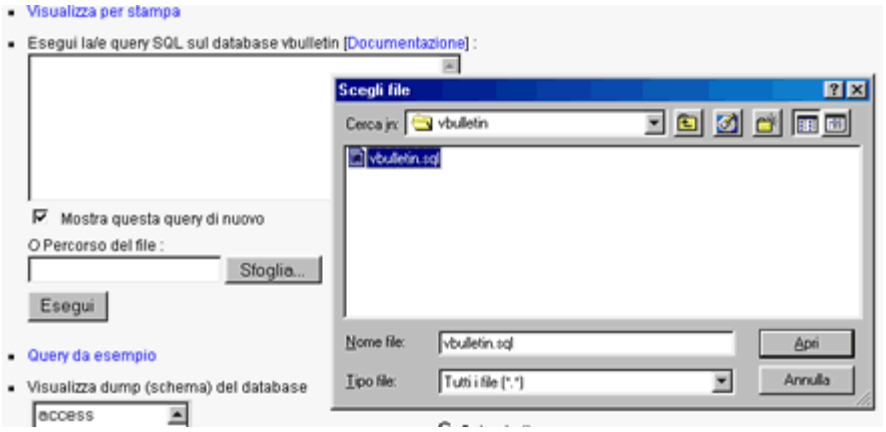
Per scaricare sul nostro PC una copia dei dati basta cliccare (nel frame di sinistra) sul nome del DB che vogliamo salvare; fatto questo cerchiamo, nella parte centrale della pagina, la scritta **Visualizza dump (schema) del database** e spuntiamo i checkbox come da figura:



Una volta cliccato su "**Esegui**" inizierà il salvataggio (nel formato scelto) del nostro database in modo da poter custodire una copia di sicurezza per ogni evenienza.

Reinstallare una copia di backup del database

Supponiamo, adesso, di dover reinstallare un backup del DB perché i dati sono andati persi. Prima di tutto scompattiamo il file zip che avevamo precedentemente scaricato; dentro ci sarà un file con estensione **.sql** che rappresenta, appunto, il backup effettuato. Adesso, nel frame di sinistra, clicchiamo sul nome del DB che vogliamo ripristinare (se questo non esiste più basterà ricrearlo). Nella schermata centrale, poco sotto il link **Visualizza per stampa**, ci sarà un text field con accanto il bottone "**Sfoglia**". Andiamo a cercare il file .sql come da figura:



e clicchiamo su "**Esegui**". Il tempo d'importazione varierà a seconda della grandezza del file e alla fine del processo dovremmo ricevere questo messaggio:
Database xxxx
La query è stata eseguita con successo:
Il contenuto del file è stato inserito. (xxx Istruzioni)

Creazione della tabella con file .sql

La stessa procedura per reinstallare un backup è valida anche per importare un file **.sql** adatto al funzionamento di uno script. Riprendiamo, quindi, il discorso sulla creazione delle tabelle lasciato a metà in precedenza; avevamo detto che, oltre alla possibilità di crearle a mano, si può anche importare direttamente un file a patto che questo sia scritto in un certo modo. Dovete sapere, infatti, che molti script (specialmente i più complessi) hanno un file .sql che permette di creare la struttura del database senza dover impostare ogni singola tabella a mano. L'utente non dovrà fare altro che scegliere il DB appropriato e uploadare il

file su MySQL. Ad esempio vi avrei reso la vita più facile se avessi incluso nello zip dello script utenti, un documento **useronline.sql** strutturato così:

```
# PHPMyAdmin MySQL-Dump
# version 2.2.1
# http://PHPwizard.net/PHPMyAdmin/
# http://PHPmyadmin.sourceforge.net/ (download page)
# Host: localhost
# Generato il: 10 Dic, 2001 at 05:07 PM
# Versione MySQL: 3.23.41
# Versione PHP: 4.0.6
# Database : `utenti_connessi`
# --------------------------------------------------------
# Struttura della tabella `useronline`
CREATE TABLE useronline (
zeit int(15) NOT NULL default '0',
ip varchar(15) NOT NULL default '',
file varchar(50) NOT NULL default '',
PRIMARY KEY (zeit),
KEY ip (ip),
KEY file (file)
)
# Dump dei dati per la tabella `useronline`
```

Avendo a disposizione un file di questo tipo in pochissimo tempo avremmo impostato in modo corretto la tabella; sarebbe stato sufficiente uploadare **useronline.sql** nel database **utenti_connessi** senza tutti i passaggi che eseguiremo prossimamente.

Installazione dello script

Adesso che abbiamo familiarizzato con PHPMyAdmin possiamo passare all'installazione dello script. Visto che ormai sappiamo come fare, cancelliamo la tabella di prova che avevamo creato all'interno del db **utenti_connessi** in modo da lasciarlo vuoto. Scompattiamo lo zip dello script; al suo interno troveremo tre file oltre ad una cartella contenente il documento per configurare l'accesso al database MySQL. Apriamo, con un qualsiasi editor di testo, il file **dbconf.php** situato dentro la cartella *include*.
La prima parte del documento contiene questo codice:

```
//Dati di accesso al database
//host
$db_server = "localhost";
//username
$db_username = "root";
//password
$db_passwort = "";
//nome del database
$db_database = "utenti_connessi";
//nome della tabella
$db_table = "useronline";
```

Se stiamo lavorando in locale probabilmente non, avremo bisogno di toccare nulla. Nel caso lavoriate in remoto (oppure abbiate cancellato l'utente "root") dovrete personalizzare i dati di accesso. Il resto dello script non va toccato; quello che invece dobbiamo fare è creare a mano la tabella "**useronline**" all'interno del database "**utenti_connessi**".

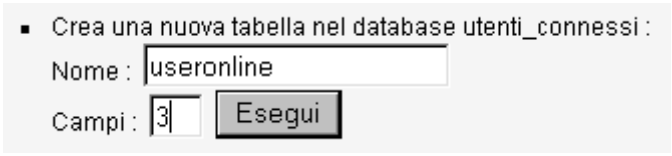
Creazione della tabella "useronline"

La tabella necessaria al funzionamento dello script ha questa struttura:

```
CREATE TABLE useronline (
zeit int(15) NOT NULL,
```

ip varchar(15) NOT NULL,
file varchar(50) NOT NULL,
PRIMARY KEY (zeit),
KEY ip (ip),
KEY file (file));
Da una prima occhiata possiamo vedere che sono necessari tre campi: *zeit*, *ip* e *file*, ognuno con determinate caratteristiche. Vediamo passo passo come impostare il tutto:

- Clicchiamo, nel frame sinistro, sul db "**utenti_connessi**";
- Nel frame centrale (in basso) creiamo una tabella chiamata "**useronline**" con 3 campi;
- Premiamo "**Esegui**";



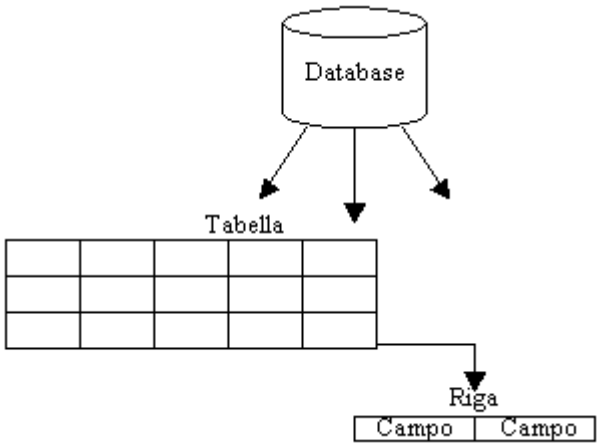
Nella schermata successiva abbiamo la struttura dei tre campi ancora da riempire.
Campo zeit
Alla voce "**Campo**" della prima riga scriviamo *zeit*; dal menù a tendina "**Tipo**" scegliamo *INT*; nel campo "**Lunghezza**" inseriamo 15; il campo "**Null**" lasciamolo su *Notnull*; infine spuntiamo il checkbox della voce "**Primaria**".
Campo ip
Nel primo campo inseriamo *ip*; dal menù a tendina "**Tipo**" scegliamo *VARCHAR*; nel campo "**Lunghezza**" inseriamo *15*; il campo "**Null**" lasciamolo su *Notnull*; spuntiamo il checkbox della voce "**Indice**".
Campo file
Nel primo campo scriviamo *file*; dal menù a tendina "**Tipo**" scegliamo *VARCHAR*; nel campo "**Lunghezza**" inseriamo *50*; il campo "**Null**" lasciamolo su *Notnull*; spuntiamo il checkbox della voce "**Indice**".
Adesso premiamo "**Salva**" per completare l'operazione.

TEST DELLO SCRIPT

La configurazione del database è completa. Spostiamoci dentro la cartella di lavoro di Apache (quella in cui solitamente mettiamo i documenti da testare) e mettiamoci dentro la cartella dello script (è chiamata *utenti*) con tutto il suo contenuto: il folder **include**, il file **useronline.php** e il file **userin. php**.
Creiamo due semplici pagine con estensione .php: la prima sarà chiamata *index.php* e la seconda *prova.php* (o qualsiasi altro nome).
L'*index* conterrà solamente questa stringa:
<? include ("useronline.php"); ?>
La pagina *prova*, invece, avrà questa inclusione:
<? include ("userin.php"); ?>
Salviamo le due pagine appena create dentro la cartella **utenti** e, dopo aver attivato MySQL e Apache, digitiamo l'URL che porta all'index:
http://localhost/utenti/index.php
Sperando che abbiate seguito alla lettera le mie istruzioni dovrebbe apparire questa tabella:

Ecco qui: adesso possiamo visualizzare quanti utenti accedono al nostro sito.
Prima di concludere una breve spiegazione sullo script. Il file **useronline.php** va incluso nella pagina in cui si vuole visualizzare la tabella che mostra gli utenti on line. La grafica della tabella stessa può essere modificata agendo sul file dbconf.php all'interno della cartella *include*. Il file **userin.php**, invece, va inserito in tutte le pagine del sito che si vogliono includere nel conteggio degli utenti. Se infatti un altro utente, con un IP diverso dal nostro quindi, si collegasse alla pagina **prova.php**, noteremmo che, "refreshando" l'index.php, il contatore è aumentato di un'unità. Va da sè che, solitamente, tutte le pagine di un sito non stanno nella stessa cartella; ecco quindi che, per facilitare le cose, consiglio di includere il file **userin.php** con percorso assoluto e non relativo:
<? include ("http://www.vostro_sito.it/utenti/userin.php"); ?>

ORGANIZZAZIONE INTERNA



Un database è organizzato in tabelle che possono avere svariate colonne organizzate in righe. I nostri dati sono inseriti nei campi con una catalogazione ordinata per colonne. In un'ipotetica Mailing List costruiremo una tabella con una colonna chiamata e-mail ed all'interno dei campi di quella colonna potremo inserire tutti gli indirizzi dei nostri utenti. Durante l'inserimento, se la colonna è presente MySQL creerà una nuova riga per ogni dato che noi inseriamo.
Un database articolato potrà avere più tabelle organizzate in più colonne collegate tra loro. Per accedere al database, inserire i dati, modificarli e qualsiasi altra azione utilizzeremo le funzioni che PHP ci mette a disposizione nel suo corredo di funzioni. Assieme a PHP utilizzeremo SQL che sarà il linguaggio che utilizzeremo per interfacciare PHP con MySQL. Possiamo anticipare già da ora come deve essere organizzata una connessione al database. Per poter prendere i dati dal nostro database occorre stabilire anzitutto una connessione con questo, pensando ancora una volta a MySQL come una stanza contenente tutti i nostri dati quando instauriamo una connessione non stiamo facendo altro che aprire la porta di questa ipotetica stanza. Una volta connessi possiamo inserire, modificare, cancellare i dati, le colonne, le tabelle del database. Quando finiamo di fare tutte le operazioni dobbiamo chiudere la connessione e liberare le risorse del sistema per altri utenti.

PRIMA FASE: CONNESSIONE

La prima cosa da fare è la connessione al MySQL. A questo riguardo PHP ci fornisce una funzione apposita: **MySQL_connect**.
La sintassi della funzione di connessione è la seguente:

```
<?php
mysql_connect (nome_host,nome_utente,password);
?>
```

Il nome utente e la password sono indispensabili perché MySQL è un database molto sicuro e non si può accedere ad esso senza aver prima creato un account.
Se state provando MySQL in locale potete tranquillamente lasciare vuoti questi due campi. Se invece dovete provare i vostri script nel vostro sito dovete richiedere lo user e la password ai vostri amministratori di sistema. Per quanto riguarda il nome dell'host esso è quasi sempre localhost, per maggiore sicurezza potete richiedere anche questo dato ai vostri amministratori di sistema.
Vediamo uno script completo per la connessione.

```
<?php
// script per la connessione a MySQL
$host = 'localhost';
$db = ‘prova’;
$user = 'vostro_user';
$password = 'vostra_password';
$conn=mysql_connect ($host,$user,$password) or die ("Non riesco a connettermi");
print "Connessione eseguita";
// fine script di connessione
?>
```

La funzione "die" ci permette di stampare a video la scritta compresa tra virgolette nel caso la connessione non vada a buon fine. Nel caso PHP non riesca a connettersi a MySQL vedremo la scritta "Non riesco a connettermi al database", nel caso contrario vedremo la scritta "Connessione eseguita".
Abbiamo fatto per metà la prima fase, ora per manipolare i nostri dati dobbiamo selezionare il database da utilizzare. MySQL permette, per ogni account, illimitati database. Se non abbiamo ancora creato nessun database dobbiamo prima crearlo e poi selezionarlo, utilizzeremo quindi varie funzioni PHP nel prossimo script.

```
<?php
// script per la connessione, creazione e selezione di un database MySQL
// Mi connetto a MySQL
$host = 'localhost';
$db = ‘prova’;
$user = 'vostro_user';
$password = 'vostra_password';
$conn=mysql_connect($host,$user,$password) or die ("Non riesco a connettermi");
print "Connessione eseguita";
// Creo il database "prova"
mysql _create_db($db,$conn) or die ("Non riesco a creare il database");
// seleziono il database appena creato
mysql _select_db ($db,$conn) or die ("Non riesco a selezionare il database");
print "Connessione, creazione, selezione del database eseguita";
// Fine script
?>
```

Con questo script abbiamo completato la fase di connessione ad un database MySQL.
Abbiamo visto due nuove funzioni PHP.

```
mysql _create_db ("nome_database",”nome_connessione”);
```

e

```
mysql _select_db("nome_database", “nome_connessione”);
```

Notate che in ogni funzione abbiamo inserito l'istruzione die, questo ci aiuta in caso di problemi ad identificare subito il perché degli errori e la loro risoluzione.

TABELLE E DATI

Ora che ci siamo connessi al database da noi scelto possiamo eseguire qualsiasi azione al suo interno: creazione/cancellazione tabelle, creazione/modifica/cancellazione dati all'interno dei campi. Dobbiamo prima creare una tabella nella quale inserire i dati. A questo scopo utilizzeremo SQL. Una premessa è d'obbligo prima d’iniziare a creare la nostra tabella: è buona abitudine studiare il database e l'organizzazione interna delle nostre tabelle a tavolino prima di cominciare a costruire lo script. Un buon database con una buona organizzazione interna è senz'altro più facile da manipolare e leggere ed inoltre questo migliora la velocità di connessione e manipolazione dei dati. Per quanto riguarda i campi, nella loro creazione bisogna specificare il tipo di dato che il campo dovrà contenere, MySQL supporta una moltitudine di tipologie dei campi secondo i dati che si dovranno inserire, vediamo quali sono i tipi più importanti. Per quanto riguarda i numeri

reali MySQL supporta svariati tipi di archiviazione, la differenza fra i tipi elencati sotto non è nel formato del numero che andrà inserito nel campo ma la grandezza del numero (più grande è il numero è più è la memoria utilizzata).
TINYINT 1 byte
SMALLINT 2 byte
MEDIUMINT 3 byte
INT 4 byte
BIGINT 8 byte
Per quanto riguarda i numeri in virgola mobile abbiamo invece le seguenti tipologie di archiviazione.
FLOAT 4 byte
DOUBLE 8 byte
Per quanto riguarda le stringhe sono due i tipi di campo più utilizzati.
CHAR(numero_caratteri)
VARCHAR
Come possiamo intuire mentre in un campo *CHAR* la lunghezza è fissa ed è stabilita da noi durante la creazione della tabella nel campo *VARCHAR* il numero di caratteri è variabile e non bisogna specificare preventivamente il numero di caratteri massimo che dovrà contenere il campo. Naturalmente i campi di tipo *VARCHAR* occuperanno più memoria e la lettura al loro interno impiegherà qualche decimo di secondo in più rispetto ai campi *CHAR*. Un altro motivo dunque per cui dobbiamo organizzare anticipatamente il database è anche per la scelta del tipo di campo sapendo che questa influisce sulle prestazioni delle tabelle. Per capirci: è uno spreco di spazio e di prestazioni dichiarare un tipo di campo *VARCHAR* se sapete già che quel campo non potrà contenere un numero determinato di caratteri.
Supponiamo di voler costruire una tabella che raccoglie una serie d'indirizzi e-mail per una eventuale mailing-list del nostro sito. Dovremo costruire una tabella con tre colonne: una colonna raccoglierà un numero identificativo che ci aiuti ad indicizzare i dati, una colonna per il nome e cognome degli utenti ed infine la colonna che conterrà le e-mail: il codice sarà questo.

```
<?php
// script per la creazione di una tabella per la catalogazione delle e-mail per una mailing
//list per non appesantire il codice supponiamo di esserci già connessi al database con le
//funzioni viste in precedenza scrivo l'istruzione SQL per la creazione della tabella
$sql= "CREATE TABLE mail (
id_utente INT(10) NOT NULL,
nome_cognome CHAR(100) NOT NULL,
mail CHAR(50) NOT NULL)";
// Invio l'istruzione SQL a MySQL
$res = mysql_query("$sql",$conn) or die ("Non riesco a creare la tabella",mysql_error());
print " La tabella mail è stata creata con successo!";
// fine dello script
?>
```

In questo modo abbiamo creato la nostra tabella con tre colonne. L'istruzione SQL per comodità l'abbiamo inserito all'interno di una variabile e vorrei anche farvi notare la funzione per l'invio delle istruzioni SQL al database: **MySQL_query**. La sua sintassi è molto semplice.
MySQL_query("istruzioni SQL","nome_connessione");
Il risultato dipenderà da ciò che si chiede di fare al MySQL.

MANIPOLAZIONE DATI: LETTURA, INSERIMENTO

Ora che abbiamo creato la nostra tabella possiamo iniziare ad inserire i dati. Vediamo ora i comandi SQL e le funzioni PHP per inserire i dati.

```
<?php
```

// script per inserire i dati nella tabella mail per comodità supponiamo di esserci già
//connessi al database voglio inserire un nuovo indirizzo, scriveremo:
mysql_query("insert into mail (id_utente, nome_cognome,mail) values ('1','Mario Rossi','mario@suosito.com')");
// fine script
?>

Con questo piccolo script abbiamo inserito un nuovo indirizzo all'interno della tabella mail. Non bisogna specificare a MySQL di creare un nuovo campo in quanto questo avviene in automatico con l'uso dell'istruzione SQL "insert into".
La sintassi dell'istruzione INSERT INTO è una delle istruzioni più utilizzate e la più complessa perché molto articolata:
INSERT INTO nome_tabella (nome_campi) values (dati_da_inserire_nei_campi);
Notate che l'elenco dei dati presente nella seconda coppia di parentesi tonde deve corrispondere all'ordine dei campi che abbiamo inserito nella coppia delle prime parentesi. Ripeteremo lo script tutte le volte che dobbiamo inserire un nuovo campo. Ora che la nostra tabella contiene qualche dato possiamo leggere al suo interno per sapere quali dati ci sono al suo interno, per far questo utilizzeremo l'istruzione SQL "SELECT".

<?php
// script per leggere i dati contenuti in un campo della tabella mail
// per comodità supponiamo di esserci già connessi al database
*$dati = mysql_query("select * from mail");*
$array = mysql _fetch_array($dati);
// fine script
?>

L'istruzione SELECT chiede i dati di una riga della tabella che abbiamo selezionato nell'istruzione FROM (nel nostro caso la tabella si chiama mail). Per poter utilizzare i dati che MySQL invia dobbiamo utilizzare la funzione **MySQL_fetch_array** che crea un array associativo che ha come indice il nome delle colonne, continuando lo script avremo che per visualizzare i dati che abbiamo estrapolato dal database scriveremo.

<?php
// codice per leggere i dati contenuti in un campo
print "Contenuto della colonna id_utente: $array[id_utente] ";
print "Contenuto della colonna nome_cognome: $array[nome_cognome] ";
print "Contenuto della colonna mail: $array[mail] ";
?>

Naturalmente questo script legge una riga per volta e se nella istruzione SELECT non selezionate nulla MySQL restituirà i dati dell'ultima riga inserita, diversamente potete controllare il flusso dei dati MySQL con l'istruzione WHERE.

<?php
// Volendo visualizzare i dati dell'utente Mario Rossi avrei scritto
// in questo modo l'istruzione SQL
*$dati = mysql_query("SELECT * FROM mail WHERE nome_cognome='Mario Rossi");*
$array = mysql _fetch_array($dati);
// fine script
?>

Nella nostra lingua l'istruzione SQL risulterebbe così organizzata:
"Seleziona (SELECT) tutto (*) dalla tabella (FROM) mail dove (WHERE) la colonna nome_cognome è uguale a Mario Rossi.
Questo script va bene nel caso dovessimo leggere solo una riga della tabella, nel caso in cui vogliamo invece leggere tutto il contenuto della tabella dovremo aggiungere un cido while così organizzato.

<?php
// script per leggere i dati contenuti in tutti i campi della tabella mail
// per comodità supponiamo di esserci già connessi al database

```
$dati = mysql _query("select * from mail");
while ( $array = mysql _fetch_array($dati)) {
print "Contenuto della colonna id_utente: $array[id_utente] ";
print "Contenuto della colonna nome_cognome: $array[nome_cognome] ";
print "Contenuto della colonna mail: $array[mail] ";
}
// fine script
?>
```

Con questo script finche la tabella mail non sarà vuota PHP creerà l'array associativo contenente i dati letti dal database.
Le ultime importanti nozioni sulla manipolazione dei dati inseriti nel database riguardano la modifica e la cancellazione dei dati. A questo scopo, rispetto agli script visti precedentemente riguardo l'inserimento cambiano solo le istruzioni SQL. Per la modifica utilizzeremo l'istruzione UPDATE mentre per la cancellazione l'istruzione DELETE.
La modifica.

```
<?php
// script per la modifica dei dati nella tabella mail
// supponiamo di essere già connessi al database
$dati = mysql _query ("UPDATE mail SET mail='mario@tiscalinet.it' WHERE
nome_cognome='MARIO ROSSI'");
// fine script
?>
```

L'istruzione UPDATE è molto semplice, ricordate sempre di specificare il WHERE perché altrimenti la modifica sarà eseguita in tutti campi della tabella mail.
Vediamo la sintassi dell'istruzione UPDATE.

```
UPDATE nome_tabella SET nome_colonna='nuovo_valore' WHERE
nome_colonna='identificativo_colonna';
```

Nella creazione della tabella abbiamo all'inizio previsto una colonna che contiene l'identificativo numerico del campo. Questo indice è importantissimo per le istruzioni di modifica e di cancellazione perché in questo modo ogni riga ha un numero univoco e non si rischia di cancellare/modificare altre righe. Utilizzando l'identificativo avremo scritto.

```
<?php
// script per la modifica dei dati nella tabella mail
// supponiamo di essere già connessi al database
$dati = mysql _query ("UPDATE mail SET mail='mario@tiscalinet.it' WHERE
id_utente='1'");
// fine script
?>
```

Molto più semplice e senza rischio d'errore (se ci fossero stati due Mario Rossi all'interno della tabella mail).
L'istruzione DELETE permette di cancellare un intera riga dalla tabella.

```
<?php
// script per la cancellazione di tutti i dati nella tabella mail
// supponiamo di essere già connessi al database
$dati = mysql _query ("DELETE FROM mail");
// fine script
?>
```

Con questo script cancelliamo tutte le righe presenti all'interno della tabella mail. Nel caso volessimo cancellare una determinata riga inseriremo nell'istruzione SQL l'istruzione WHERE, come segue.

```
<?php
// script per la cancellazione di una riga nella tabella mail
// supponiamo di essere già connessi al database
$dati = mysql _query ("DELETE FROM mail where id_utente='1'");
```

```
// fine script
?>
```
Questo script cancellerà la riga in cui l'id_utente è uguale ad uno.
PHP contiene molte funzioni, un semplice elenco di quelle più importanti.

MySQL_num_rows()
Restituisce il numero di righe interessate dall'istruzione SQL.
```
<?php
$dati = mysql _query("SELECT * FROM mail");
$numero_righe = mysql _num_rows($dati);
?>
```
MySQL_insert_id()
Restituisce l'ultimo id (se presente) della riga interessata dall'ultima operazione di INSERT.
```
<?php
$dati = mysql _query("INSERT INTO mail (id_utente, nome_cognome, mail values ('2',
'Mario Rossi', mario@tiscalinet.it') ");
$ultimo_id = mysql _insert_id();
?>
```
MySQL_drop_db
Elimina un database MySQL.
```
<?php
mysql _drop_db("nome_database_da_eliminare");
?>
```
MySQL_list_dbs
Restituisce la lista dei database presenti nel server MySQL.
```
<?php
$connessione = mysql _connect($host, $user, $password);
mysql _list_dbs("$connessione");
?>
```
MySQL_list_tables
Restituisce la lista delle tabelle presenti nel database selezionato.
```
<?php
mysql _list_tables("nome_database");
?>
```

CHIUSURA DI UNA CONNESSIONE

Una volta conclusa la fase di manipolazione dei dati è sempre opportuno chiudere tutte le connessioni al server MySQL e magari liberare la memoria occupata dai risultati della query SQL. È buona norma specificare sempre queste due funzioni per non caricare inutilmente il server che sta eseguendo i vostri script e dare la possibilità a tutti gli utenti di poter accedere al database.
Come ultimo esempio uno script completo di connessione, manipolazione e disconnessione ad un server MySQL.
```
<?php
// Mi connetto a MySQL
$host = 'localhost';
$user = 'vostro_user';
$password = 'vostra_password';
$conn = mysql _connect($host,$user,$password) or die ("Non riesco a connettermi");
print "Connessione eseguita";
// seleziono il database appena creato
mysql _select_db("mail") or die ("Non riesco a selezionare il database");
print "Il database è stato selezionato";
// visualizzo tutti i campi presenti nella tabella mail
```

```
$dati = mysql _query("select * from mail");
// inizio il ciclo while per la visualizzazione delle righe
while ($array = mysql _fetch_array($dati) {
print "Contenuto colonna id_utente: $array[id_utente] <br>";
print "Contenuto colonna nome_cognome: $array[nome_cognome] <br>";
print "Contenuto colonna mail: $array[mail]";
}
// libero la memoria occupata dall'istruzione SELECT
mysql _free_result($dati);
// chiudo la connessione al server MySQL
mysql _close($conn);
// Fine script
?>
```

Da notare che la funzione MySQL_close può non contenere argomenti perché MySQL chiuderà automaticamente tutte le connessioni aperte.

UBERTINI MASSIMO
http://www.ubertini.it
massimo@ubertini.it

Dip. Informatica Industriale
I.T.I.S. "Giacomo Fauser"
Via Ricci, 14
28100 Novara Italy

tel. +39 0321482411
fax +39 0321482444

http://www.fauser.edu
http://www.fauser.edu/fau/sistem.htm
massimo@fauser.edu