Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut

Funzionamento del sistema GNU/Linux

Funzionamento di base

Moduli per la gestione hardware

Il sistema operativo svolge una molteplicità di ruoli, astraendo le risorse fisiche (dischi, porte, schede di rete) e logiche (filesystem, stack di rete) e controllandone l'accesso. In Linux queste funzioni sono realizzate mediante un framework che permette di attivarle e disattivarle modularmente, in particolare l'accesso all'hardware è astratto a moduli chiamati device driver.

Alcuni sono parte del kernel, ma la maggior parte viene caricata dinamicamente dalla directory /lib/modules. Più in dettaglio, a seguito della rilevazione di un dispositivo fisico il controller di I/O invia un interrupt alla CPU, che eseguendo un handler posta un evento su dbus (canale pub-sub per eventi di sistema). Successivamente, udev riceve l'evento e consulta /lib/modules/<kernel_vers>/modules.alias dove individua il device driver adatto al dispositivo.

Ogni modulo definisce in che modo vanno implementate le system call previste per i dispositivi:

- Dispositivi a blocchi: usano buffer e cache per ottimizzare il trasferimento di blocchi di una certa dimensione. Si comportano come file conservando un elenco di byte singolarmente indirizzabili (es. dischi)
- Dispositivi a caratteri: trasferiscono dati un carattere alla volta, possono consumarli per farci qualcosa (es. mostrarli su un terminale) o fornirli se ne hanno disponibili, lasciando il consumatore in attesa se non ne hanno (es. tastiera)
- Dispositivi vari (misc)

Le operazioni definite nei device drivers sono attuate usando le system call tipiche dei file su alcuni file speciali detti Device Files. Questi file si trovano in /dev e hanno degli attributi particolari:

es. brw-rw---- 1 root disk 8,1 Mar 20 11:14 /dev/sda1

- Un bit speciale per identificare se sono a blocchi (b) o a caratteri (c)
- Un major number (es. 8) per identificare quale device driver usare
- Un minor number (es. 1) per indicare l'istanza di dispositivo di quella classe

Alcuni device files non rappresentano vere periferiche, essendo implementati dal kernel per utilità nel lavoro coi processi:

- /dev/zero produce uno stream infinito di zeri binari
- /dev/null restituisce EOF a ogni read e ogni write viene scartata
- /dev/random produce byte casuali ad alta entropia, bloccante ove non ce ne sia abbastanza
- /dev/urandom produce stream pseudocasuale infinito

Altri device files notevoli relativi a vere periferiche

- /dev/tty* terminali fisici del sistema
- /dev/pts/* pseudo-terminali (finestre grafiche)
- /dev/sd* dischi e partizioni

Il sistema di storage

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut

I dispositivi di storage sono tipicamente dispositivi a blocchi, essi possono essere utilizzati come un semplice elenco di blocchi numerati. Per sfruttarli in maniera più organizzata è necessario compiere tre operazioni:

- Partizionamento: suddivisione in sottoinsiemi di blocchi. Non è obbligatoria ma è sempre utilizzato perché permette di usare ogni partizione come un dispositivo indipendente, utile per separare spazi con esigenze diverse (filesystem diversi, dati di sistema e dati utente). Un esempio utile può essere una partizione di swap, che, seppur oggi il calo di prestazioni sia notevole per usarla realmente come memoria virtuale, viene spesso usata per caricare la RAM andando in ibernazione.

I due standard più diffusi sono

  - MBR (Master Boot Record) in cui si ha una tabella principale con massimo 4 partizioni primarie (delimitate da blocco di inizio e blocco di fine), e una di queste può essere segnata come partizione estesa contenente fino a 12 unità logiche (utilizzabili come fossero una partizione primaria)
  - GPT (GUID Partition Table Scheme) usato specialmente con UEFI, permette di avere un massimo di 128 partizioni da 8 ZB invece delle 15 da 2 TB offerte da MBR

  I device file per dischi e partizioni seguono il formato /dev/sdXXNN dove XX è una o più lettere che identifica il disco e NN è il numero di partizione. Per i dischi NVMe/M.2 il formato è /dev/nvmeXnYpZ dove X identifica il disco, Y il namespace e Z la partizione. Gli identificatori di disco non sono fissi ma variano in base all'ordine in cui essi vengono rilevati al boot.

- Formattazione: creazione del filesystem per organizzare lo spazio in modo comprensibile. La formattazione richiede spazio contiguo, potendo crescere e ridursi ma senza avere "buchi". Permette l'accesso ai file secondo il modello del VFS (Virtual File System) di Linux, che astrae dall'organizzazione dati specifica di vari FS.

Il FS più comune oggi in Linux è ext4, nato per estendere ext3 nei limiti di dimensione file, filesystem e numero di subdirectory e per introdurre il journaling nativo, offrendo prestazioni superiori rispetto a quello aggiunto da ext3 su ext2; aggiungendo anche la possibilità di usare un checksum per la consistenza del journal invece di un commit a due fasi. Il journal è disattivabile per evitare l'eccesso di scritture in pochi blocchi per i dischi SSD. Esso introduce il concetto di extent invece di allocazione indiretta, permettendo a file di grandi dimensioni di essere allocati in modo contiguo.

- Mount: innestamento della gerarchia locale creata dal partizionamento nella gerarchia globale. A seguito del mount, il nome assoluto di un file diventa la combinazione tra la posizione relativa nel device di appartenenza e il punto di mount di questo, ad esempio
  - 1) mount /dev/sda1 /
  - 2) mount /dev/sda5 /home

  Essendo /home nella root di sda1, montato su /,
  il file mary presente nella root di sda5 avrà nome assoluto /home (punto di mount di sda5) + /mary (posizione relativa nel device di appartenenza) = /home/mary

Collocazione delle risorse

La struttura del filesystem UNIX viene definita da FHS (Filesystem Hierarchy Standard), per rendere più facile l'individuazione delle risorse e la condivisione di parti del filesystem. Le distinzioni base per la corretta collocazione dei dati sono 2:

- Dati statici / variabili (i dati statici possono essere salvati su supporti read-only e non richiedono backup con la stessa frequenza di quelli variabili)

- Dati condivisibili / non condivisibili (I dati condivisibili possono stare in un solo host ed essere usati in molteplici, alcuni sono obbligatoriamente host-only)

La radice della gerarchia, /, deve contenere tutti i dati necessari all'avvio del sistema ma essere il più compatto possibile per ridurre i rischi di corruzione e poter risiedere in media a bassa capacità.

In un sistema Linux si possono osservare diversi filesystem virtuali, non corrispondenti a nessun dispositivo fisico. Notiamo proc e sys, che permettono accesso diretto ai dati del kernel come aree di memoria, parametri di processi e di configurazione moduli; udev, che permette la generazione automatica dei device files da parte dei device drivers, tmpfs per mappare in memoria anziché su disco dati volatili.

FUSE è un driver che permette di accedere a filesystem diversi da quelli definiti nel kernel mediante un processo user, è utile ad accedere a filesystem che non necessitano di vero hardware come quelli di rete.

Gestione dei pacchetti software

Ogni sistema ha bisogno di un'oculata gestione del software installato, dal momento della sua installazione fino a quello della disinstallazione: generalmente parlando è sempre opportuno disinstallare software che non ci serve più. La fase dell'aggiornamento dei pacchetti è particolarmente critica, in quanto nuove versioni potrebbero interrompere il supporto ad alcune componenti hardware o software del sistema, o causare problemi con altri pacchetti con cui c'è dipendenza. Generalmente, se un software funziona non è importante avere l'ultima versione, può però essere necessario applicare eventuali patch di sicurezza ad hoc, nel caso in cui non si possa passare ad una versione più recente che includa i fix necessari. La configurazione dei software è altrettanto importante in quanto deve tenere conto delle politiche di utilizzo ed eventualmente prevedere particolari casi d'uso o caratteristiche del sistema.

Installazione manuale

L'installazione manuale può passare da binari precompilati, che necessitano soltanto di essere copiati nelle posizioni di interesse, o da file sorgente, che necessitano di compilazione ma sono indipendenti dall'architettura e offrono più flessibilità nel soddisfacimento delle dipendenze. Questo perché i sorgente spesso offrono interfacce usate per adeguare l'installazione al sistema, tenendo conto ad esempio di componenti già presenti.

Nella maggior parte dei casi il software è scritto in C e viene distribuito con archivi tar.gz, dotandolo di un'autoconf che verifica la soddisfazione dei prerequisiti, rilevando la presenza dei pacchetti necessari e la loro versione, permette all'utente di specificare configurazioni particolari e infine genera i Makefile tenendo conto delle specificità del sistema e delle scelte utente.

Per un'installazione manuale da sorgenti, i passi tipici sono:

- reperimento software:
    - E' importante autenticare i pacchetti usati, solitamente mediante una chiave pubblica fidata (comando gpg) o come minimo verificando il fingerprint del file (es. comando sha256, si possono avere varie hash)
- estrazione pacchetto

    - I file vengono tipicamente prima compattati in un solo tar, e poi compressi. E' buona prassi individuare una posizione sensata per l'estrazione, e testare l'archivio prima dell'estrazione per verificare la gerarchia di directory che sarà generata

- esame delle scelte offerte
    - Nella directory data dall'estrazione è opportuno leggere eventuali README e INSTALL. Se si ha un eseguibile *configure*, si lancia con –help per avere una lista dei parametri di configurazione; ad esempio collocazione del software, attivazione di componenti, caricamento dinamico di moduli invece che integrazione statica. Opzioni Without oppure with per scegliere componenti previsti di default/per escluderli
- configurazione sorgenti
    - Si lancia *configure* con i parametri scelti, risolvendo eventuali problemi evidenziati (di solito mancanza di pacchetti necessari, può essere necessaria un'indagine manuale). Viene generato un file config.log
- compilazione
    - Si lancia *make* o si seguono le indicazioni del config.log
- installazione, usando *sudo make install*. È importante che solo questo passo venga eseguito come root essendoci stati in passato malware che sfruttano privilegi eccessivi nei passi precedenti.

L'installazione di sorgenti offre la possibilità di verificare il codice, tuttavia è più difficile da mantenere richiedendo una verifica manuale di dipendenze e configurazioni. Richiede inoltre molti componenti ausiliari (header, librerie, compilatori) che possono facilitare un attaccante.

Le distribuzioni e i pacchetti puntano ad alleviare questi contro, fornendo una gestione automatica delle dipendenze e garantendo compatibilità tra tutti gli elementi del set di pacchetti.

Installazione assistita

L'installazione assistita si effettua tipicamente usando un package manager, nelle distribuzioni Linux. Il tool si fa carico di verificare le dipendenze, scorrendo il grafo delle dipendenze di un dato pacchetto individuando tutti i componenti di cui ha bisogno.

Un pacchetto tipicamente si presenta come un singolo file che contiene sia il software precompilato che i criteri per la verifica di compatibilità e prerequisiti. Chiaramente la garanzia di compatibilità può esserci solo vincolando l'architettura di esecuzione, la versione della distribuzione e la versione del software.

Distribuzioni

Nella scelta di una distribuzione per un particolare applicazione bisognare tenere conto di diversi fattori:

- Architetture supportate: tutte le distribuzioni supportano Intel 32bit, alcune 64bit, altre sono disponibili per la grande varietà di processori su cui è stato portato il kernel, ma è bene ricordare che il supporto a pacchetti di terze parti non è garantito per architetture esotiche
- Stabilità vs aggiornamento: in GNU/Linux il processo di rilascio frequente di software porta a dover confrontare i benefici dell'avere versioni più recenti del software, con più funzionalità, con la stabilità di versioni già ampiamente testate; e la stabilità è importante su un server in produzione

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut

- Version vs rolling: Le distribuzioni con versione forniscono solo aggiornamenti correttivi durante tutto il loro ciclo di vita. Eventuali novità vengono testate e accumulate per una nuova versione, che dovrà essere installata sovrascrivendo la precedente. Per contro in una distribuzione rolling ogni novità viene testata e distribuita, quindi si ha sempre a disposizione la versione più recente. Le distribuzioni rolling per loro natura non sono molto adatte all'uso su un server

- Supporto e durata: Il supporto garantito è tipico delle distribuzioni commerciali, per quelle gratuite dipende dalla dimensione della comunità di utenti. Per installazioni server esistono distribuzioni LTS (Long Term Support) per le quali i gestori assicurano per 5/7 anni che le API restino le stesse, garantendo il backporting dei security fix

Le due distribuzioni capostipite da cui derivano le varianti più diffuse sono Debian e Red Hat. Offrono sistemi di gestione pacchetti che si assomigliano, con tool di basso livello per la gestione dei singoli pacchetti, tool intermedi per la gestione coordinata delle dipendenze e tool per reperire automaticamente i pacchetti necessari dai repository (depositi).

Pacchetti

Il nome dei pacchetti segue tipicamente il formato nome-versioneSoftware-versionePacchetto-architettura.estensione, quest'ultima per Debian e derivate è .deb, per Red Hat e derivate in .rpm. La versione del pacchetto permette di patchare eventuali bug senza installare una versione del software superiore, che potrebbe si includere nuove funzionalità ma anche interrompere il supporto ad altre utilizzate dalle dipendenze.

Si distingue tra pacchetti base (dotati solo del necessario per eseguire funzioni di libreria, ovvero codice oggetto in formato adatto al linking dinamico, per Linux .so Shared Object) e pacchetti development (-dev per Debian e -devel per RedHat), che dispongo anche di codice oggetto in formato adatto al linking statico e prototipi per il compilatore.

Verifica dipendenze dinamiche: *ldd percorsopacchetto*

I pacchetti possono essere scaricati e gestiti singolarmente, ma di solito si utilizzano dei repository, ovvero raccolte indicizzate di pacchetti, che possono essere sia online che locali. I package manager leggono per ogni repo l'indice e i metadati dei pacchetti e riconoscono quali versioni sono disponibili, oltre a conoscere le dipendenze come risolverle. Le liste di repo si trovano in /etc/apt/sources.list in deb.

Per i pacchetti .deb il package manager è tipicamente *apt* ad alto livello e *dpkg* a basso livello, per quelli .rpm corrispondono *yum* e *rpm*. Si aggiungono poi i sottocomandi *update* per aggiornare le liste pacchetti e *upgrade* per passare alla versione più recente, *install* e *remove* per ovviamente installare e rimuovere.

La firma dei pacchetti viene gestita centralmente, i mantainer di una distribuzione forniscono le chiavi di verifica nei media di installazione o sui repo online.

Per utilizzare pacchetti ben supportati ma non inclusi nei canali ufficiali della distribuzione è sufficiente aggiungere il repo all'elenco. Bisogna però fare attenzione, perché avere un pacchetto con lo stesso nome con versione diverse in più repo può causare confusione, in quanto i package manager scelgono sempre la versione più avanzata a default. Si pensi a un repo semisconosciuto in cui viene caricato un pacchetto "core" con versione più avanzata di quella ufficiale, realizzando una software injection.

E' sconsigliabile mischiare installazioni manuali con pacchetti. Quando possibile è utile pacchettizzare le proprie applicazioni, seguendo il processo per costruire un proprio repo, in RPM.

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut
Per aggiornare un un pacchetto già in uso bisogna tenere conto di problemi derivanti da

- prerequisiti: pacchetti necessari perché il candidato funzioni, come per l'installazione
- configurazione: si pensi a modifiche al formato delle configurazioni già messe a punto per la versione funzionante
- dipendenze di altri software: modifiche alle interfacce mostrate potrebbero influire sul funzionamento di altri software. Se il software interagisce mediante interfacce standard, potrebbero esserci problemi di trasparenza qualora si volesse usare una configurazione di test per provare una nuova versione (legando le due versioni a socket/porte/IP diversi

La disinstallazione di un pacchetto pone problemi simili all'aggiornamento in termini di eventuali dipendenze di altri software da quello che si vuole rimuovere, in entrambi i casi può essere difficile valutare gli effetti sul sistema con una gestione manuale.

Il grafo delle dipendenze rappresenta il valore aggiunto più importante offerto dai package manager.

Crittografia e SSH

La sicurezza delle informazioni necessita di

- **riservatezza:** evitare che entità non autorizzate vengano messe a conoscenza di informazioni
- **integrità (**comprende **autenticità):** riconoscere con certezza se un dato è come era stato prodotto. L'autenticità consiste nella corretta attribuzione dell'autore ad un'informazione, per evitare che appaia prodotta da qualcun altro
- **disponibilità:** non può essere difesa crittograficamente, se l'informazione diventa inaccessibile lo resta, a prescindere da come fosse stata cifrata.

Attacchi a riservatezza e integrità possono essere attuati da intrusi posti tra mittente e destinatario; la soluzione è la crittografia; ovvero un'elaborazione matematica e algoritmica della codifica delle informazioni. Violazioni alla riservatezza sono prevenute alterando il codice in modo che risulti incomprensibile a chi non ha diritto di conoscere l'informazione (l'operazione di cifratura deve essere fatta per intervenire in maniera preventiva e far si che l'attacco non abbia successo, un approccio a posteriori non ha senso nel caso di riservatezza perché ciò che doveva restare segreto non lo è più), violazioni all'integrità possono essere solo rilevate a posteriori (impossibile prevenirle) mediante strumenti in grado di dire se un dato è integro o meno; per prendere una decisione razionale in cui si rifiuta l'informazione finché non passa i controlli.

I cifrari per la sicurezza prevedono due azioni, cifratura (per passare da testo in chiaro a testo cifrato, Encrypt) e decifrazione (viceversa).

La funzione D è critica: potrebbe esistere uno schema in cui chiunque può svolgere E (se è una funzione pubblica), la cosa importante è che la funzione D sia segreta (conosciuta solo al legittimo destinatario). E può essere pubblica ma deve essere robusta (ovvero trasforma testo in chiaro in cifrato in modo che sia obbligatorio avere D per decifrarlo), D invece deve essere segreta necessariamente.

Alla base dei sistemi crittografici sono i principi di Kerckhoffs:

- 1) Il sistema di cifratura deve essere materialmente, se non matematicamente, indecifrabile (sicurezza computazionale o assoluta)

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut

- 2) Il sistema non deve basarsi sul fatto che il segreto, se cade nelle mani nemiche, causi un grave problema. Il segreto deve diventare solo un parametro della funzione: non è più la funzione ad essere il segreto. segreto=chiave; NON segreto=algoritmo

- 3) La chiave deve essere qualcosa di semplice, possibilmente memorizzabile senza supporti. Dover ricordare un segreto semplice permette lo scambio di molti segreti di dimensioni arbitrarie

Un crittoanalista di fronte ad un testo cifrato con algoritmo noto può analizzare le proprietà statistiche del testo per rilevare (si evidenzia l'importanza della robustezza di E, ovvero la sua capacità di occultare le proprietà del testo in chiaro) oppure cercare la chiave tra tutte quelle possibili (entra in gioco il primo principio: con la sicurezza assoluta, la chiave corretta è indistinguibile dalle altre – es. con analisi si ottengono 800K testi insensati e 300K sensati: anche di fronte ad attacco a forza bruta il crittoanalista non può essere certo del messaggio corretto- ; con la sicurezza computazionale si rende troppo oneroso cercare la chiave – es. con analisi si trovano 999K testi incomprensibili e 1 comprensibile; analista ha certezza di aver trovato la chiave, ma è oneroso scoprirla).

Alla base della robustezza ci sono le proprietà di confusione (l'analisi del testo cifrato non restituisce informazioni sulla chiave, modifica ad un singolo elemento della chiave si riflette sul 50% del cifrato) e di diffusione (l'analisi del testo cifrato non restituisce informazioni sul testo in chiaro, modifica di un singolo elemento del testo in chiaro altera il 50% del cifrato).

Un esempio di cifratura con scarsa robustezza è la sostituzione monoalfabetica (ogni elemento dell'alfabeto viene sostituito con un altro, con tabella di corrispondenza). La chiave ha uno spazio di 26! (Avendo 26 lettere con cui sostituire la A, una volta sostituita la A, mi restano 25 lettere con cui sostituire la B…) quindi 2^88 chiavi: non è male, si ha resistenza alla forza bruta ma manca la robustezza: le proprietà statistiche del testo in chiaro sono mantenute nel testo cifrato (per ogni lettere si ha una lettera). C'è scarsa diffusione: gli attacchi alla sostituzione si basano sulla frequenza statistica dei caratteri, in una data lingua naturale (es. in inglese, il 12% dei caratteri è 'e'); ma in binario ogni "lettera" può essere un lungo blocco di bit: se le lettere non sono più 26 ma 256, il valor medio di ogni colonna è molto più basso. Inoltre la compressione che si può attuare sul binario cifrato fa sì che la ridondanza venga rimossa e quindi la differenza statistica tra le lettere si riduce ancora di più.

La diffusione viene introdotta in modo semplice dalla trasposizione, che si basa su una tabella in cui il testo viene scritto per colonne e letto per righe. La ricerca della chiave prevede quindi di scoprire la dimensione della tabella e l'ordine di lettura delle righe, la minaccia alla robustezza sta nel fatto che digrammi e trigrammi potrebbero presentarsi a distanze simili, permettendo di scoprire di quanto sono stati trasposti i caratteri. Applicando la trasposizione più volte rende difficile questa analisi perché i passi successivi partono da un testo i cui digrammi non sono quelli della lingua originale.

La sostituzione polialfabetica prevede la somma della chiave al testo, modulo 26 (A=0, … Z=25). Il risultato è che la frequenza di un carattere in chiaro sia sparsa su più caratteri cifrati (stesso char chiaro diventa diversi char cifrati a seconda di quale parte della chiave viene sommata); e la frequenza di un carattere cifrato deriva da contributi di più caratteri in chiaro (stesso char cifrato può essere risultato di diversi char chiaro in base alla parte di chiave sommata). L'analisi statistica diventa difficile, ma è attaccabile osservando il ripetersi delle sostituzioni (stessi char chiari diventano stesso char cifrato ogni X sostituzioni) oppure conoscendo parte del messaggio (se si può sapere che tutti i messaggi iniziano con "domani" o che i bollettini meteo inizino con l'indicazione della stazione da cui provengono).

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut

Dall'idea di base che la polialfabetica sia debole perché la chiave si ripete, nasce il one-time pad: anziché generare la chiave in modo deterministico, ogni simbolo è scelto casualmente per avere una chiave con la stessa lunghezza del messaggio. Questo fornisce sicurezza assoluta (non cambia l'incertezza sul contenuto del messaggio se l'attaccante fa analisi statistica prima o dopo il tentativo a forza bruta: in entrambi i casi lo spazio è quello di tutte le parole della lingua considerata). È difficile da implementare perché c'è rischio di perdere la sincronia tra mittente e ricevente su quale sia il carattere usato per cifrarne uno, e c'è il costo di tenere in memoria tutte le chiavi possibili.

I cifrari simmetrici moderni operano basandosi sugli stessi principi di confusione, ripetendo sostituzioni e trasposizioni. Sono studiati per essere computazionalmente sicuri, questa sicurezza risiede nella lunghezza della chiave: lo standard attuale, AES, usa chiavi di oltre 64 bit, già da 128 sono introvabili in tempi umani con le tecnologie attuali.

Gli stessi principi si possono usare senza chiave per ottenere un'"impronta digitale" compatta di documenti di dimensione arbitraria. I fingerprint sono il risultato di dimensione fissa di funzioni pubbliche senza chiave (funzioni di *hash*), avendo dimensione fissa non permettono una corrispondenza biunivoca tra fingerprint e documento. Le funzioni hash sono robuste se non si riesce a trovare un documento fornendo il fingerprint (unidirezionalità) e non si possono trovare due documenti con lo stesso fingerprint (assenza di collisioni). La difficoltà di invertire la funzione è fondamentale, si realizza mediante molte operazioni in aritmetica modulare (come risultato di un'operazione si prende il resto della divisione per un modulo fisso). Queste operazioni rendono difficile la ricerca efficiente delle radici.

La crittografia asimmetrica si basa sulla generazione di chiavi partendo da due numeri primi p e q : viene calcolato il modulo $n = p \cdot q$ (p e q devono essere abbastanza grandi che da n non si possa risalire ad essi in modo efficiente), dopo aver scelto a caso un numero d si calcola e tale che $e \cdot d \mod (p-1)(q-1) = 1$. Calcolare e è facile soltanto conoscendo p e q, che dopo la generazione delle chiavi vengono dimenticati: la chiave pubblica sarà (e,n), quella privata (d,n). La cifratura di un valore seguirà c = m^e mod n ; la decifrazione m = c^d mod n (l'elevamento a potenza è un'operazione facile; l'estrazione della radice è difficile: esiste ma è grande da calcolare).

La chiave pubblica può essere distribuita (essendo impossibile usarla per trovare la corrispondente privata) e chiunque può usarla per cifrare, mentre solo la chiave privata corrispondente può decifrare (garantendo la riservatezza). La chiave privata, essendo specifica di un utente, può essere utile anche per autenticare (guardando a integrità e autenticità): si prende il documento, si passa attraverso l'hash, poi l'hash viene cifrato con chiave privata. Si ottiene così un piccolo dato chiamato firma, allegato al documento originale: il destinatario fa hash del documento e verifica che il fingerprint sia uguale alla decifrazione della firma usando la chiave pubblica corrispondente alla chiave privata che ha prodotto la firma. L'integrità viene garantita dal non cambiare dell'hash, l'autenticità dal fatto che il confronto ha successo solo se la firma è stata prodotta dalla privata corrispondente (si assume che la pubblica usata per decifrare la firma sia effettivamente del mittente...).

I vantaggi della crittografia asimmetrica risiedono nella possibilità di distribuzione delle chiavi e nell'utilità per tutte le proprietà di sicurezza (riservatezza, integrità, autenticità); i punti deboli sono

- le prestazioni (5-10 volte più lento di AES, si ovvia usando sistemi ibridi: si genera una chiave casuale e si usa con AES (molto efficiente) per cifrare il messaggio grande. Questa chiave si cifra con la chiave pubblica del destinatario, che userà la propria chiave privata per decifrarla e poi la chiave simmetrica ottenuta per riavere il messaggio originale)
- attacchi specifici (known plaintext: es. referendum dove il messaggio è sempre SI o NO, se cifrato con chiave pubblica, anche l'attaccante può verificare qual è il risultato. Si usano

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut

> algoritmi per garantire che plaintext troppo corti o semplici vengano trasformati in numeri di complessità maggiore).

E' possibile usare la crittografia asimmetrica per l'autenticazione attiva. Nell'auth passiva (es. password) il segreto viene svelato a chi verifica ed è sempre uguale (intercettare una password permette un replay attack, riutilizzandola). Con auth attiva un prover dimostra a un verifier di avere il possesso di una chiave privata (ovvero la capacità di decifrare un grande numero R random scelto dal verifier e cifrato con la chiave pubblica del prover). Si superano entrambi i problemi dell'auth passiva: il segreto non viene svelato (il prover invia soltanto il risultato della decifrazione e non la chiave privata), e R non viene mai riutilizzato quindi intercettarlo e rigiocarlo è inutile. Tuttavia si ha lo stesso problema della verifica di autenticità di una firma, in quanto non c'è garanzia che il verifier detenga la chiave pubblica effettivamente del prover e non di un impostore.

Il metodo più efficiente di attacco alla crittografia asimmetrica è la fattorizzazione del modulo, per questo la lunghezza del modulo consigliata è di 2048 bit e oltre.

~~Vagrant e SSH~~

SSH

Per poter avere amministrazione remota è stato sviluppato TELNET, che però non offriva nessuna confidenzialità del canale né autenticazione dell'host, vi era soltanto autenticazione passiva dell'utente. Secure Shell (SSH) è l'evoluzione nata per offrire maggiore sicurezza, basandosi sull'autenticazione a chiave pubblica. Questa si basa si basa sulla crittografia asimmetrica, il segreto è una chiave privata posseduta soltanto da chi vuole autenticarsi, che riceve una challenge di autenticazione dal verifier e usa la chiave privata per fare response; il verificatore possiede una chiave pubblica legata matematicamente a quella privata (ma che non permette di calcolarla) con cui può controllare se la risposta è corretta. Il collegamento SSH tra client (ssh) e server (sshd) percorre alcuni passi:

- Negoziazione dei cifrari disponibili
- Autenticazione host remoto per mezzo della sua chiave pubblica
- Inizializzazione canale comunicazione cifrata
- Negoziazione dei metodi disponibili per l'autenticazione utente
- Autenticazione utente

Autenticare l'host remoto è importante per evitare attacchi MITM che potrebbero catturare la password dell'amministratore spacciandosi per l'host, ma non è previsto un sistema centrale per la verifica della chiave dell'host; quindi alla prima connessione è l'amministratore a dover usare un metodo out-of-band (verifica esterna ad ssh) per verificare la chiave pubblica presentata dall'host. Le chiavi pubbliche vengono salvate in *known_hosts* nella directory .ssh nella home client, permettendo autenticazione attiva alle connessioni successive.

Per autenticare l'utente si può avere autenticazione passiva (tradizionale con username e password trasmessi su canale cifrato) o attiva (usando un protocollo challenge-response a chiave pubblica, supponendo che l'utente abbia una coppia di chiavi e che installi correttamente sull'host remoto la chiave pubblica. In entrambi i casi l'identità dell'utente che vuole fare login sull'host remoto può essere selezionata, a default viene usata lo stesso nome utente che sta lavorando sul client (es. user esegue "ssh rmserver" e si dovra autenticare come user, user esegue "ssh marco@rmserver" e dovrà autenticarsi come marco).

Per fare l'autenticazione attiva un utente deve

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut

- generare una coppia di chiavi asimmetriche (comando *ssh-keygen -t rsa -b 2048)*
- installare sull'host remoto la chiave pubblica (file locale .ssh/is_rsa.pub. copia su host remoto (*scp .ssh/id_rsa.pub user@remote:))*
- append alla lista di utenti autorizzati (su remote, *cat id_rsa.pub >> .ssh_authorized_keys*)

In questo caso la segretezza della password viene sostituita dalla riservatezza del file della chiave privata sul client, in pratica è necessario avere grande cura dei permessi di file e directory (es. il passwordless login può non funzionare se l'host remoto ha permessi troppo larghi sulla dir .ssh, il server sshd non si fida dell'integrità del contenuto). Se proteggiamo il file chiave privata con una password perdiamo la possibilità di passwordless login ma abbiamo più sicurezza del digitare direttamente la password dell'account remoto, ed è più pratico se si gestiscono molti host remoti.

Con *ssh utente@host* si ottiene un terminale remoto interattivo, aggiungendo un ulteriore parametro viene interpretato come comando da eseguire sull'host remoto al posto della shell interattiva (es. ssh root@server "grep pattern") , gli stream di I/O vengono riportati attraverso il canale cifrato sul client (STDIN del processo ssh sul client passa dati a STDIN del processo invocato sul server, STDOUT e STDERR del processo remoto fuoriescono dagli analoghi stream dal processo ssh sul client).

~~Vagrant~~

~~Vagrant è una libreria per la gestione di macchine virtuali che permette di rendere portabile la configurazione rispetto a diversi virtualizzatori. L'immagine della VM di base (box) può essere depositata e facilmente importata da un deposito (VagrantCloud), i parametri per istanziare una VM sono definiti in un file (VagrantFile). Un V.File contiene informazioni sulla box di base, sui vari tipi di interfacce di rete, la mappatura di cartelle condivise host-guest, il provider del virtualizzatore e la configurazione dell' HW virtuale, oltre che la configurazione d'ambiente, il provisioning: una serie di comandi eseguiti da vagrant nella shell del guest al primo avvio (se fallisce in modo plateale è vagrant a ripeterlo, altrimenti tocca all'amministratore).~~

~~Per installare una box si può ricorrere al repo ufficiale VagrantCloud, scegliendo una box tra queste l'installazione è automatica al primo avvio ma si può scaricare localmente per evitare l'attesa del download all'avvio VM o per scegliere fonte diversa (*vagrant box add autore/version [url]*). La box è un template che viene salvato in una directory; per crearne diverse istanze VM tipicamente si crea una directory per ogni istanza, ci si pone lì e si lancia *vagrant init autore/version*, così il V.File generato avrà il riferimento a quella VM in un luogo separato. Con la init viene creato in quella dir un V.File con i default della box, modificabile; i comandi *vagrant {up|status|halt|destroy}* cercano un V.File nella directory per lanciarla/vederne lo stato/fermarla/distruggerla.~~

~~Vagrant + SSH~~

~~*vagrant up* crea automaticamente una chiave privata sull'host, nella directory in cui viene fatta init, e installa la chiave pubblica nella VM, predisponendo una mappatura di rete da una porta host alla 22 guest. v*agrant ssh* usa in automatico queste impostazioni per connettersi alla VM come user vagrant, senza password, abilitato a sudo. v*agrant ssh-config* mostra come configurare il client *ssh* per comportarsi come *vagrant ssh*, in modo da poter usare *ssh* originale per operazioni non supportate da *vagrant ssh*, come *scp.*~~

~~*vagrant ssh-config > ssh.conf ; ssh -F ssh.conf default* Connessione~~

~~Entrambi da host *; scp -F ssh.conf default:/path pathlocale* copia da guest~~

~~*; scp -F ssh.conf fileLocale default:/pathremoto* copia verso guest~~

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut
~~Vagrant permette di modificare hostname della VM con direttiva *config.vm.hostname = "prova"* , inoltre permette di utilizzare linked-clone mediante direttiva~~

~~*config.vm.provider "virtualbox" do |vb|*~~

~~*# Display the VirtualBox GUI when booting the machine*~~

~~*vb.linked_clone = true*~~

~~*end*~~

Gestione servizi (demoni)

/sbin/init è il processo che fa partire tutte le attività di servizio in background (di sistema), detti demoni in Unix. Non ci si accorge della loro presenza essendo disconnessi da qualsiasi terminale e eseguiti a nome di utenti specifici senza accesso a shell interattiva, gestiscono in maniera automatica il sistema.

Per gestire correttamente i processi è utile sapere che origine hanno, come terminarli evitando che ricompaiano: processi inutili consumano risorse e offrono opportunità di attacco. **Ps, top** e **kill** sono efficaci per individuare e risolvere problemi istantanei ma non per evitare che riappaiano. Ci sono tre fonti primarie di processi**:** pianificatori periodici e sporadici, demoni di gestione eventi, procedure di avvio del sistema.

La più semplice delle 3 fonti è l'esecuzione pianificata, che può essere periodica o sporadica.

crond è il demone che si occupa dell'esecuzione periodica.

- Ogni utente ha la propria *crontab* (cron table), non importa sapere dove si trovano perché ognuno può configurarle usando ***crontab -e*** che apre editor. Opzione ***-l*** elenca le task. Se root lancia ***crontab -e*** edita il suo cron table, con -u edita quello di altri utenti.
- I task di sistema si trovano in /etc/crontab, questo editabile solo da root e solo direttamente; a differenza di crontab contiene un campo in più che indica a nome di quale utente dovrà essere eseguito un certo comando, perché se root vuole pianificare task per altro utente non va a modificare il crontab di quell'utente (che potrebbe anche cambiarlo) ma modifica /etc/crontab. Contiene già forme "cron.frequenza" standard agganciate a tutto ciò che trova nelle directory /etc/cron.hourly daily weekly monthly, queste forme standard rendono più facile settare task periodici senza far danni ad altri task del crontab generale.
- Ogni crontab ha un elenco di direttive nella forma

MINUTO ORA G.MESE MESE G.SETTIMANA <comando>

L'azione è eseguita quando l'ora corrente corrisponde a tutti i selettori di una riga (campi in AND logico), tranne se c'è sia giorno mese che giorno settimana (in quel caso OR)

atd è il demone che si occupa di gestire code di compiti da svolgere in momenti prefissati, offre scheduling differito di un servizio, con possibilità di disinnesco.

L'interfaccia ad atd si basa sui comandi

- ***at [-V] [-q queue] [-f file] [-mldbv] TIME*** prende come parametro quando fare un job, quando si fa invio poi mostra altro prompt, perché si aspetta l'inserimento dei comandi da eseguire POI sullo stdin. **at** non è silente, restituisce il numero del job. Questo si può usare con **atrm** per rimuoverlo.

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut

- ***atq [-V] [-q queue] [-v]*** elenca i comandi in coda, formato

9 Sat Apr 2 16:04:00 2022 a vagrant

- ***atrm [-V] job [job…]*** rimuove comandi dalla coda.
- AT E CRON SONO DEMONI, gli stream vanno obbligatoriamente ridiretti, se non ci interessano si ridirigono a /dev/null. Se non li ridirigiamo non sappiamo che fine fanno, IMPORTANTE usare path assoluti dentro file crontab o dentro comando at perché non è detto che abbiano lo stesso environment della shell interattiva.
  - ***PATHCOMANDO="$(readlink -f "${BASH_SOURCE}")"*** salva nella variabile il percorso assoluto dello script in cui è contenuto

Esempio esercizio:

ATJOB=$(mktemp) preparo file temp per salvare il job id

echo "/bin/date > /tmp/prova2.las" | at now + 30 minutes 2>&1 | grep '^job ' | cut -f2 -d' ' >"$ATJOB" linea perfettamente non interattiva adatta per script:

con echo diamo input ad ***at*** per sapere cosa fare, con grep prendiamo la linea che ci interessa, con cut prendiamo il numero che ci interessa, con redirezione finale scriviamo sul file.

poi con cat $ATJOB possiamo recuperare il jobid (e possiamo usare ***atrm $(cat $ATJOB)*** per de-schedulare prima dell'esecuzione)

Esempio uso watchdog: esempio applicato sta nel book Gestione di servizi e monitoraggio

Watchdog: Utile se si vuole porre un limite al tempo di esecuzione di un task, senza ricorrere a una busy wait

start_long_task & PID=$!

WD=$(mktemp)

echo "/bin/kill $PID" | at now + $LIMITE minutes 2>&1 | grep '^job ' | cut -f2 -d' ' > $WD

Opportuno fare un array dove i PID sono indici e i valori sono i nomi dei comandi, così quando è il momento di lanciare kill, si può testare se al pid è ancora associato quel particolare comando. (Vedi parallenne.sh)

Init è il primo processo avviato dal kernel, si occupa di gestire le sequenze di eventi per raggiungere un dato **runlevel**: questi sono degli stati definiti del sistema, in cui sono stati sicuramente avviati alcuni processi e sicuramente fermati altri. La variante di interesse per noi è Systemd. (dettagli su sysvinit e upstart, nel pdf 3_gestione_servizi, c'è anche cheat-sheet per comandi)

Systemd

Systemd è il demone che si occupa di

- gestire le dipendenze tra i servizi: ha senso avviare alcuni servizi soltanto se sono già avviati i servizi su cui si basano (es. non ha senso chiedere login a utenti se non si è riusciti a montare la loro home) così come quando si installa software ha senso installare un pacchettoB solo se esiste già pacchettoA da cui dipende

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut

- avviare servizi a richiesta: se in una sequenza di passi c'è dipendenza ad es. del passo 5 dal 4, ma non dei passi dal 6 al 18; ho la possibilità di lanciare questi in parallelo invece di attendere la fine del 5

- logging precoce: il sistema di log è un demone, ma anche se lo lancio presto, tutto ciò che è partito prima di lui non può essere registrato. systemd può registrare diagnostica di qualsiasi servizio anche se non è ancora partito il log di quel particolare servizio

Si propone di sostituire molti moduli (init, udev, crond/atd...) ma in realtà, come da modello UNIX, è un framework che delega compiti a sottocomponenti. Questi sono detti **control unit.** I cui nomi seguono la convenzione di *name.type.* *type* indica lo scopo dell'unit, alcune unit gestiscono i demoni (*Service*), altre le comunicazioni tra processi (*Socket*) , altre i target che sostituiscono il concetto di runlevel (*Target)*.

Le unit sono file di testo, quelli tipici di riferimento (anche da copiare) sono in /lib/systemd/system/. Quando si installano pacchetti, vengono forniti anche i file necessari alla configurazione dei demoni, posti in /usr/lib/systemd/system/ (tipicamente sono link a quelli di riferimento). I file con le personalizzazioni sono in /etc/systemd/system e sono sempre prioritari rispetto alle definizioni di sistema.

systemctl è il comando con cui si controlla il funzionamento del sistema (interfaccia a systemd). Nasce per gestire gerarchie di servizi rispettando le dipendenze tra uno e l'altro. Segue formato ***systemctl COMANDO nomeservizio*** COMANDO è {***start***|***stop***|***status***|***restart***|***reload***}

con opzione -H [hostname] si connette a host remoto via ssh

I comandi da eseguire per ogni opzione sono definiti nelle unit con parametri (es. ExecStart), in generale systemd tiene traccia dei processi avviati con *start*, in modo che i loro PID possano essere usati come parametri nei comandi *reload/stop.* Inviare un segnale per fermare un processo in modo diretto generalmente è una cattiva idea, si usa ***systemctl stop*** per avere una corretta gestione (SIGTERM, poi SIGKILL dopo timeout: per demoni definiti da noi, è opportuno implementare handler per SIGTERM per garantire gestione di ***systemctl stop***).

Le operazioni di start, stop, status, restart sono volatili, ovvero non cambiano nulla nella configurazione del sistema. I comandi di {*enable*|*disable*|*mask*|*unmask*} viceversa hanno effetto persistente sulla configurazione e nessun effetto immediato, si usano per automatizzare avvio al boot e arresto allo shutdown di servizi. Se un processo dà fastidio serve sia *stop* per fermarlo subito che *disable* per evitare che riparta. *disable* lascia disponibile la possibilità di usare manualmente *start* sul servizio, *mask* neutralizza l'intera definizione della unit, impedendo anche il controllo manuale. Qualche esempio di altri comandi:

- ***systemctl list-units*** mostra tutte le unit gestite di tutti i tipi elencati

- ***systemctl -t TIPO*** mostra tutte le unit attive del tipo specificato (es. ***systemctl -t timers***)

- ***systemctl list-unit-files [-t TIPO]*** mostra tutte le unit installate del tipo specificato (esempio utile ***systemctl list-unit-files -t services*** tutti i servizi installati; aggiungendo ***--state=enabled*** elenca i servizi che partono al boot)

- ***systemctl --state STATO*** mostra tutte le unit che si trovano nello stato specificato (esempio utile ***systemctl –state failed*** tutte le unit che systemctl ha cercato di avviare senza riuscire)

Con systemd i runlevel sono rimpiazzati dai target, ***systemctl get-default*** restituisce il target di default (***systemctl set-default [target]*** per modificarlo). I target gestiscono le dipendenze tra le unità,

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut
queste partono appena sono soddisfatti i vincoli espressi dalle direttive. Una unit può avere direttive del tipo:

- ***Requires*** elenco di altre unit di cui questa necessita: se l'avvio di queste fallisce, questa viene arrestata (configurabile relazione temporale dopo, prima, simultaneamente)
- ***Wants*** dipendenze di questa meno strette di Requires, viene tentato l'avvio delle altre unit ma se fallisce, questa viene avviata comunque
- ***Conflicts*** vincolo negativo per rendere unit mutualmente esclusive
- ***OnFailure*** unit da avviare quando questa fallisce
- ***RequiredBy / WantedBy*** specifica dipendenza inversa: quando si installa questa unit vengono informate le altre che hanno bisogno di lei che esiste; aggiungendo automaticamente entry *Requires / Wants*
- ***Restart*** riavvia il servizio in caso di terminazione; è l'equivalente del respawn di init che si occupava di garantire il riavvio dei servizi fondamentali (es. servizio per gestione terminale)

Alcune unit speciali (principalmente target) hanno nomi fissi e funzioni fondamentali, usate come punti di controllo nella sequenza di boot (vedi ***systemd.special(7)*** e ***bootup(7)***).

Scrittura di unit file

Volendo scrivere un unit file, le configurazioni sono molte (vedi ***man 5 systemd.service*** o https://www.freedesktop.org/software/systemd/man/systemd.service.html) ma gli elementi davvero indispensabili sono:

- [Unit]
  - Description= Descrizione
  - Requires/Wants= da chi dipende questa unit
  - Documentation= inserire indicazione su dove trovare documentazione, agevola la vita al sistemista
- [Service]
  - Type= tipo di avvio
  - ExecStart= comando da lanciare all'avvio
  - Exec[Stop/Reload]= opzionali, comandi per stop e reload
  - Restart= opzionale, reazione alla terminazione
- [Install]
  - WantedBy= chi dipende da questa unit
  - Alias= nome con cui la unit è nota a sistemd

Le dipendenze vanno risolte direttamente in fase di design degli unit file; ad esempio se A non solo ha bisogno di B, ma deve anche attendere che sia partito, si aggiunge *Requires=B* e *After=B* alla sezione *[Unit]* di A (senza vincolo After, la semantica sarà la partenza in parallelo delle Unit con

verifica a posteriori del vincolo). Le dipendenze sono tipicamente inserite nei servizi e non nei Target.

Il parametro *Type* nella sezione *[Service]* specifica il tipo di start-up da considerare.

- Type=simple default, systemd considera il servizio avviato con successo appena ha forkato un figlio per eseguire il comando *ExecStart* (se la exec fallisce, non ce ne accorgiamo)
    - Non è adatto ad un servizio che deve ordinarne altri
- Type=forking fa eseguire un processo in background, con un pid che si può catturare e mettere in un file. Ricordiamo che quando si forka un figlio, il pid viene messo da parte perché si possa recuperare senza usare ***ps*** cercandolo con il nome, che potrebbe anche essere ambiguo
    - systemd considera il servizio partito quando il processo avviato con *ExecStart* esegue una sua fork e il genitore esce). Tipicamente usato per riutilizzare un classico demone UNIX, utile opzione *PIDFile=* per tracciare il processo principale.
- Type=oneshot per lanciare un processo una volta e poi uscire, es. gestione interfaccia di rete fa ciò che deve una sola volta e poi esce.
    - Si può settare *RemainAfterExit=yes* così che systemd possa considerare il servizio come attivo dopo la sua uscita.

Tutti i servizi usano *ExecStart* per lanciare un comando, se è una riga di shell andrà eseguita con *sh -c 'comandi'*. Per supportare il comportamento di *reload* (sarebbe diverso da *restart* che è stop+start) è necessario specificare *ExecReload* per gestire il segnale lanciato da systemd. Nella configurazione avremo ad esempio ***ExecReload=kill -HUP $MAINPID***

Per un servizio persistente (tipo *simple*) è comune non specificare *ExecStop*. I servizi di tipo oneshot (es. gestione interfaccia di rete) tipicamente quando avviati configurano qualche aspetto del sistema, e quando arrestati ripristinano lo stato precedente. Avremo quindi ad esempio ***ExecStart=/usr/bin/mio-configuratore ExecStop=/usr/bin/mio-de-configuratore***

Con opzione *Restart* si può delegare systemd a monitorare il processo avviato. Se il processo termina per cause diverse da systemctl (diverse da *systemctl stop* altrimenti loop infinito, è necessario un watchdog per fare questo tipo di controllo), systemd lo riavvierà o meno a seconda della combinazione tra il valore di *Restart* e la causa di malfunzionamento rilevata. I timeout dei watchdog vanno gestiti separatamente con le loro unit (un watchdog è un processo indipendente che ogni tanto va a controllare se un processo risponde).

Le configurazioni default delle unit di sistema (si trovano in ***/lib/systemd/system***) sono molto utili, anche da copiare quando dovremo realizzare servizi che si comportano in un certo modo. Esempio oneshot: lancia comandi di configurazione con start e termina ripristinando lo stato precedente con stop, forking: fare qualcosa e restare in background

- oneshot: /lib/systemd/system/networking.service
- forking: /lib/systemd/system/dnsmasq.service
- con opzione Restart:
    - /lib/systemd/system/atd.service non ha type perché è di tipo simple, viene lanciato e sorvegliato automaticamente: basta fare il file service e poi fare enable perché sia monitorato

- /lib/systemd/system/cron.service Restart=on-failure significa che se viene terminato in modo non pulito, ci penserà systemd a riavviarlo automaticamente
- /lib/systemd/system/rsyslog.service

Monitoraggio

Il logging (che tiene una traccia dettagliata dell'attività dei demoni) e la diagnostica istantanea (comandi per sondare lo stato corrente delle risorse del sistema) sono necessari per tracciare se sono avvenuti eventi necessari al corretto funzionamento dei servizi, o diagnosticare i problemi successi.

I log (diari) tenuti dal sistema sono indispensabili per la diagnostica, anche per rilevare attività malevole o sospette. Vanno replicati su macchine remote sia per garantirne la sicurezza, sia perché se abbiamo più macchine, possiamo guardarli tutti in uno stesso posto e non cercarli in più punti.

systemd ha integrato il logging di Linux, tenuto in un journal: questo è binario, quindi va prima passato dal comando ***journalctl*** per renderlo visualizzabile in un dato formato.

syslog è la versione originale, i cui principi di base sono mantenuti dalle evoluzioni:

- I messaggi dai processi vengono serializzati, poi
- viene posto un timestamp: importante per stabilire relazione causale tra due eventi, quando si diagnostica un problema, poi
- vengono classificati: questo permette di gestire dove finiscono i messaggi (magari alcuni di uso e alcuni di diagnostica) inviandoli a diverse destinazioni

Ogni messaggio è etichettato con una coppia ***<facility>.<priority>***, ovvero argomento (*auth, authpriv, cron, daemon, ftp, kern, lpr, mail, news, syslog, user, uucp,* ***local0...local7***. Da local0 a local7 utilizzabili a piacere, se una delle nostre diagnostiche non rientra nella categorie precedenti) e importanza (in ordine decrescente, dal più importante *emerg, alert, crit, err, warning, notice, info, debug*). Le destinazioni possibili sono

- File indicato con nome assoluto
- STDIN di un processo, identificato da pipe verso il programma da lanciare
- Utenti collegati username, o * per tutti
- Server di syslog remoto @indirizzo o @nome
  - A default comunicazione su UDP porta 514, dove ci sarà un suo syslog che gestisce, l'unica cosa che non potrà fare è fare un altro salto (permesso 1 solo salto)

/etc/syslog.conf contiene le regole di configurazione per lo smistamento. Ogni riga è una regola formata da ***<etichetta di interesse> <destinazione>.*** Le priority vengono trattate in modo che una regola che ne specifica una faccia match con tutti i messaggi di tale priority e superiori (a meno che questa non sia preceduta da =, in quel caso match solo esatto) e c'è priority speciale *none* per ignorare i messaggi con la facility specificata prima del punto. Esempi:

- *kern.* */dev/console* invia su device speciale che fa vedere i messaggi sulla console tutti i messaggi di facility kern di qualsiasi priorità
- **.info;mail.none;* */var/log/messages* invia su quel file messaggi con qualunque facility con importanza info o superiore, con esclusione di quelli che hanno facility mail

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut

- **.emerg* * invia ai terminali di tutti gli utenti messaggi con priority pari o superiore ad emerg
- *kern.crit* *"|/usr/bin/alerter"* | significa che lancia processo figlio e tutti messaggi vengono inviati su stdin del processo indicato
- **.=warning @loghost* invia al server di nome loghost SOLO i messaggi con priority warning

rsyslog è un'evoluzione, può ricevere messaggi anche da socket, api di identificazione... Può scrivere non solo su host remoti, file utenti; ma anche direttamente su qualsiasi database; può lanciare un programma e passargli parametri. È modulare e carica solo le funzioni necessarie. Ad esempio

- attivazione della ricezione di messaggi via rete (v8.4 / v8-16):
  - ***$ModLoad imudp*** / ***module(load="imudp")***
  - ***$UDPServerRun 514*** / ***input(type="imudp" port="514")***
- integrazione del kernel logging
  - ***$ModLoad imklog*** / ***module(load="imklog")***

Le sue direttive globali si trovano in ***/etc/rsyslog.conf*** , quelle specifiche in file separati sotto ***/etc/rsyslog.d/*** (il nome del file generale è quasi uguale a syslog, cambia solo la r. Va a leggere nella directory ***/etc/rsyslog.d/***: se devo automatizzare (config automatica di uno script per syslog) è più robusto mettere un file in una cartella ed eventualmente rimuoverlo, che modificare una riga del file principale. Permette di scartare messaggi ponendo ~ come destinazione. Può fare output complesso, sfruttando template per definire canali; offre anche TCP logging e può passare il messaggio come parametro a un programma.

Differenza da cron: ogni utente può riconfigurare la propria tabella di cron; mentre la riconfigurazione del syslog è possibile solo per root: la cartella syslog.d/ non è scrivibile da utenti standard, il comando *systemctl restart syslog* non è disponibile per utenti standard. Sarebbe possibile dare solo questi particolari permessi usando il file sudoers; ma anziché dare ad un utente questi poteri, potremmo dargli la possibilità di usare un comando che rinomina un file da *myconf.off* a *myconf.conf*. Avendo tanti snippet pronti all'uso nella cartella, che non finiscono con .conf e quindi non vengono presi in considerazione.

logger è il comando per inviare messaggi di log a rsyslog, quello che si invia al syslog non è esattamente ciò che finisce nel file: viene aggiunta una serie di informazioni utili alla gestione del sistema:

- un timestamp,
- il nome della macchina che ha prodotto il messaggio,
- il nome dell'utente che ha prodotto il messaggio, poi duepunti e il messaggio vero e proprio
- **formato:** *Lun 16 15:36:56 hostname utente: testomessaggio*

logger -p <facility.priority> "testomessaggio" invia a syslog un messaggio con l'etichetta specificata

opzione **-t** permette di scegliere come taggare il messaggio (a default, il tag è il nome utente)

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut
opzione **-n** permette di inviare messaggi ad un server di logging remoto, senza passare da quello locale

I comandi essenziali per il monitoraggio del sistema sono ***ps, top, uptime*** per i processi; ***df, du, free*** per lo spazio in memoria, ***fuser*** per i file. Mettendoli in script, si hanno fotografie in tempo reale di cosa succede nel sistema. La maggior parte di essi sono interfacce verso il filesystem /proc, che appare come FS ma in realtà occupa 0 byte. Quando si prova ad accedere a /proc, in realtà si stanno chiedendo informazioni al sistema.

– ***ps***: stato dei processi

– ***uptime***: carico del sistema

– ***free***: occupazione memoria

ps ha moltissime opzioni avendo tre diverse sintassi (UNIX singolo trattino e singole lettere, BSD singole lettere senza trattino, GNU estensioni con parole precedute da doppio trattino). Per gli usi più comuni all'inizio della man page abbiamo

- la sezione PROCESS SELECTION BY LIST mostra come ottenere una lista di processi secondo le loro proprietà (es. comando lanciato, pid, utente, ecc.), molto meglio che fare *grep* dell'intera lista di processi
- la sezione OUTPUT FORMAT CONTROL mostra come formattare la lista prodotta (in particolare le opzioni (equivalenti) -o, o, --format seguite da una stringa di specificatori documentata nella sezione STANDARD FORMAT SPECIFIERS permettono un controllo completo sui campi che si vogliono far comparire nella lista)
- la sezione PROCESS STATE CODES spiega il significato della colonna STAT e dà un'indicazione fondamentale dello stato del processo

ps aux mostra tutti i processi dell'utente, ordinati in base all'utente del processo, e include i processi non agganciati ad un terminale (es. crond). Opzione ***w*** mostra la riga di comando completa che ha originato il processo, ***f*** mostra i rapporti di discendenza tra i processi, ***h*** rimuove header dall'output

uptime riporta il tempo trascorso dal boot e il carico del sistema: lo scheduler registra il numero totale di processi in stato R (runnable) o D (uninterruptable sleep). Questo viene mostrato come media su tre periodi diversi. Formato output :

21:27:56 up 7:10, 2 users, load average: 0.00, 0.00, 0.00

ORA , tempo trascorso dal boot, numero utenti connessi, carico medio negli ultimi 1/5/15 minuti

free mostra informazioni sulla memoria. La maggior parte della memoria usata per cache può essere liberata per usi prioritari, per cui available ≈ free + buff/cache. L'impatto sulle prestazioni della rinuncia alle cache non è nullo. Used swap > 0 significa solo che in qualche momento è servita la swap

top riassume ps, uptime, free + uso dettagliato cpu, è interattivo e viene aggiornato regolarmente, permette di interagire coi processi. Utile per stima intuitiva dello stato di salute, esaminando lo stato della CPU:

- CPU scarica – molti processi in D → un dispositivo non risponde
- CPU usata principalmente in userspace – sistema CPU-bound

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut

- CPU usata principalmente in iowait – sistema I/O-bound (possono esserci periferiche lente ma anche sovraccaricate per altri motivi, esempio swap molto usata in sistemi memory-bound)
- indagini più approfondite con ***vmstat*** e ***iostat*** disponendo di una baseline (valori tipici misurati durante uso ottimale del sistema)

Hardening e sicurezza di base

La sicurezza può essere gestita in modo efficace solo se si vieta ogni comportamento non esplicitamente consentito (default deny) e se ciò che è consentito viene svolto con i minimi permessi necessari (minimum privilege). Sicurezza comprende **riservatezza, integrità, autenticità e disponibilità**, tutte da individuare e difendere caso per caso a seconda dei sistemi e delle informazioni trattate. Solitamente ci si preoccupa di difendere un sistema dagli attacchi via rete a software come SO e applicazioni, le contromisure sono facili da scavalcare con accesso fisico al sistema: prima di tutto va considerata la sicurezza fisica del server: lo storage potrebbe essere sottratto, potrebbero essere connessi apparati di raccolta dati alle interfacce, potrebbe essere avviato un sistema operativo arbitrario. In ambito cloud queste preoccupazioni cambiano faccia ma sussistono ancora. Se la collocazione della macchina è fuori dal controllo diretto, si può scegliere un case che si possa chiudere e fissare al rack, installare dispositivi di rilevazione delle intrusioni, disabilitare periferiche non utilizzate.

Il *default deny* inizia dall'installazione, scegliendo solo software strettamente necessario, evitando pacchetti ignoti e di dubbia utilità.

Il processo di avvio

Per andare a regime il sistema segue un processo di boot, dove tipicamente

- il BIOS individua i dispositivi di possibile caricamento di bootloader per esaminarli in un certo ordine
- il bootloader sceglie il sistema operativo (sia BIOS che bootloader possono avere password)
- il sistema operativo carica di device driver e lancia il processo init
- init avvia i servizi necessari all’inizializzazione del sistema nell’ordine corretto

Si può avere password sia sul BIOS che sul bootloader. Se l'avvio del sistema richiede una password può essere difficile il funzionamento non supervisionato (es. mancanza di alimentazione e impossibilità a ripartire per ore), ma è importante proteggersi almeno dai cambi di configurazione, e inoltre non contare su un unico strato di protezione (le password del BIOS sono solitamente resettabili, o si possono indovinare).

I bootloader tipici sono LILO (dalle origini) e GRUB (più potente e flessibile, ha una shell che permette di eseguire comandi per modificare al volo la procedura d'avvio, permettendo abusi).

Entrambi hanno delle direttive per specificare password globali (richieste al boot) o specifiche per alcuni item (password richiesta solo per alcune immagini).

Per garantire l'esecuzione di software autentico, è necessaria una *chain of trust*: anti-malware verifica applicazioni, a sua volta è verificato da SO, a sua volta verificato dal bootloader. Per verificare il bootloader, il BIOS si avvale di HW speciale non modificabile dal SO quindi immune a infezioni: per garantire la *root of trust* si segue l'ordine del *trusted boot* (basato su un modulo TPM Trusted Platform Module, un chip con funzioni crittografiche) o del *secure boot* mediante UEFI (che può usare TPM per velocizzare i controlli).

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut
UEFI è la standardizzazione di EFI, interfaccia tra SO e firmware creata da Intel per avere più flessibilità di BIOS. E' una sorta di pre-sistema operativo, molto più ampio di BIOS avendo qualche milione di righe di codice, e si occupa di verificare la legittimità di ogni componente software prima di passare il controllo al bootloader e poi al SO, mediante un database di chiavi, bloccando il boot in caso di difformità. Essendo queste chiavi depositate in hardware, all'inizio era possibile autenticare solo sistemi commerciali, ma la fondazione Linux ha poi ottenuto da Microsoft una chiave che permette di verificare sistemi aperti.

Permettere l'accesso alle risorse passa attraverso le fasi di autenticazione (verificare se soggetto ha credenziali valide per identificarlo) e autorizzazione (decidere se soggetto ha autorità per effettuare operazione). L'autenticazione ideale dovrebbe disporre di tre prove: qualcosa che sei (conferma identità confrontando caratteristiche fisiologiche con dati "biometrici" di riferimento), qualcosa che hai (conferma identità mediante possesso di oggetto fisico riconoscibile dalla macchina, come token RFID, schede a banda magnetica), qualcosa che sai (conferma identità dimostrando di conoscere un segreto -password, chiave-).

Utenti e file

Gli utenti sono i soggetti di tutte le operazioni svolte sul sistema, utilizzati per determinare i permessi di accesso a qualsiasi risorsa (oggetto). Anche i gruppi possono avere diritti specifici per delle risorse, anche essi sono soggetti: ogni utente appartiene ad almeno un gruppo (quello con cui si trova ad operare appena fa login) e può appartenere anche a più gruppi, durante una sessione di lavoro può cambiare identità di gruppo tra quelli di cui fa parte.

useradd serve per creare nuovi utenti, con granularità fine e grande possibilità di automazione. I valori predefiniti per le caratteristiche dell'utente creato stanno in */etc/login.defs*. I parametri principali sono

- *-m* crea la home dell'utente seguendo come template i file dentro */etc/skel*
- *-s* assegna la shell all'utente, le possibili shell sono indicate dentro il file */etc/shells*, altrimenti prende il default
- *-U* crea un gruppo con lo stesso nome dell'utente
- *-K* con questo parametro è possibile specificare la UMASK=0077
- *-G* per assegnare l'utente ad un gruppo supplementare (es. sudo) alla creazione

Le credenziali degli utenti sono in

- */etc/passwd*, formato prandini:x:500:500:Marco Prandini:/fat/home:/bin/bash
  - Accessibile a tutti, tutti i dati pubblici dell'utente. Ogni riga è un record, i diversi attributi separati da :
  - Primo argomento è username, il secondo parametro è una x che ricorda la retrocompatibilità con il campo con password, importante tenerlo così che il sistema cerchi corrispondenza in /etc/shadow
  - Poi userid numerico e groupid numerico (relativo al gruppo principale, quello associato al momento del login).
  - Poi campo informazioni, home directory dell'utente, cwd (current working directory) da cui parte all'avvio il sistema

- Infine il programma avviato normalmente al login utente (solitamente è una shell, ma potrebbe anche essere un programma diverso. Così si può limitare utente ad un solo programma, che se scritto bene impedisce di uscire). Altra possibilità è /bin/false, programma che appena lanciato termina con esito falso. Questo è usato per gli utenti creati in automatico all'installazione di pacchetti, con l'unico scopo di far girare un demone con l'identità di quell'utente. Così facendo non si dà shell interattiva, se qualcuno dovesse riuscire a fare login come questi utenti, avrebbe bin/false e terminerebbe immediatamente

- */etc/shadow*, formato
  prandini:1/PBy29Md$kjC1F8dvHxKhnvMTWelnX/:12156:0:99999:7:::

  - Linee corrispondenti a passwd ma accessibile solo a root. Spostando le password offuscate dal file passwd, si rende più difficile tentare di indovinarle essendo shadow accessibile solo a root.

  - Lasciare passwd pubblico è pratico: ad esempio per leggere gli id numerici utente, da usare in diversi contesti per identificare gli utenti. come quando facciamo ***ls -l*** ci dice l'id utente, non il nome

  - Contiene dati sulla validità temporale della password, esaminabili e modificabili con ***chage:*** quando si crea un nuovo utente, i valori di default vanno presi da qualche parte: alcuni sono in /etc/login.defs (come PASS_MAX_DAYS numero massimo di giorni prima che vada cambiata la password, e PASS_MIN_DAYS numero minimo di giorni prima che si possa fare un nuovo cambio dopo l'ultimo effettuato), altri in /etc/default/useradd (come EXPIRE, scadenza dell'account), utili per configurare il sistema con valori arbitrari.

  - È importante non avere password deboli, magari impedendole con PAM. *pam_cracklib.so* è la libreria per il controllo della robustezza delle password (all'atto della scelta di queste), e nei file */etc/pam.d/system-auth* o */etc/pam.d/common-password*, nella riga che inizia con *password requisite* si possono specificare i parametri *minlen* (lunghezza minima) e *Xcredit* (X è tra *l u d o* , per lower/uppercase, digits, otherchars: lunghezza minima = minlen – i credits per ogni carattere indicato con il suo peso). Permette inoltre di indicare quanti caratteri devono essere diversi dalla precedente (*difok)* e quante password precedenti vengono memorizzate (*remember*=); ma anche di avere lockout dopo tentativi sbagliati con modulo *pam_tally*:

    1. auth required pam_tally.so onerr=fail no_magic_root
    2. account required pam_tally.so deny=5 no_magic_root reset

La prima riga abilita conteggio dei tentativi falliti, la seconda blocca account dopo N=5 tentativi. Un login corretto resetta il contatore, comando ***faillog*** permette di esaminare stato account e riattivarlo.

In Linux è possibile settare limiti di uso per varie risorse per prevenire DOS; i limiti possono essere hard (set da root, non superabili ad utente) o soft (valori di default, modificabili da utente entro il massimo permesso dal SO e dal limite hard). La configurazione può essere locale ad ogni shell (comando ***ulimit***) o applicata ad ogni login con modulo *pam_limit* (file */etc/limits.conf*, forma *<domain> <type> <item> <value>* Domain user, @group o *; type hard, soft o – (both), item ha una sua tabella di limiti (1_hardening_access_control slide 31).

L'appartenenza ai gruppi viene verificata unendo informazione in */etc/passwd* (che contiene gruppo principale di ogni utente) e il contenuto del file */etc/group*, leggibile da tutti, una riga per gruppo

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut
(formato sudo:x:27:prandini , il campo 4 dovrebbe contenere i membri del gruppo, ma non necessariamente li tiene tutti; bisogna leggere entrambi i file per scoprire tutti i membri del gruppo, esempio prandini:x:500:)

id <USER> riporta tutte le informazioni di identità.

usermod permette di modificare, coi suoi diversi parametri, tutte le caratteristiche dell'utente (come useradd ma può essere usato solo da root. Esiste anche una serie di comandi specifici per cambiare singole proprietà, usabili da utenti standard per agire sul proprio account:

chsh modifica la shell di login

chfn modifica del nome reale

passwd modifica della password

adduser non è standard in tutte le distro e chiede i dettagli interattivamente, quindi utile se usato in maniera estemporanea, ma non se dobbiamo scriptare creazione utenti

groupadd e ***addgroup*** per la creazione di gruppi, analogamente (primo scriptabile, secondo interattivo)

gpasswd modifica password e lista utenti di un gruppo

getent interroga il db utenti o gruppi (magari utenti sono memorizzati in un db centralizzato accessibile via rete, è possibile configurare macchine per autenticare utenti interrogando il db centralizzato)

last mostra i login effettuati sul sistema, ***lastlog*** mostra la data di ultimo login di ogni utente, ***faillog*** mostra i login falliti sul sistema.

L'idea base del controllo dell'accesso è decidere se un soggetto può eseguire una certa operazione su un oggetto: il modo più banale di esprimere i permessi sarebbe una matrice completa; ma in un sistema reale si avrebbero migliaia di soggetti e milioni di oggetti, con la maggior parte delle celle al valore di default. Per risparmiare spazio, si può partizionare la matrice in due modi:

- Per soggetto: capability lists, una lista associata a ogni soggetto che contiene solo gli oggetti con permessi diversi dal default
- Per oggetto: *access control lists* ACL, una lista associata ad ogni oggetto del sistema, contente i soli soggetti con permessi diversi dal default. Esplicitamente implementata in POSIX e MS Windows, il filesystem UNIX tradizionale ha una ACL con 3 soli soggetti: utente proprietario U, gruppo proprietario G, others O (gruppo implicito di tutti i soggetti non U e non appartenenti a G)

Il controllo dell'accesso viene gestito secondo due modelli

- DAC (Discretionary Access Control), ogni oggetto un proprietario e questo decide i permessi, le ACL sono esempio tipico (***setfacl*** per impostare, ***getfacl*** per visualizzare: ***ls -l*** mostra un + se ACL è presente per un file).
- MAC (Mandatory Access Control), la proprietà di un oggetto non consente di cambiarne i permessi, c'è una policy centralizzata da un *security manager*, espressa di solito con qualche capability list

Ogni file (regolare, directory, link, socket, block/char speciali) è memorizzato in un i-node, che mantiene anche informazioni sulle autorizzazioni: un (solo) utente proprietario del file e un (solo) gruppo proprietario del file, oltre a 12 bit che rappresentano permessi standard e speciali.

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut
I 9 bit meno significativi sono i bit di accesso (User,Group,Other); i 3 più significativi sono speciali, in ordine di importanza: SUID (set user id), SGID (set group id), sticky.

Il significato dei bit di accesso cambia tra file e directory, ma è deducibile ricordando che una directory è un file che contiene un DB di coppie (nome,i-node; i cosiddetti hardlink (link di filesystem), nome dato al file - inode che rappresenta quel file nel filesystem)

- R read: permette la lettura di un file, per le directory equivale a leggere la prima colonna (elenco dei file)
- W write: scrittura dentro un file, aggiunta/cancellazione/rinomina di file in una directory (permesso W in una directory consente a un utente di cancellare file sul cui contenuto non ha alcun diritto)
- X execute: esegue il file come programma, permette lookup tra nome e inode della directory (se ho la R posso vedere i nomi, se ho la X posso vedere quali inode corrispondono a un nome: averli entrambi su una directory permette di vederli e recuperarne l'inode. Per avere l'accesso a un file -che richiede lookup di tutti gli i-node corrispondenti ai nomi delle directory nel path- serve il permesso X per ognuna, mentre R non è necessario.)

Assegnazione di ownership

- viene fatta alla creazione di un file, l'utente creatore è assegnato come proprietario del file, il gruppo attivo dell'utente creatore è assegnato come gruppo proprietario (default gruppo predefinito da /etc/passwd, gruppo attivo può essere cambiato con un altro tra quelli di appartenenza con comando ***newgrp***, comando locale alla shell in cui viene chiamato).
- Per cambiarla dopo la creazione del file, si usano
  - ***chown [new_owner]:[new_group] <file>*** modifica owner e/o group owner del file
  - ***chgrp [new_group] <file>*** cambia group owner del file (comunque solo tra quelli di cui l'utente è membro; con opzione -R attiva ricorsione su cartelle)

Assegnazione dei permessi

- alla creazione di file viene fatta seguendo logica negativa, ovvero con tutti quelli sensati meno quelli della umask. Quelli sensati sono diversi per file (rw-rw-rw- oppure 666) e directory (rwxrwxrwx oppure 777), perché per un file non è mai ragionevole assegnare automaticamente bit X, ma se una directory non ha X, invece, serve a poco. La umask può quindi essere unica e specificare quali permessi non concedere. Poiché in Linux il gruppo di default di un utente è uguale all'utente stesso, una umask sensata è 006 (toglie agli other lettura e scrittura: file per uso personale avranno gruppo uguale all'utente, così facendo si apre strada alla collaborazione non essendo necessario cambiare i permessi se si vuole condividere, basta cambiare gruppo quando lo si crea). Col comando ***umask*** si può interrogare e settare interattivamente nella sessione corrente, per rendere persistente la scelta si usano i file di configurazione della shell.
- Successivamente alla creazione, si usa ***chmod*** per modificare i permessi:
  - modo numerico in base ottale, es. 2770 (la cifra meno significativa riguarda le conf per gli other, la seconda meno sign. per il gruppo, la terza per l'utente, la quarta per i bit speciali quindi es.
    1. chmod 2770 miadirectory sarà ~~SUID~~ SGID ~~STICKY~~ rwx rwx ---
    2. chmod 4555 miocomando sarà SUID ~~SGID~~ ~~STICKY~~ r-x r-x r-x

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut

- modo simbolico utile per modifiche a singoli bit
    1. soggetto (a all, u user, g group, o other) dice in quale terzina stiamo operando
    2. operatore (= setta precisamente, con + aggiungiamo permesso, con – rimuoviamo)
    3. permesso (r read, w write, x execute, s special)
    4. esempio *chmod 'a=r,g-rx,u+rw' file* all esatto ad r, group privato di r e x, user dotato di r e w

Il sistema controlla i permessi secondo uno schema: se utente A vuole fare un'operazione su file di utente U e gruppo G; prima si controlla se A è U (in tal caso finisce il controllo e si applicheranno RWX di U), solo in caso negativo poi si controlla se A appartiene a G (in tal caso si usano RWX di G); altrimenti si usano RWX di O. Quindi se A è U e appartiene a G, i permessi associati a G sono ignorati anche se più favorevoli.

Ogni programma avviato da U appartenente a G, parte con quattro valori di identità:

- real user id (ruid) U,
- real group id (rgid) G,
- effective user id (euid), identità assunta dal processo per operare con soggetto diverso da U
- effective group id (egid), stessa cosa ma per identità di gruppo assunta dal processo per operare come soggetto diverso da G

Bit speciali:

- Applicati ai file hanno senso solo se eseguibile, possono fare in modo che euid e/o egid siano diversi dai corrispondenti ruid/rgid. :
    - bit 11 SUID 1 fa sì che un programma eseguito da utente qualsiasi, venga lanciato in processo che gira con identità dell'utente proprietario.
    - bit 10 SGUID stessa cosa ma per identità del gruppo (processo lanciato con gruppo proprietario del file).
    - bit 9 STICKY bit è obsoleto, dice al SO di tenere in cache una copia del programma…
- Applicati alle directory:
    - bit 11 SUID non è usato
    - bit 10 SGID serve a poter cambiare automaticamente membership di un utente quando entra a lavorare in una directory:
        1. precondizione: SGID settato, l'utente appartiene anche al gruppo proprietario della directory
        2. effetto: l'utente assume come gruppo attivo quello proprietario della directory, i file creati hanno quel gruppo proprietario
        3. vantaggi, mantenendo una umask 006: nelle aree collaborative i file sono automaticamente resi leggibili e scrivibili da tutti i membri del gruppo. Nelle aree personali i file sono privati perché sono proprietà del gruppo principale dell'utente, che contiene solo lui

- bit 9 Temp le directory temporanee (quelle world-writable predisposte perché le applicazioni dispongano di luoghi noti dove scrivere), hanno un problema, chiunque può cancellare un file (permesso W su directory dà cancellazione). Temp settato a 1 impone che i file nella directory siano cancellabili solo dai rispettivi proprietari.

Attributi: usati principalmente per tuning del filesystem, alcuni hanno rilevanza per la sicurezza: *a* append only, impedisce il taglio dei logfile; I immutable vieta cancellazione, creazione di link, rinomina e scrittura, utile per file di sistema; s secure deletion sovrascrive con 0 i blocchi dei file cancellati, valida contro strumenti in linea. ***chattr*** per modificarli, ***lsattr*** per visualizzarli.

Comandi utili per lavorare su file

- Elenco e navigazione:
  - Il filesystem Linux è un unico albero anche con più dischi fisici, la radice è '/' e tutti i path completi iniziano da questa.
  - ***pwd*** mostra directory di lavoro (present working directory) e tutti i path relativi si riferiscono alla pwd, i processi continuano ad occupare la pwd dov'era l'utente che li ha lanciati.
  - ***cd*** <*destinazione*> per cambiare directory (homeutente quando senza argomenti, la directory dove si era prima dell'ultimo cd con opzione '-').
  - In ogni directory D si hanno le subdir '.' che coincide con D e '..' che coincide con la dir superiore
- Analisi dei metadati:
  - ***ls*** per elencare contenuto di directory, opzioni
    - -l abbina al nome le info associate al file
    - -a non nasconde i file che iniziano per '.' (per convenzione sono quelli di configurazione)
    - -A come -a ma nasconde i file '.' e '..'
    - -F postpone carattere '*' a eseguibili e '/' alle directory
    - -d lista nomi dir senza listarne contenuto al contrario del default, può essere utile quando i nomi sono espansi partendo da wildcard
    - -R percorre ricorsivamente la gerarchia
    - -i mostra gli i-number dei file oltre al nome
    - -r inverte l'ordine elenco
    - -t lista i file in ordine di data/ora di modifica (dal più recente)
    - -1 forza output singola colonna
  - Metadati per ls -l: formato -rw-r--r-- 1 root root 1514 Mar 29 11:07 /etc/passwd
    - primo bit indica il tipo: - file standard, d directory, l link simbolico, b block special, c character special, p named pipe (FIFO), s socket
    - il numero dopo i permessi indica i link allo stesso inode, possono aversi più nomi per lo stesso inode

        - seguono utente e gruppo, dimensione, e data di modifica (se l'ultima è avvenuta oltre un anno fa, verrà mostrato l'anno invece che l'ora)
        - **Link simbolici**: anche detti softlink, sono un nome diverso da quello principale dato a un file - a differenza dell'hardlink, memorizzato in una directory che punta a un'inode-. Un softlink è un file di nome X con inode Y; dentro questo file c'è solo una linea di testo: il nome di un altro file Z con inode F. Spesso link simbolici creano problemi per manipolazioni massicce di copia e simili: a volte è utile prendere il link simbolico in quanto file a sé, altre volte si vuole fare l'operazione in modo trasparente (quindi operare non sul file in sé ma sul puntato).
            - L'hardlink referenzia un inode direttamente: si possono creare hardlink SOLO a inode esistenti (il sistema fa lookup dell'inode per creare l'hardlink). Il softlink invece può puntare a file inesistenti, perché non c'è il controllo che il file indicato abbia effettivamente un inode. Softlink fa riferimento ad un nome, non ad un inode.
            - I softlink sono utili per creare link tra diversi filesystem (hardlink fanno riferimento ad inode, i cui numeri sono locali al filesystem. Ma si possono tranquillamente creare connessione con softlink a nomi logici di altro filesystem)
    - Ogni file ha 3 o 4 timestamp distinti Xtime (
        - *mtime* modifica contenuto,
        - *atime* accesso al contenuto,
        - *ctime* modifica metadati,
        - *wtime* creazione del file, se supportato)
        - gestiti dal FS, possono essere cambiati a mano con ***touch***
        - per estrarre i metadati, comando ***stat***, formato
            - **stat --format='%U %a %z' filepath** con U utente proprietario, a permessi, z ctime
- Creazione e generazione file
    - ***rm*** – cancella un file o meglio rimuove il link (verrà cancellato quando link = 0, conteggio = n link su filesystem + n FD aperti)
    - ***cp*** – copia 1+ file in una dir, attenzione con file device e softlink: copia concetto o contenuto?
    - ***mv*** – sposta 1+ file in una dir
    - ***ln*** – crea un link a file (a default hardlink, solo stesso FS e non verso directory. Con opzione -s symlink, senza limitazioni)
    - ***mkdir*** – crea dir
    - ***rmdir*** – cancella dir: deve essere vuota, con opzione -r cancella ricorsivamente (occhio alla ricorsione, rm -r /* pialla disco)
- Ricerca nel filesystem:

- ***find***, ricerca in tempo reale il FS, occhio al carico indotto. Si possono combinare più criteri di ricerca, ad esempio
  - *-name 'expr'* Nome contiene espressione, es. '*.c' tutti i file che finiscono con .c
  - Timestamp entro periodo
  - *-size* +N-M dimensione entro limiti
  - Tipo specifico (file, dir, link simbolici)
  - Di proprietà di utente o gruppo specificato, o "orfani"
  - Permessi di accesso specificati
  - Esempio: ***find /usr/src -name '*.c' -size +100k -print***
    - ricerca sotto /usr/src tutti i file che finiscono per .c, hanno dimensione maggiore di 100K, ed elencarli sullo standard output
  - E' possibile eseguire un comando su ogni oggetto individuato con l'opzione *-exec comando {} \;* (le graffe vengono espanse con il nome di ogni file, \; serve a find per individuare fine comando)
    - Es. ***find / -type f -nouser -mtime -2 -exec grep -l TXT {} \;***
    - Find trova in / i file regolari, orfani, modificati meno di 2 giorni fa (2*24H), su ognuno di essi viene invocato *grep* che ricerca TXT nel contenuto
- Il comando ***locate*** ricerca su un db indicizzato (DB aggiornabile con ***updatedb***), con meno carico sul sistema rispetto a *find*, ma si può specificare solo pattern nel nome e le risposte possono essere obsolete per modifiche effettuate dopo l'esplorazione.
- Comando ***file*** per identificare il contenuto di un file, fa tre test
  - 1) usa ***stat*** per capire se il file è vuoto o speciale
  - 2) usa il DB dei "magic number" (verifica dell'header per vedere il magic number)
  - 3) usa metodi empirici per capire se è un file di testo, in tal caso quale sia la lingua naturale o linguaggio di programmazione
- Due formati dei file di testo:
  - nei sistemi UNIX le linee sono terminate da un solo carattere, line feed o LF o \n o 0x0A
  - nei sistemi DOS/Windows le linee sono terminate da due caratteri: carriage return line feed o CRLF o \r\n o 0x0D0A
  - senza opportuna conversione i file di origine DOS hanno caratteri extra fine linea (visualizzati da editor di testo come ^M, possono causare errori negli script e nei file di configurazione). *Dopo aver messo in piedi un grande sistema che funzioni per lavorare nativamente su Linux... Un peccato che la gente metta roba scritta su blocco note, piena di caratteri che non permettono di eseguire gli script*

        - conversione: Ubuntu pacchetto ***tofrodos*** , comandi **todos** / **fromdos** (su altre distro es. *unix2dos, dos2unix)*

- Trasferimento dati da/per device (locali): i comandi più ovvi non sono pratici per trasferire dati con file speciali, ***cat*** e ridirezioni sono utilizzabili in modo "tutto o niente", ***cp*** non funziona.
    - ***dd*** può leggere/scrivere byte da qualsiasi file, bypassa filesystem scrivendo direttamente su memoria fisica, formato ***dd if=<NOME> of=<NOME>***
        - se NOME= - si intende STDIN/STDOUT.
        - *skip=<N>* da che punto leggere
        - *seek=<N>* in che punto scrivere
        - *count<N>* quanti dati trasferire
        - *bs=<N>* dimensione blocco
- Archiviazione e compressione:
    - Per archiviazione si intende prendere una gerarchia di directory e serializzarle (renderle un'unica stringa di byte, che tenga traccia sia della struttura gerarchica dei file che del contenuto). Per gestire più file agevolmente senza perdere i metadati di ognuno si usa ***tar***, con solo uno dei comandi:
        - -A concatena più tar
        - -c crea nuovo tar
        - -d trova differenze tra archivio tar e FS
        - -r aggiunge file a un tar
        - -t elenca contenuto di un tar
        - -u aggiorna file in un tar
        - -x estrae file da un tar
        - --delete cancella file da un tar
        - A default assume che l'archivio sia su /dev/tape, quindi si usa sempre opzione ***-f <FILENAME>***. Con FILENAME= - si può indicare o STDIN da cui leggere archivio con opzioni d,t,x ; o STDOUT su cui scrivere con opzione c
        - Altre opzioni:
            - -p conserva tutte le informazioni di protezione, ma utenti standard sono forzati a dare la loro ownership quando estraggono (funziona appieno solo con root)
            - -v verbose stampa i dettagli
            - -T <ELENCO> prende i file da archiviare da ELENCO invece che da parametri
            - *-C <DIR>* svolge tutto come dopo *cd DIR*

- --*files-from=nomelista* indica un file da cui leggere (uno per linea) la lista dei file da archiviare

- Esempi: non è necessario specificare '-' fintanto che non è necessario usare più di un'opzione che richiede parametri.
  - Creazione – ***tar cvpf users.tar /home/**** : la barra iniziale verrà rimossa in modo da rendere relativi i path
  - Estrazione – ***tar -C /newdisk -xvpf users.tar**:* poiché i path nell'archivio sono relativi, la directory home viene ricreata dentro /newdisk e tutta la gerarchia sottostante ricostruita
  - In pipeline – ***tar cvpf - /home/* | tar -C /newdisk -xvpf -***

- Compressione: tar non comprime, i formati di compressione più comuni in Linux sono
  - .gz con comando ***gzip*** (più comune)
  - .bz2 con comando ***bzip2***
  - .xz con comando ***xz***
  - I comandi base prendono un argomento file e lo comprimono aggiungendo estensione,
    - con opzione -d si decomprime ricreando il file e rimuovendo l'estensione
    - con opzione -c si riversa su STOUT invece che su file (filtro, si può redirigere l'output. Es. ***tar cf - * | xz -c > archive.tar.xz)***
  - tar ha opzioni per invocare decompressione
    - -z per gzip (estensione .tar.gz o .tgz),
    - -j per bzip2 (.tar.bz2 o .tbz2)
    - -J per xz (.tar.xz o .txz)
    - Es precedente ***tar cJf archive.tar.xz ****

- Copia massiva di file (anche remota):
  - Il trasferimento di gerarchie di file e cartelle, contenenti file non standard non è gestito correttamente da tutte le versioni di ***cp -a*** e ***scp -R*** (cosa fa cp quando incontra un link? cosa fa quando incontra file device?). ***tar*** archivia correttamente i metadati, ma richiede di archiviare → trasferire con scp su altro host → estrarlo nella cartella destinazione
  - ***rsync*** permette di effettuare più controlli, come la possibilità di non trasferire file già presenti a destinazione, o di configurare un comportamento per file speciali
    - sintassi base ***rsync [OPZIONI] SORGENTE DESTINAZIONE*** sorgente elenco di file e cartelle, destinazione cartella. Offre copia via rete
      - con protocollo nativo (necessario demone rsyncd)
        - ***rsync [USER@]HOST::SRCDIR DESTINAZIONE***

- ***rsync SORGENTE [USER@]HOST::DESTDIR***
  - con SSH (non richiede demone rsyncd), stessa sintassi di sopra ma con un solo : e non 2
- opzioni:
- Come copiare:
  - -l / -L copia i link come link/come file puntato
  - -p / -o / -g preserva i permessi, il proprietario, il gruppo
  - -t / -a / -N preserva i timestamp di modifica/accesso/creazione
  - -D preserva i file speciali
- Cosa copiare:
  - -u salta i file che sono più nuovi a destinazione o che a parità di età hanno stessa dimensione
  - -c salta i file che a destinazione hanno lo stesso checksum

Il backup è la copia dei dati dal sistema live ad un supporto offline, è impegnativo da organizzare ed eseguire ma è l'assicurazione contro qualsiasi causa di distruzione dei dati del sistema principale. Regola d'oro: 3-2-1-1 (3 copie dei dati, 2 media diversi, 1 copia offside (cloud) e 1 copia offline). Va pianificato, considerando tra le altre cose, cosa copiare (compromesso tra praticità di ripristino e tempi/spazi necessari), tipicamente si fa full backup del sistema appena messo in piedi, prima di metterlo in produzione, poi incrementali.

- Full backup è la copia completa di ogni singolo file nel/nei filesystem oggetto del backup, ingombrante quindi difficile farlo frequentemente
- Incremental backup è la copia dei soli file cambiati da una data di riferimento, tipicamente quella di esecuzione dell'ultimo full backup. Adatto all'esecuzione frequente a tendere verso ogni modifica di file (offre point-in-time restore, ma attenzione al carico dell'operazione di indicizzazione file). Per il ripristino servono sia il full che l'incremental. Si può fare a più livelli: Full → Incremental/level0/volume1 rispetto al full

Incremental/level1/volume1 rispetto a incremental/0/1

Incremental/level1/volume2 rispetto a incremental/0/1

→ Incremental/level0/volume2 rispetto al full

Cautele da tenere:

- correttezza della copia: si dovrebbe tenere il FS a riposo durante il backup, ma è raro nella pratica quindi va fatta attenzione ai dettagli relativi a lettu di file aperti o di strutture complesse come DB
- protezione dei dati: un backup contiene tutti i file del sistema, ma non c'è il sistema operativo a mediare l'accesso quindi in caso di riservatezza dei dati va difeso diversamente (in modo fisico o cifrando i dati)
- affidabilità dei supporti: con periodicità dipendente dalla criticità dei sistemi, ci si deve assicurare che i dati siano scritti correttamente e siano leggibili per tutta la durata prevista della copia: protezione da fattori tecnologici (graffi, smagnetizzazione, obsolescenza hw/sw:

c'è ancora sistema in grado di leggere i dati?) e da fattori ambientali (polvere, umidità, temperatura).

Vengono tipicamente effettuati su HD per sistemi di fascia medio/bassa, i nastri storicamente usati anche in questa fascia continuano a prevalere soprattutto in quella alta; avendo le soluzioni moderne e performanti un alto costo d'ingresso compensato da un basso costo marginale (per GB).

Riga di comando

Prontuario riga di comando

- ***whoami*** – indica il proprio username
- ***id*** – dà informazioni su identità e gruppo di appartenenza
- ***who*** – chi è collegato alla macchina
- ***shutdown*** *[-h|-r] now* – solo root, spegne la macchina (-h come comando halt) o la riavvia (-r come reboot), now indica quando

I caratteri digitati dall'utente dopo il prompt e terminati da fine linea costituiscono la command line (che termina con $ per utente, # per root).

I comandi possono essere:

- Keyword (es. ***for*** per lanciare ciclo for)
- Builtin (comandi interpretati direttamente dalla shell; ad esempio *cd* è builtin, non avrebbe senso lanciare un comando esterno e che poi qualcuno debba dire a me shell dove mi trovo)
- Comandi esterni (programma da file, lanciato dalla shell)
- Alias (stringa sostituita a un'altra)
- Funzioni (come in un linguaggio di programmazione, scrivo codice, do un nome alla funzione e poi lo invoco)

type permette di distinguere che tipo di comando eseguo, con opzione -a tutti i modi in cui la shell può trovare il comando. Per alterare ordine di default:

- \ davanti al comando previene solo l'espansione degli alias
- Keyword *builtin* previene l'espansione degli alias e l'uso di funzioni e invoca l'esecuzione del builtin specificato
- Keyword *command* utilizza un comando esterno anche se esiste una funzione con lo stesso nome
- comando ***unalias*** cancella un alias definito in precedenza

Gli alias permettono di dare un nome ad una command line (es. ***alias*** *miols='ls -l'*), queste associazioni vengono perse al termine della sessione (stessa cosa per ***unalias***). Gli alias definiti hanno la priorità rispetto ai builtin e i comandi omonimi

Per lanciare un eseguibile lo si può individuare col percorso completo

- assoluto */usr/local/bin/top*
- relativo *./mycommand*

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut
La shell usa var. d'ambiente PATH per eseguire la ricerca dei comandi nel file system. La sua struttura è quella di un elenco di directory separate da *:* --> *PATH=/bin:/usr/bin:/sbin*

Se ci sono eseguibili omonimi in directory diverse il sistema usa la prima istanza che trova nel path:

which ci dice quale versione si sta usando

Per ottenere configurazione persistente (es. per gli alias, ma anche per valori variabili ambiente, anche PATH) all'avvio della shell si usano i file di configurazione bash (vedi INVOCATION -verso l'inizio- e FILES -verso il fondo- di *man bash*): per configurazioni globali applicate a tutti gli utenti i file /etc/profile e /etc/bash.bashrc; per config. personale che scavalca i settaggi globali, i file .bash_profile .bash_login .profile .bashrc, si trovano nella home di quell'utente

Per controllare la documentazione sui comandi si usano:

- man pages: si invoca con ***man*** <*nomepagina*>, ogni applicazione installa pagine di manuale, raggruppate in sezioni
    1. User commands
    2. System calls
    3. Funzioni di libreria
    4. File speciali (/dev/*)
    5. Formati dei file, protocolli, relative strutture in C
    6. Giochi
    7. Varie: macro, header, filesystem
    8. Comandi di amministrazione solo per root

Opzioni utili:

- *man -a* <*comando*> : cerca in tutte le sezioni
- *man* <*sez*> <*comando*> : cerca in una sezione
- *man -k <keyword>:* cerca tutte le pagine attinenti alla parola chiave

- i builtin bash non hanno man page, si possono avere info sommarie con ***help*** **<*builtin*>** o in particolare nella man page bash(1)
- info files: si leggono con il comando *info*, a metà strada tra man page e ipertesto
- HOWTO: si trovano in /usr/[share]/doc/HOWTO, utili per la risoluzione di problemi pratici. Nello stesso path sotto /translations molti sono presenti anche in italiano
- On-line: molte risorse, in particolare tldp.org The Linux Documentation Project

Ogni comando accede ai caratteri che lo seguono nella command line quando viene invocato, i gruppi di caratteri separati da spazi costituiscono gli argomenti. La shell carica ARGV in memoria prima di generare il processo. Un argomento che inizia con '-' è un'opzione, non da elaborare di per sé ma usato per specificare varianti al comportamento del comando. Più opzioni si possono raggruppare in una sola stringa.

I comandi possono essere autocompletati con TAB, che con due pressioni mostra i possibili completamenti quando c'è ambiguità. ***history*** mostra l'elenco di tutti i comandi eseguiti in un terminale, si scorre con freccia-su che mostra editabili i precedenti, e può essere ricercata con CTRL-r (premuto ancora per andare a istanze meno recenti). Una volta trovato il match, si può

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut
lanciare con invio o renderlo editabile con freccia-dx/sx. ***script*** permette di catturare in un file un sessione di terminale, sia comandi che risultati; si termina con ***exit*** o CTRL-d. Con opzione -t include timestamps, il file prodotto può essere utilizzato come documentazione o come argomento per **scriptreplay** che visualizza senza rieseguire i comandi, ma funziona solo se il file include i timestamp

Intro a VIM

VIM è un'editor a tutto schermo, versione più amichevole dello storico VI. Ha un'interfaccia modale, il programma può trovarsi in uno dei seguenti stati, e si cambia tra uno e l'altro digitando alcuni caratteri:

- COMMAND
    - Il cursore è posizionato sul testo
    - La tastiera è utilizzabile solo per richiedere l'esecuzione di comandi, e non per introdurre testo
    - I caratteri digitati non vengono visualizzati
    - Si passa a INPUT con oOiIaACR
    - Si passa a DIRECTIVE con :/?
    - Si può richiedere lo **spostamento** del cursore con i comandi di movimento
        - **h** 1 a sinistra (come backspace)
        - **l** 1 a destra (come space)
        - **k** 1 linea sopra (stessa colonna)
        - **j** 1 linea sotto (stessa colonna)
        - Questi 4 Sostituibili con le frecce
        - ^ inizio riga
        - **$** fine riga
        - **gg** prima linea del testo
        - **G** ultima linea del testo
        - **#G** sulla linea numero #
    - E' anche possibile apportare **modifiche** (con comandi, in sostanza, di modifica)
        - **x** cancella il carattere su cui si trova il cursore
        - **dd** cancella la linea su cui si trova il cursore
        - **rX** rimpiazza il carattere sotto il cursore con X (modifica singolo carattere, non si passa ad INPUT)
        -
        - I successivi sono **comandi di modifica, causano l'ingresso in INPUT**
        - **i** inserimento nella posizione del cursore
        - **I** inserimento a inizio riga

- **a** append nella posizione del cursore
- **A** append a inizio riga
- **R** replace (sovrascrittura)
- **cw** change word, elimina la parola che inizia sotto il cursore ed edita
- **C** change, elimina fino a fine riga
- **o** linea vuota sotto al cursore
- **O** linea vuota sopra al cursore

- Ricerca e spostamenti possono essere usati come terminatore per alcuni comandi di modifica
  - d$ cancella dalla posizione corrente a fine riga
  - dG cancella dalla posizione corrente a fine file
  - c/ciao<RET> cancella dalla posizione corrente alla prima occorrenza della stringa ciao e si porta in insert mode
- Con il punto . si ripete l'ultimo comando impartito
- Precedendo un comando con un numero N, il comando verrà eseguito N volte consecutivamente (Es. 10x cancella 10 caratteri)
- ***u*** annulla l'ultima azione eseguita (**undo**)
- Copia e incolla:
  - La copia viene eseguita con buffer interni a vi, è possibile specificarne altri
  - ***yy*** copia la linea corrente nel buffer
  - ***"ayy*** copia la linea corrente nel buffer "a"
  - ***d*** esegue il cut
  - ***p*** incolla dopo la linea corrente
  - ***P*** incolla prima della linea corrente
  - Per copiare blocchi di linee si può
    - Usare marcatori: ***ma*** marca la posizione con il simbolo "a". ***y'a*** copia nel buffer tutto il testo dalla posizione marcata "a" in precedenza fino alla posizione corrente
    - Usando le ripetizioni: ci si posiziona sulla prima linea del blocco, per copiare 10 righe si digita ***10yy***
    - Usando la ricerca: ci si posiziona sulla prima linea del blocco, per copiare fino alla parola 'basta' (esclusa) si digita y/basta<RET>

- INPUT
  - Tutti i caratteri digitati vengono visualizzati ed inseriti nel testo
  - Si passa a COMMAND con <ESC>
- DIRECTIVE

- Ci si trova posizionati con il cursore nella linea direttive (l'ultima linea del video) e si possono richiedere tutti i comandi per il controllo del file
- Si passa a COMMAND con <RET>
- I comandi per **caricare/salvare/uscire** sono DIRECTIVE
  - :r <file> inserisce il contenuto di file al punto del cursore
  - :w scrive il file corrente
  - :q esce (:q! per uscire senza salvare)
  - ZZ scrive ed esce
- Digitando la barra / si entra in DIRECTIVE per **cercare stringhe** (es. ***/ciao<RET>)***
  - Con **n** si passa alla successiva occorrenza, con **N** alla precedente. Se si entra con **?** invece che con **/,** il verso di ricerca è verso l'inizio (e i significati di **n** e **N** si adeguano)
  - **/<RET>** e **?<RET>** ripetono ultima ricerca
  - ***:s/trova/sostituisci/*** cerca "trova" e lo sostituisce con "sostituisci"
  - *:**%s/trova/sostituisci/cgi*** cerca "trova" in ogni linea del file e lo sostituisce con "sostituisci"
    - Dopo aver chiesto conferma (**c**)
    - Anche più volte nella stessa linea (**g**)
    - Case insensitive (**i**)
    - ***%*** è una scorciatoia per ***1,$*** , in realtà il comando può essere invocato come ***:I,Fs****/trova/sostituisci/* per applicarlo tra le linee I ed F

Digitando ***q<lettera minuscola>*** inizia la registrazione della macro identificata dalla lettera indicata. Tutte le azioni compiute saranno registrate, finché non si preme nuovamente ***q*** per terminare la registrazione.

Digitando **@<lettera minuscola>** viene invocata la macro. (Ovviamente ripetibile con ***N@<lettera>***)

Composizione comandi e filtri

Tutti i programmi *nix che lavorano su stream di testo (filtri) hanno 3 stream standard:

- Fd 0 STDIN
- Fd 1 STDOUT
- Fd 2 STDERR

Bash può disconnettere gli stream predefiniti (chiudendoli nel figlio dopo la fork) e far trovare gli stessi fd aperti su un file diverso (aprendolo prima della exec).

- Ridirezione stdout:*ls* > *miofile*: scrive lo stdout di ls su miofile, troncandolo. se miofile non esiste viene creato. >> scrive in append
- Ridirezione stderr: 2> e 2>>

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut

- Confluenza: *ls > miofile 2>&1*: ridirige stderr dentro stdout, e poi stdout su file. L'ordine è importante
- Ridirezione stdin: *sort < miofile* riversa il contenuto di miofile su stdin di sort
- Piping: *ls | sort* : bash fa sì che ciò che ls produce su stdout venga riportato su stdin di sort

Filtri

La ridirezione è usata da comandi pensati per elaborare stream testo da stdin e produrre risultati in stdout, **i filtri**. I più importanti sono:

- ***cat****:* copia stdin su stdout. *tac* produce le righe in ordine inverso
- ***less****:* non è un vero filtro essendo l'output destinato al terminale ma è utile per l'uso interattivo. Posto al termine di una pipeline intercetta l'output e lo mostra riempiendo il terminale, opzioni
  - -h comandi disponibili
  - frecce – scorrimento
  - <N>g si porta alla riga num <g> default:1
  - G si porta al termine del file
  - /<pattern> cerca la riga successiva al cursore contenente <pattern>
  - ?<pattern> cerca la riga successiva al cursore contenente <pattern>
  - n ripete la ricerca fatta in precedenza
  - N ripete la ricerca fatta in precedenza, ma nel verso opposto
  - q esce da *less*
- **rev**: filtro che inverte l'ordine dei caratteri di ogni linea in stdin, su stdout. Usato di solito con *cut* per estrarre campi la cui posizione è nota dal fine linea
- ***head e tail****:* estraggono la parte iniziale/finale di un file. A default prendono 10 righe, con le opzioni
  - ***-c NUM*** i primi/ultimi NUM caratteri, con -/+NUM tutto il file eccetto/a partire da NUM caratteri
  - ***-n NUM*** prime/ultime NUM righe, con -/+ NUM tutto il file eccetto/a partire da NUM righe
  - ***-f*** per ***tail***, mantiene il file aperto e mostra in tempo reale le ultime righe. Utile per vedere output di processo che scrive su file. Per far ciò ignora EOF, quindi va usato –pid=PID per terminare quando termina processo PID. –retry fa riprovare tail finché non riesce ad aprire file quando produttore è in ritardo. -F == -f –retry
- ***cut****:* taglia parti di righe. Opzioni
  - -c ELENCO_POSIZIONI_CARATTERI intervallo per caratteri scelti (es. 1-8 per ogni riga)
  - Per i file a record (un record per riga), opzioni -d e -f per uno o più campi di ogni record
    - cut -d<CARATTERE_DELIMITATORE> -f<ELENCO_CAMPI>

- -s evita che vengano poste in output le righe senza delimitatore
- Es. *cat /etc/passwd | cut -f1 -d: -s* Estrae campo username da file passwd
- Es2. *cat /etc/passwd | cut -f5 -d: -s | cut -f2 -d' ' | cut -c1* Se nel campo note (campo 5) metto 'Nome Cognome' (separati da spazio), estrae dal cognome degli utenti (campo 2) l'iniziale (carattere 1)

- ***sort:*** ordina le linee di uno stream. L'ordine dei caratteri è stabilito dal locale scelto, con opzione LC_ALL=C è il valore dei byte che li codificano.
  - -u elimina entry multiple (equivale a *sort | uniq*)
  - -r reverse (ordinamento decrescente)
  - -R random (permutazione casuale righe)
  - -m merge di file già ordinato
  - -c controlla se il file è già ordinato

Criteri aggiuntivi all'ordinamento di default

- -b ignora spazi a inizio riga
- -d considera solo i caratteri alfanumerici e gli spazi
- -f ignora maiusc/minusc
- -n interpreta stringhe di numeri per valore numerico
- -h interpreta numeri leggibili come 2K, 1G…

Può inoltre cercare le chiavi di ordinamento in posizioni specifiche della riga, senza considerarla per intero.

- -t<SEP> imposta SEP come separatore, a default spazi
- -k<KEY> chiave di ordinamento, se usato più volte, ordina per la prima chiave, poi la seconda…
- KEY è nella forma semplificata F[.C] [,F[.C]] [OPTS] dove
  - F = numero di campo
  - C = posizione in caratteri nel campo
  - OPTS = una delle opzioni di ordinamento (n, f, d...)

Es. *sort -t. -k 1,1n -k 2,2n -k 3,3n -k 4,4n* Ordina un elenco di IP addess (byte1.b2.b3.b4)

- ***uniq****:* elimina i duplicati consecutivi. Con -c indica il numero di righe compattate, con -d mostra solo entry non singole
- ***wc:*** con -c conta i caratteri, con -l le linee e con -w le parole (stringhe separate da spazi)
- ***grep****:* esamina le righe in ingresso e produce in uscita quelle che matchano un pattern passato come argomento (espressione regolare, nel caso più semplice una sottostringa).

Più usata è la variante ***egrep*** che permette espressioni regolari "moderne". Corrisponde a grep -E

Segue il principio di "greediness" per selezionare i match:

- Nel caso una RE corrisponda a più di una sottostringa in una stringa, la RE matcha quella che inizia per prima.
- A partire da quel punto, se la RE matcha più di una sottostringa, si seleziona la più lunga

Nelle RE multilivello, le sottoespressioni selezionano sempre le sottostringhe più lunghe, tranne quando l'intera corrispondenza è la più lunga possibile. La priorità viene data a sottoespressioni che iniziano prima nella RE su quelle che iniziano dopo.

Opzioni principali di grep:

- Controllo matching:
  - -E usa extended RE (come egrep senza parametri)
  - -F disattiva le RE e usa il parametro come stringa letterale
  - -w/-x fa match solo con RE "whole word" o "whole line"
  - -i rende l'espressione case insensitive
- Controllo input
  - -r cerca ricorsivamente nei file di una cartella
  - -f FILE prende le RE da un FILE invece che come parametro
- Controllo output
  - -o restituisce solo le sottostringhe che corrispondono alla RE invece della riga che le contiene, separatamente una per riga di output
  - -v restituisce le linee che non contengono l'espressione
  - -l utile passando a grep più file su cui cercare: restituisce solo i nomi dei file in cui l'espressione è stata trovata
  - -n restituisce anche il numero della riga contenente l'espressione
  - -c restituisce solo il conteggio delle righe che contengono la RE
  - -q quiet, non scrive niente su stdout e esce con 0 appena trova match (anche se rileva errori)
  - --line-buffered disattiva il buffering

Espressioni regolari moderne (o estese): regexp o RE, documentazione in man page regex(7)

- RE = uno o più rami non vuoti separati da |
- Ramo = uno o più pezzi concatenati
- Pezzo = atomo eventualmente con moltiplicatore
- Atomo = uno di
  - ( ) contiene RE
  - [ ] contiene charset
  - ^ o $ o .

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut

- Backslash sequence
- Singolo carattere

Atomi speciali:

- . qualsiasi carattere
- ^ inizio linea
- $ fine linea
- Backslash sequence:
- \< - \> stringa vuota a inizio/fine parola
- \b stringa vuota a confine di parola
- \B stringa vuota a condizione che non sia confine di parola
- \w qualsiasi lettera o numero
- \W qualsiasi carattere non in \w

Moltiplicatori:

- {n,m} indica da n a m occorrenze dell'atomo che lo precede
- ? indica zero o una occorrenze dell'atomo che lo precede
- * zero o più occorrenze atomo precedente
- + una o più occorrenze atomo precedente

Esempi charset:

- [abc] UN qualsiasi carattere tra a,b o c
- [a-z] UN qualsiasi carattere tra a e z compresi
- [^dc] UN qualsiasi carattere che non sia né d né c

Charset basati su character class ***[:NOME_CLASSE:]*** dove NOME_CLASSE appartiene all'insieme definito in **wctype(3)** tipicamente:

alnum ***digit*** ***punct*** ***alpha*** ***graph*** ***space***

blank ***lower*** ***upper*** ***cntrl*** ***print*** ***xdigit*** o eventualmente nel locale attivo

Esempi RE

Essendo molti caratteri speciali delle RE anche caratteri speciali shell, è necessario proteggerli dall'espansione quando possibile, ponendo l'intera RE tra apici ‘’

- **egrep '^Nel.*vita\.$' miofile** ha come output tutte le righe di miofile che iniziano per **Nel** e finiscono per **vita.** (notare il punto quotato con \)
- **egrep '.es[^es]{3,5}e' miofile** ha come output tutte le righe che contengono in qualsiasi posizione la sequenza:
  - 1 carattere qualsiasi (il punto)
  - **es**

- una sequenza di 3-5 caratteri potenzialmente diversi uno dall'altro, a patto che ognuno sia diverso da e ed s
- **e**

Per altri esempi https://www.cyberciti.biz/faq/grep-regular-expressions/

tee legge stdin e lo scrive sia su stdout che sui file indicati.

- -a apre il file in append
- utile per tenere file intermedio *comando1 | tee FILE | comando2*
- con process substitution permette anche varianti più complesse in cui ad ogni stadio pipeline si elabora: *ls | tee >(grep foo | wc > foo.count) | tee >(grep bar | wc > bar.count) | grep baz | wc > baz.count*

diff mostra le differenze tra due file

paste unisce righe di posizione omologa in più file

join funziona in modo simile a paste, ma seleziona le righe non in base alla posizione: unisce quelle che iniziano con la stessa "chiave" (necessita di due file ordinati in modo identico sulla chiave selezionata)

sed e ***awk*** non sono semplici filtri avendo un vero e proprio linguaggio di programmazione, vediamo solo alcuni esempi pratici.

sed Stream EDitor, segue formato base ***sed -e 'comando'*** o ***sed -f 'script'***, non useremo script generici ma il solo comando di sostituzione:

sed 's/VECCHIO_PATTERN/NUOVO_VALORE/[modificatori]'

sostituisce in ogni riga il NUOVO_VALORE alla parte di testo coincidente con VECCHIO_PATTERN. Con sed -E i pattern sono *circa* quelli di ***egrep***, es. (inserisce la stringa "Linea:" all'inizio di ogni riga di passwd: ***cat /etc/passwd | sed 's/^/Linea:/'***). I modificatori del comando di sostituzione sono (si mettono dopo l'ultimo slash es. ***sed 's/^/Linea:/i'***

- i case insensitive
- g global (sostituisce tutte le occorrenze sulla riga)
- NUM sostituisce solo l'occorrenza NUM-esima

Opzioni su riga di comando:

- -i[SUFFIX] edita il file dato [backup con estensione SUFFIX se fornita]
- -u unbuffered

awk è un interprete per AWK (Turing-completo) che useremo solo come evoluzione di ***cut*** perché permette di considerare qualsiasi sequenza di caratteri come unico delimitatore. Nell'uso più comune permette di superare uno dei più evidenti limiti di ***cut*** in presenza di più delimitatori consecutivi: ad esempio, ***cut -f2 -d' '*** se ci sono due spazi dopo il primo campo considera il 2° spazio come 2° campo.

Esempi:

- stampa il secondo campo del file, purché sia separato dal primo da un numero qualunque di blanks ***cat personale | awk '{print $2}'***

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut

- in un file che riporta il risultato di un'operaz. come *[stringhe...] stat=esito [stringhe ...]* estrae tutti gli esiti: ***cat log | awk –F 'stat=' '{print $2}' | awk '{print $1}'*** dove *-F 'stat='* definisce il separatore dei campi
- a differenza di ***cut*** non ha il concetto di "-f 5-" (dal quinto campo in poi), ma: ***cat file | awk '{print substr($0, index($0,$5)) }'***

tr è utile per sostituire più rapidamente singoli caratteri, senza regex. Esempi:

- ***tr 'A-Z' 'a-z'*** trasforma maiuscole in minuscole
- ***tr ';:.!?' ','*** sostituisce qualsiasi occorrenza dei caratteri nel primo set con ,
- in generale, se il secondo set è più limitato del primo set, il suo ultimo carattere viene ripetuto quanto basta a generare la corrispondenza 1:1
  - ***tr ';:.!?' ',-'*** In questo caso quindi ; → , : → - . → - ! → - ? → -
- ***tr -d '\r'*** elimina ogni occorrenza del carriage return

xargs **<comando>** si aspetta su stdin un elenco di stringhe, da dare come argomento al comando indicato. Es. *find /usr/src -name '*.c' -size +100k -print | xargs cat* lancia *cat* con tutti i file risultanti che find mette su stdin

Shell e gestione processi

La shell, in particolare l'incarnazione bash che studiamo, nasce per automatizzare task, evitando di richiedere l'inserimento di comandi manuale. Si pone quindi non come ambiente per codice general purpose: lo scopo fondamentale è avviare processi di sistema, predisporre flussi di comunicazione tra questi, controllare come terminano e che diagnostica producono. Gli aspetti principali da considerare rispetto a un linguaggio come C o Java sono due: il primo è che gli elementi di base gestiti sono file e processi, (quindi è fondamentale pensare, quando si scrive o analizza una riga di comando, a quali processi verranno eseguiti e quali file sono coinvolti); il secondo è che bash è un linguaggio interpretato e non compilato, quindi la riga di comando passa attraverso un processo detto espansione, **che vedremo più avanti**, prima di essere eseguita.

In ambiente UNIX i processi sono processi pesanti, e possono essere creati usando

- fork, che crea una copia del processo corrente duplicando tutte le risorse (inclusi file) e condivide il codice
- exec, che sostituisce il codice del processo con quello caricato da un programma modificando la text table

Quando si lancia un programma quindi la prima cosa che accade è che viene duplicato il processo bash con tutte le sue risorse. Questo segue direttamente dal processo d'avvio del terminale: una volta che il kernel ha inizializzato i dispositivi HW e li ha esposti come device driver (/dev/tty* per i terminali virtuali collegati alla console, /dev/pts/* per pseudoterminali collegati a finestre grafiche), lancia *init* e smette di occuparsi dell'avvio, lasciando a *init* il compito di lanciare tutti gli altri processi. Il primo che parte è getty che fa una open su tty0 in lettura e una in scrittura (il processo ha quindi ora un fd0 da cui può leggere da tastiera e fd1 e 2 con cui può scrivere sullo schermo). Getty dopo la config del terminale fa exec e si trasforma in login, che attende inserimento user:pass avendo ereditato i fd da login. Una volta inseriti us:pw, il processo fa fork/exec e lancia bash, che a sua volta eredita i fd. Da questo momento in poi ogni comando aperto da bash segue la stessa filosofia, ereditando i fd dopo fork(nuova bash figlia)/exec(cambio codice).

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut
Per convenzione tutti i comandi *nix che operano su stream di testo (filtri) sono progettati per disporre di tre stream con cui comunicare con il resto del sistema: stdin su fd0, stdout su fd1, stderr su fd2.

A=40 mycommand | othercommand > outfile

L'interpretazione della linea di comando passa attraverso fasi ordinate:

- Espansione degli elementi della riga (**descritta in seguito,** la shell interpreta alcuni caratteri speciali, usati per gestire come eseguire) {*mycommand, othercommand}*
- Preparazione ridirezioni {| , > outfile}
- Assegnamento di variabili {A=40}
- Esecuzione comandi

Ridirezione

La ridirezione per un processo figlio di shell consiste nel disconnettere gli stream predefiniti dal terminale (chiudendoli dopo la fork, nel figlio) e far trovare gli stessi fd aperti su un file diverso (aprendoli prima della exec). Questo è possibile perché quando lancia un comando, la shell per prima cosa duplica sé stessa e quindi il figlio avrà gli stessi fd, facendo close dopo la fork e open di un file prima della exec il SO porrà quel file nel primo fd disponibile. Una volta fatta la exec il processo avrà gli fd già settati e aperti come da modello, e per lui non farà differenza dove puntino.

- L'stdout si ridirige con > myfile per aprire myfile da file pointer 0 (quindi troncandolo o creandolo), con >> per aprirlo da file pointer in fondo al file; stderr si ridirige con 2> o 2>>; stdin con command < myfile porta il contenuto di myfile su stdin di command.
- Particolare è la confluenza: ls > myfile 2>&1 prima ridirige stderr di ls su stdout (&1 indica proprio fd1, non un file di nome 1); poi stdout su myfile. Il risultato è che entrambi gli stream finiranno su myfile, ma l'ordine di ridirezione è importante, hanno priorità quelle fatte più a destra.
- Inoltre si può evitare l'uso di file temporanei per inviare dati su stdin di un comando, usando *command <<MARCATORE* (scrivendo poi più linee, e chiudendo l'input con una linea MARCATORE) o *command <<< "testo"* per un sola linea.
- Per ridirigere stream permanentemente si usa *exec [ridir]*, che ridirige gli stessi stream della shell quindi tutti i comandi successivi avranno quegli fd (2>/dev/null, che toglie stderr da terminale. interattivamente fa sparire anche prompt e echo ma è utile negli script).
- Con *exec* si possono anche creare nuovi fd: *exec 3< fin 4> fout 5<> frw* permette letture con <&3 su fin, scritture con >&4 su fout e frw sia in lettura che in scrittura; per chiudere un fd si usa *exec fd>&-*.

Pipe e subshell

La bash pipe è un array di due fd [0,1], con elemento 1 scrivibile e tutti ciò che viene scritto su 1 può essere letto da 0 (0 leggibile 1 scrivibile come standard). Quando c'è una pipeline (Es. *ls | sort*)

- bash fa due fork (due bash figlie, ognuna con 3 fd in lettura e scrittura, standard + i due pipe)
- la prima shell figlia chiama dup2(fd[1],1) per dirottare il suo stdout su fd[1] (quello della pipe) e viceversa la seconda chiama dup2(fd[0],0) per dirottare sul suo stdin quello della pipe

- infine la prima chiude fd[0] e la seconda fd[1] per evitare inconsistenze sulla pipe,
- poi ognuna fa exec del suo comando facendoli partire con gli fd già settati in modo da avere trasparenza senza sapere della pipe.

La pipe è quindi un mezzo di comunicazione tra processi fornito dal SO, il buffering offerto da questo permette di passare dati dal primo al secondo processo senza usare file temporanei. È il SO a occuparsi della sincronizzazione: se il secondo comando è lento a svuotare il buffer e questo si riempie, quando il primo chiama syscall *write* viene reso non schedulabile (sospeso) dal SO. Appena il secondo fa una syscall *read* e il buffer si libera, il SO va a richiamare il primo.

Caso particolare è quando il secondo elemento di una pipeline è un builtin o una funzione, in quanto non va caricato codice binario ma vanno interpretati da una shell: il secondo figlio non chiamerà *exec* e quindi resterà bash. Una pipeline con builtin crea quindi una subshell implicita, che è ancora un processo pesante (coincide il text segment avendo accesso alle stesse istruzioni, ma ha suo program counter), quindi ha i propri dati: variabili ereditate dal padre e poi modificate nel figlio non riflettono le modifiche sulle variabili nel padre (processi pesanti, copia per valore).

Si può forzare la creazione di subshell per eseguire sequenze di comandi nello stesso processo bash usando *(comando1 ; comando 2; ...)* dove ; equivale ad un accapo (eseguiti in ordine). Ciò è utile ad esempio quando nei processi figli modifico variabili d'ambiente: le modifiche si vedrebbero sul comportamento della shell padre, quindi preferisco lanciare shell figlie. Tutto ciò che viene dato su stdin della subshell è disponibile su stdin dei comandi e ciò che questi producono su stdout/err è prodotto dagli stream corrispondenti della subshell (Es. *producer | (step1 ; step2 ; step3) 2>/dev/null | consumer* dove producer legge stdin da tastiera non avendo <, scrive stdout su stdin della subshell che esegue i 3 comandi in ordine; ognuno dei 3 ha facoltà di scrivere sui propri std, e tutte le scritture su stderr della subshell finiscono su /dev/null. Infine l'stdout di subshell finisce su stdin di consumer).

Interazione coi processi e segnali

Ogni comando lanciato da shell diventa un processo, identificato da un Process ID o PID globale (in alcuni casi anche da un JobID, localmente alla shell che l'ha lanciato; anche se usando *ps* su altra shell è visibile con lo stesso PID). Un processo svolge le sue azioni a nome dell'utente che l'ha lanciato (processi root possono assumere altre identità ma perdono il potere di tornare indietro) e processi anche non lanciati da una stessa pipeline possono comunicare tra loro, mediante strumenti da preparare come socket, o in modo limitato ma semplice usando segnali.

I segnali sono eventi asincroni notificati dal kernel a un processo, generati dal kernel stesso (Es. SIGPIPE quando termina un processo in lettura su una pipe, così che l'altro possa smettere di scrivere) o da un altro processo; il cui contenuto informativo è limitato ad un numero. Non vengono ricevuti istantaneamente, il controllo avviene ogni volta che il processo rientra in user space (Es. dopo una syscall o quando schedulato da CPU): se tra un controllo e il successivo sono stati ricevuti più segnali diversi, vengono posti in uno stato "pending" senza notificare ricezioni multiple (viene settato il flag pending nel process descriptor nella process table, identificatore dei segnali ricevuti ma non gestiti) e vengono poi gestiti in modo non deterministico (come sviluppatore di script non si può assumere che vengano ricevuti nello stesso ordine in cui sono stati inviati). La gestione a livello di SO avviene mediante l'esecuzione automatica di handler, dirottando il flusso di esecuzione di un dato processo, in seguito alla rilevazione di segnale pending diretto a tale processo. L'esecuzione di un handler blocca segnali dello stesso tipo, ricezioni durante l'handler non causano esecuzioni annidate ma settano il flag.

Il comportamento di un processo alla ricezione di un segnale può essere terminare, ignorarlo, sospendersi (stato stop) o riprendersi da stop (cont), le disposition predefinite sono in *signal(7)*. La

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut
disposition per un processo può essere modificata: opzione di default; ignorare il segnale; eseguire un handler. Fanno eccezione KILL e STOP, che non possono essere bloccati, ignorati, o intercettati da un handler.

Gli handler possono essere definiti in bash usando il builtin ***trap*** *(trap [-lp] [[codice_da_eseguire] segnale ...]),* bash riconosce oltre ai segnali standard anche pseudosegnali come

- DEBUG (lanciato dalla shell prima di eseguire ogni comando. Es. utile per lanciare uno script ogni volta che un comando esegue),
- RETURN (lanciato da shell dopo ritorno da chiamata a funzione o dopo inclusione di file con *source*),
- ERR (lanciato dopo ogni comando che fallisce),
- EXIT (shell in uscita, da fine script, exit, o segnale di terminazione tranne KILL, con il quale l'handler verrebbe eliminato prima che possa essere eseguito).

Nota che i signal handler non vengono ereditati dai figli (mentre con ***export*** si possono passare loro variabili per riferimento, è invece necessario usare di nuovo *trap* nella subshell), l'esecuzione di un handler non blocca segnali dello stesso tipo e quando bash esegue un comando, il processo non è schedulato fino alla terminazione del child, quindi non vengono controllati i segnali finché il child non termina.

Per inviare un segnale a un processo si può usare ***kill*** *[options] <pid> [...],* dove pid <0 identificano l'intero process group; inoltre getty trasforma alcune combo di tasti su terminale in segnali inviati al processo che lo occupa:

- CTRL + Z per SIGSTP,
- CTRL + C per SIGINT,
- CTRL + \ per SIGQUIT.
- Inoltre CTRL+D produce carattere EOF che non è un segnale, CTRL + S stoppa scrolling congelando il terminale e CTRL + Q riavvia scrolling.

Esempio di chiusura pulita gerarchia processi in 9_job_control slide 12

Per esempi e approfondimenti sulla propagazione di segnali a child process:
https://linuxconfig.org/how-to-propagate-a-signal-to-child-processes-from-a-bash-script

Il comando ***sleep*** pone un timer per far dormire il processo per il numero di secondi indicato nel parametro. E' un comando esterno quindi valgono le solite regole: essendo eseguito in un processo figlio, inviare un segnale alla shell che lo genera non lo tocca (la shell padre non è schedulabile mentre esegue il figlio); inoltre anche sleep stesso è immune ai segnali (ricevuti ma non vengono processati) dato che non rientra in user space fino alla sua terminazione, poiché usa syscall per avere l'attesa, non fa busy wait.

E' possibile usare una sola shell per eseguire in contemporanea più comandi che non hanno necessità di accedere al terminale, lanciandoli in background postponendo ***&*** alla command line. La shell risponde con *[job_id]*, che identifica il job localmente a quella shell: per usarlo al posto di un pid si usa *%job_id*, mentre il PID del processo viene salvato nella variabile *$!*. In $! viene messo il pid dell'ultimo processo mandato in background, utile salvarselo se serve in seguito.

Il processo in background non riceve più lo stdin, in quanto appena si lancia *comando&* lo stdin ritorna alla shell che lo ha lanciato e si può inserire un altro comando; l'output resta però agganciato

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut
quindi il terminale potrebbe essere “sporcato” se il comando ha uscite, se non ci interessa l’output sarà opportuno ridirigerlo su /dev/null al lancio.

Se si lancia un comando senza & e si vuole rimediare, si può dare un segnale di STOP con CTRL+Z, ricevendo comunque un job_id; usandolo con ***bg %job_id*** si invia SIGCONT che riavvia il processo mettendolo in background.

Il builtin ***wait*** permette di bloccare l’esecuzione fino al completamento dei job in background. Di default attende il completamento di tutti i job. Utile quando si deve lanciare massa di processi in background ma dobbiamo aspettare che tutti abbiano concluso prima di tornare. (sarà utile per monitoraggio, avendo molte macchine che devono rispondere anche in modo lento. Se lanciamo in background in parallelo la funzione facendole monitorare 10 macchine. Faremo uno script dove lanciamo le verifiche, e per attendere tutti i risultati metteremo un wait). Esempio frequente: dovendo monitorare 40 macchine, abbiamo una funzione che interroga una macchina per sapere es. se ha abbastanza memoria, sappiamo che questo comando può metterci 3 secondi perché lato macchina interrogata è un quesito pesante. Non si fa un ciclo che interroga una macchina alla volta (impiego max 120 sec), si fanno 40 processi in background (monitora macchina $i), ognuno dei quali è leggero e ci mette 3 secondi ad interrogare la macchina di riferimento. Dopo si usa wait per attendere che siano terminati tutti i processi in background; l’output di ognuno dei processi finirebbe comunque su stdout, ma sarebbe più opportuno salvarlo. Usando un solo file ogni processo avrebbe la sua idea di fd, quindi anche scrivendo in append il file sarebbe sovrascritto da ogni processo essendo il file una struttura non concorrente; quindi si crea un file per ognuna delle interazioni, ridirigendo output per ogni processo su file $i.

watch prende come argomento altri comandi, li lancia periodicamente e aggiorna il suo output.

Se è necessario riportare in foreground un processo ricollegandolo così al terminale, si usa ***fg %job_id***. Il comando ***jobs*** mostra l’elenco dei job, cioè di tutti i processi avviati dalla shell corrente (l'elenco dei jobs è locale, quelli avviati in un'altra shell non vengono mostrati: usando ***ps*** invece avremo lo stesso risultato anche in'un altra shell, passandogli il pid del job), indicandone lo stato (attivo o stoppato). ***jobs -l*** mostra il pid dei job, nel caso non ce lo fossimo segnati.

Per i processi in background si possono usare i comandi ***nohup <command>*** per evitare che sia inviato SIGHUP al comando quando la shell chiude (ne causerebbe la terminazione, inoltre l'output viene rediretto -a default su *nohup.out-*) e ***nice <command>*** lancia command con niceness diversa da zero, modificandone la priorità. Sono comandi esterni, anche combinabili. Invece il builtin ***disown*** che agisce su PID/job_id lanciati in precedenza, rimuove un job dalla job table di shell (a default l’ultimo, con opzione -h offre anche immunità all’hangup).

Per gestire i processi è utile il comando ***ps*** (process status), che ha molte opzioni. Utile avere una cheat sheet. **https://www.golinuxcloud.com/ps-command-in-linux/**

top mostra i processi più attivi e altre statistiche.

Shell scripting

Il linguaggio di bash è interpretato, non compilato: il significato di molti caratteri è sintattico e non letterale: la riga di comando effettivamente eseguita risulta da un procedimento, detto espansione, che individua sottostringhe contrassegnate da caratteri speciali, sostituendole col risultato di una corrispondente elaborazione. L’espansione consiste in 12 passi in sequenza, alcuni possono essere saltati usando il quoting, ovvero proteggendo i meta-caratteri da non interpretare per mezzo di altri caratteri speciali (‘ “ e \).

(1) **Tokenizzazione**: La riga viene divisa in token usando come separatori i metacaratteri {SPACE TAB NEWLINE ; ( ) < > | &}. I token possono essere stringhe, parole chiave,

caratteri di ridirezione, carattere ":" (è il null command, comando che non fa niente). I caratteri speciali possono far comparire degli elementi che anche apparendo come normali valori, potrebbero avere un significato sintattico speciale per la shell, che potrebbe venir interpretato nei passi successivi. **Parti di riga quotate tra singoli apici ' saltano a 12) command lookup, parti tra doppi apici " saltano a 6) parameter expansion**

(2) **1st token = alias?**: La shell cerca il primo token nella lista degli alias. Se lo trova, lo espande e riparte col processing dal punto 1. Sono consentiti alias ricorsivi, bash non espande due volte uno stesso alias. Saltato con "

(3) **1st token = keyword?**: La shell controlla se il primo token è una parola chiave che inizia un comando composto, es. *if, while, function, {, (.* Se c'è bisogno crea una subshell per il comando composto e va a leggere il primo token. Saltato con "

(4) **Brace expansion**: **elementi tra {}** vengono espansi in una lista. Estensiva *{a,pippo,mamma}* o sequenza *{min..max[..incr.]},* ci sono anche molti altri tipi. Es. utile per creare un ciclo, espandendo un lungo elenco partendo dall'indicazione dei soli estremi. Saltato con "

(5) **Tilde expansion**: se c'è un token nella forma ***~username***, viene sostituito con la home directory dell'utente username (se username è vuoto, si usa l'utente corrente). Quindi è la shell a sostituire ~, non i comandi che usano percorsi. Saltato con "

(6) **Parameter expansion**: carattere **$** può marcare l'inizio di diverse espansioni (6, 7, 8), la più semplice PE è la sostituzione della stringa $NAME con il valore contenuto nella variabile NAME. Eseguito anche con "

(7) **Command substitution**: il token ***$(command)*** causa: creazione subshell, esecuzione comando, stdout di *command* viene posto sulla riga di comando al posto del token originale, a parte eventuali righe vuote alla fine. NON USARE COMMAND SUBSTITUTION quando comandi hanno grandi output perché viene messo tutto in memoria, MAI porre comandi che non terminano. Equivale a usare backtick, $() è però più leggibile e inoltre è annidabile $(c1($c2)) ma è difficile da usare. Eseguito anche con "

(8) **Arithmetic expansion**: il token ***((expr))*** causa la valutazione di *expr*, un'espressione aritmetica. **Se preceduto da $**, il risultato viene posto sulla riga di comando, altrimenti l'unico effetto è eventualmente sulle variabili usate. Eseguito anche con "

(9) **Process substitution**: con il token **<*(comando)*** o **>*(comando)***, viene eseguito *comando* in modo concorrente e asincrono rispetto al resto della riga. Serve quando abbiamo comandi che esigono di leggere da stdin o scrivere su stdout. Eseguito anche con "

- Quando abbiamo un *cmd_producer_su_stdout* e un *cmd_consumer_da_file* (che si aspetta un file come parametro), non possiamo metterli in pipeline: con process substituton *cmd_consumer <(cmd_producer_su_stdout)* il processo tra parentesi viene lanciato concorrentemente e la shell genera un nome di file (una named pipe) da fornire al primo (simmetrico *cmd_producer_su_file >(cmd_consumer_da_stdin)*)

(10) **Word splitting**: i risultati dei passi 6..9 sono esaminati e separati in *word* indipendenti. Separatore = qualsiasi carattere presente nella variabile IFS, a default IFS=<space><tab><newline>. Saltato con "

(11) **Pathname expansion**: ogni word viene esaminata e se contiene **uno tra *, ?, [** viene considerata un pattern e sostituita con tutti i file che concordano (quindi * non è un carattere speciale per i comandi, è la shell che sostituisce con tutti i file presenti e crea una command line dove tutti i file sono argomento del comando). Saltato con "

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut

(12) **Quote removal e command lookup**: vengono rimosse tutte le occorrenze di caratteri di quoting “usate” effettivamente (non protette da altri quoting, non generate dai passi 6..9), vengono impostati gli stream in caso di ridirezione, viene cercato il comando in ordine tra: funzioni, builtin, eseguibili in $PATH

Quoting

Il quoting consente di proteggere parametri che contengono i simboli ***[ ] ! * ? $ { } ( ) “ ‘ ` \ | > < ;*** che se non venissero protetti verrebbero espansi, impedendo l’uso del loro valore letterale

- \ backslash: protegge solo il carattere successivo, che non verrà interpretato come simbolo speciale
- ‘ apice: ogni carattere di una stringa racchiusa tra una coppia di apici viene protetto dall’espansione e trattato letteralmente, senza eccezioni
  - 1 → 12
- “ doppio apice: ogni carattere di una stringa racchiusa tra una coppia di virgolette viene protetto dall’espansione, con l’eccezione di $, ` backtick, \ backslash, e altri casi particolari
  - 1 → (6 7 8 9) → 12
  - vengono eseguiti solo i passi 1) Tokenizzazione, 6) Parameter Exp, 7) Command Sub, 8) Arithm. Sub, 9) Process Sub, 12) Quote Removal e Command Lookup
  - vengono saltati 2) Check token alias, 3) Check token keyword, 4) Brace Exp, 5) Tilde Exp e poi 10) Word Split, 11) Path Exp

I simboli stessi di quoting vanno protetti dall’espansione se vanno usati per il loro valore letterale. È possibile separare frammenti protetti in modo diverso, verranno semplicemente concatenati dopo l’espansione e la quote removal:

- es. ***“protetto da virgolette”*’o da apici’*** sarà espanso come SINGOLO token (la separazione in pezzi diversi viene fatta dal word splitting, che avviene prima della quoting removal. Quindi gli spazi del contenuto tra " " non vengono considerati, come quelli tra ' '. Tutto il pezzo, spazi inclusi, sono considerati come singolo token) di valore

protetto da virgolette*o da apici

Il comando ***echo*** stampa a video i caratteri che seguono, è utile per visualizzare il valore di una variabile (***echo $PATH*** visualizza il contenuto della variabile PATH*)* o di una pathname expansion (***echo **** visualizza tutti i nomi di file nella directory corrente) sfruttando l’espansione di bash (*echo non sa cosa sia una variabile o un pathname*).

Pathname expansion

La pathname expansion è molto utile per gestire file e directory, ed essendo uno degli ultimi passi svolti da bash opera su stringhe che potrebbero essere generate ai passi precedenti. Viene saltata con quoting a doppi apici “”. I pattern tipici sono

- * rappresenta una qualunque stringa di zero o più caratteri
- ? rappresenta un qualunque carattere singolo
- [SET] rappresenta un carattere appartenente a SET:
  - [afhOV] elenco

- [a-k] intervallo, il cui ordine dipende dal locale (più intervalli uniti con virgola es. [a-d,0-5])
- [!a] [^A-Z] per negare il contenuto del SET (es. tranne a, tranne da A a Z)
- [[:alnum:]] classe di caratteri come per ***egrep***
- per includere i caratteri – o], metterli come primi carattere

Bash cerca se può sostituire la stringa che contiene uno di questi pattern con qualcosa che corrisponde, dal path in cui si trova, e sostituisce con file che corrispondono, in ordine alfabetico. Se non esistono file che corrispondono, il pattern resta inalterato sulla linea di comando. Vedi sez. *pathname expansion* di ***man bash(1)***.

Esempi pathname expansion:

- echo * tutti i file del direttorio corrente
- echo [a-p,1-7]*[cfd]? File con nomi che iniziano per caratt. compreso tra a e p o tra 1 e 7, se il penultimo caratt. è c, f oppure d
- echo * fa echo del carattere *, non visto come wildcard da bash ma con valore letterale (quoting con backslash)
- echo *[!*\?]* elenca tutti i file del direttorio corrente che hanno almeno un caratt. Diverso dalle wildcard * e ?
- echo /*/*/* elenca tutti i file dei direttori di secondo livello a partire dalla root (tutti i file che stanno in una directory seguita da qualsiasi carattere che sta in una directory seguita da qualsiasi carattere che sta in una qualsiasi directory seguita da qualsiasi carattere)

È possibile abilitare comportamenti più complessi (Moltiplicatori, inversione del matching) attivando l'opzione extglob con ***shopt.***

Brace expansion

Un meccanismo di espansione per generare sequenze di stringhe secondo un pattern, con la stessa sintassi della pathname expansion, ma le stringhe sono generate indipendentemente dal fatto che esistano o meno file che rispettano il pattern (al contrario della path exp). Avviene presto nell'espansione della riga, non si possono usare variabili che contengono valori che la shell espande dopo. Viene saltata con quoting a doppi apici "".

Sintassi [PRE]{LISTA}[POST] oppure [PRE]{SEQUENZA}[POST]

Esempio lista:	a{d,c,b}e	espanso dalla shell in ade ace abe
Esempi sequenza:	file{9..13..2}.c	espanso in file9.c file11.c file13.c
	doc{009..11}	espanso in doc009 doc010 doc011 (shell offre zero-padding)
	{a..j..3}	a d g j (solo per singoli carattere alfabetico)
	{a..c}{1,3}	a1 a3 b1 b3 c1 c3 (tutte le combo, "prod. cartesiano")
	p{1{a,b},2,3{b,d}}	p1a p1b p2 p3b p3d (nesting)

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut

Variabili

Le variabili sono offerte da shell per memorizzare stringhe di testo sotto dato nome. Si creano o modificano semplicemente con pippo=valore (senza spazi, altrimenti tokenizzazione impedisce assegnamento). La param. Exp sostituisce $NOME con il valore della variabile NOME (se NOME è composto o ambiguo, si protegge con {} quindi ${NOME}).

Esempi di interazione con quoting:

- ls *** lista tutti i nomi dei file che contengono il carattere * in qualunque posizione
- echo “$A” stampa esattamente il contenuto della variabile A (inibizione dei passi successivi alla param.exp)
- echo ‘$A’ stampa esattamente $A (inibisce quasi tutti i passi, inclusa param. Exp)
- echo “’$A’” Chi vince? La prima cosa che incontra la shell è l'apice doppio, e va a cercarne la coppia. Gli apici effettivamente usati per proteggere delle sequenze vengono rimossi in 12) Quote rem, quindi l’apice singolo mantiene il suo significato letterale. Utile per usare due protezioni diverse quando si deve passare attraverso due espansioni, in particolare sarà utile per proteggere cose da passare alla shell remota delle VM.
- echo $(ls) ; echo “$(ls)” La differenza è che nel primo viene fatto 10)Word split, nel secondo no quindi tutti gli accapo restano tali e non cambiano in spazi

Le variabili d’ambiente sono usate per mantenere dati riguardo il sistema o le preferenze utente, utili a tutti i comandi. Per evitare di passarle a ogni comando, la shell dispone dell’esportazione (***export pippo***) con cui variabili standard diventano d’ambiente (ereditabili dai processi figli), altrimenti resterebbero confinate nella shell stessa. L’ambiente (environment) corrente si visualizza con builtin ***set. unset*** si può usare per eliminare una variabile

Un assegnamento prima di un comando modifica solo per tale esecuzione l’ambiente (VAR=x *command*), comando ***env*** permette un controllo più preciso.

Variabili notevoli: (man bash → Shell variables)

- Settate da bash
  - $ pid della shell capostipite (per vederne il valore, $$)
  - HOSTNAME nome dell’host
  - RANDOM num. Casuale tra 0 e 32767 (as es. utile per assegnare carico di lavoro tra vari computer)
- Usate da bash for F in "$(cat /etc/passwd | cut -f5 -d:)" ; do echo $F | tr 'a-z' 'A-Z'; done
  - PS0..PS4 prompt in diversi contesti
  - HOME home dir dell’utente

Le variabili posizionali sono usate da ogni script per accedere agli argomenti sulla riga di comando (come argv) usando le variabili che hanno per nome un numero. ***$0*** è il nome del comando, se il numero è >9 va usato ***${NUM}***.

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut
Se non si sa quanti parametri sono stati passati (es. ciclo senza sapere il totale) si può usare $* oppure $@, con differenza nel quoting:

- "$*" viene espanso in "$1 $2 $3"
- "$@" viene espanso in "$1" "$2" quindi mantiene la protezione per ognuno dei contenuti

Il builtin ***read*** legge stringhe da stdin e le assegna a variabili. L'input viene tokenizzato usando IFS (di default qualsiasi spaziatore), se ci sono più token che variabili, quelli in eccesso finiscono tutti nell'ultima variabile specificata, come unica stringa separatori inclusi.

(Facendo ***IFS=: read A B C...*** cambio IFS in : , ma solo per read, spazio non è più tokenizzatore per read. Se cambio IFS nella shell generale si scombussola tutto)

Opzioni utili	-p PROMPT	stampa PROMPT prima di leggere input
	-u FD	legge da FD invece che da stdin
	-a ARRAY	assegna i token a elementi di ARRAY

Nella realizzazione degli script è utile tenere a mente i sottoprocessi generati in pipe:

- manipolare variabili nei processi figli senza perdere i risultati prima di poterli usare → si può usare subshell con ***echo ciao | (read A ; echo $A)***
- necessità di usare read per acquisire dati interattivamente dall'utente in un processo figlio che ha stdin alimentato da una pipe anziché da un terminale → creare file descriptor per il terminale con ***exec 3<$(tty) ; exec ciao | (read -u 3 A ; echo $A)*** (tty restituisce file speciale che descrive il terminale connesso a stdin)

shift sposta gli argomenti della riga di comando (es. $1 diventa il contenuto di $2, viene buttato via il contenuto di $1), $0 rimane il nome dello scripting.

select è builtin che può essere utile per automatizzare la selezionare di alternative , esempio

```
directorylist="Finished $(for i in /*;do [ -d "$i" ] && echo $i; done)"
PS3='Directory to process? ' # Set a useful select prompt
until [ "$directory" == "Finished" ]; do
        printf "%b" "\a\n\nSelect a directory to process:\n" >&2
        select directory in $directorylist; do
                # User types a number which is stored in $REPLY, but select
                # returns the value of the entry
                if [ "$directory" = "Finished" ]; then
                        echo "Finished processing directories."
                        break
                elif [ -n "$directory" ]; then
                        echo "You chose number $REPLY, processing $directory..."
                        # Do something here
                        break
                else
                        echo "Invalid selection!"
```

```
                fi # end of handle user's selection
        done # end of select a directory
done # end of while not finished
```

getopts è builtin utile per realizzare script che supportino sintassi con opzioni precedute da trattino. Sintassi è *getopts optstring NAME [arg]* dove

- *optstring* definisce i caratteri da riconoscere come opzioni, se un carattere è seguito da : significa che si attende un parametro per quell'opzione
- *NAME* è il nome di variabile in cui collocare il parametro correntemente analizzato

Ad ogni invocazione *getopts* esamina una variabile posizionale, assegnando il suo indice ad OPTIND ed il suo contenuto a NAME. Se un'opzione richiede argomento nella successiva variabile posizionale, questo viene letto (incrementando OPTIND) ed assegnato ad OPTARG. Esempio:

```
#!/usr/bin/env bash
aflag=
bflag=
while getopts 'ab:' OPTION ; do
        case $OPTION in
                a)      aflag=1 ;;
                b)      bflag=1
                        bval="$OPTARG" ;;
                ?)      printf "Usage: %s: [-a] [-b value] args\n" $(basename $0) >&2
                        exit 2 ;;
        esac
done
shift $(($OPTIND – 1))      # getopts cycles over $*, doesn't shift it
if [ "$aflag" ] ; then
        printf "Option -a specified\n"
fi
if [ "$bflag" ] ; then
        printf 'Option -b "%s" specified\n' "$bval"
fi
printf "Remaining arguments are: %s\n" "$*"
```

Array

Bash supporta array monodimensionali, si dichiarano con *declare -a MYVECTOR*

- si assegnano più valori mettendoli tra parentesi
    - MYVECTOR=(un elenco di elementi)

echo ${MYVECTOR[2]} #output→ di

- si parsa come read manipolando IFS
    - STRING="fai.il.parsing.come.read"

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut

IFS='.' MYVECTOR=($STRINGA) #→ valori sempre tra parentesi

echo ${MYVECTOR[2]} #output → parsing

~~Accenno agli array:~~ gli array in bash usano indici non necessariamente consecutivi. Usarne uno può essere utile per gestire multipli processi, indicizzandoli con il loro PID.

echo ${!PID[@]} mostra tutti gli indici validi per l'array di nome PID

echo ${PID[@]} mostra tutti i valori contenuti nell'array di nome PID

Quindi ! chiede alla shell gli indici dell'array, non i valori; viceversa senza ! mostra i valori contenuti. La protezione dell'espansione della chiocciola è coerente alle variabili posizionali:

for i in "${A[@]}" ; do echo $i ; done espande ogni valore dell'array A con quoting tra doppi apici per ognuno di essi

Es. "1" "2" "3"

for i in "${A[*]}" ; do echo $i ; done espande l'intera sequenza dei valori dell'array A con quoting tra doppi apici

Es. "1 2 3"

È possibile ottenere l'espansione di variabili al nome a cui sono assegnate per avere riferimento indiretto ai valori corrispondenti a quel nome: (indirezione)

- CHIAVE=PIPPO

PIPPO=VALORE

echo ${!CHIAVE} #output→ VALORE

In Bash 4+ si hanno veri array associativi (indice può essere una stringa, non solo un numero)

- declare -A ASAR #uso di -A per indicare array associativo, non -a

ASAR[chiaveuno]=valoreuno

echo ${ASAR[chiaveuno]} #output → valoreuno

KEY=chiaveuno

echo ${ASAR[KEY]} #output → valoreuno

Arithmetic Expansion

Tutto viene trattato come stringa in bash, salvo contesti particolari in cui può svolgere operazioni aritmetiche. La shell ha il limite di poter lavorare solo con numeri interi e non c'è modo di accorgersi del rollover (se si ha overflow sul signed int dell'architettura); ***bc*** è calcolatore a riga di comando che può essere usato per avere risultati a precisione arbitraria in script.

Si ha valutazione aritmetica in due casi

- ***declare -i N*** per dichiarare variabili come intere:
  - La shell non ha concetto di tipi, le variabili sono stringhe. SOLO quando si dichiara una var come intera e viene assegnata una stringa che può essere interpretata come intero, questa viene così assegnata.
  - Con **-p** print mostra tipo e valore del simbolo (output – nessun tipo)
- Builtin ***let***, o equivalente comando composto ***(( ))***

- ***let N++*** l'espressione viene valutata, ha effetti sulle variabili, ritorna true se risultato non è nullo, produce errori su stder ma nulla su stdout
- **(())** si comporta come "" per proteggere elementi dell'expr, riconosce variabili anche se non sono dichiarate intere e senza bisogno di prefisso $ per espansione (ammessa, ma se aggiunta non vale il seguente) e interpretandole come zero se non definite
- Variabili mai definite non danno errore in contesto aritmetico, vengono valutate come fossero 0. Gli operatori sono quelli del C
  - id++ ++id post/pre incremento (decr. Con –) la variabile viene espansa prima/dopo l'incremento
  - + - * / somma sottraz. prodotto divis.
  - ** % elevamento a potenza, modulo
  - ! ~ & | ^ NOT NOT AND OR XOR bit a bit
  - << >> shift binario a sx/dx
  - = *= /= += -= <<= >>= &= ^= |= assegnamenti
  - <= >= < > == != confronto
  - && || AND OR logico
  - expr?expr2:expr3 restituisce risultato di expr2 o expr3 se expr è true/false
- I numeri si possono esprimere tra base 2 e base 64 con prefisso ***B#num***→ base B
  - default 10, prefisso 0 ottale, prefisso 0x esadecimale
  - cifre utilizzabili 0..9a..z..Z@_ (servono 64 cifre per base64)

Controllo di flusso

Sequenze

Con ***( )*** Si possono raggruppare comandi per trattarli come uno solo aprendo una subshell, per creare una sequenza senza aprire una subshell (mantenere esecuzione nello spazio di memoria della shell principale), si usa ***{ }***. Ogni comando della sequenza deve terminare con **;** o **newline**

In entrambi i casi l'exit code della sequenza è quello dell'ultimo comando eseguito

$'\x0a' produce ASCII newline (vedi ***man bash → $'string'***)

Funzioni

Le funzioni sono sequenze con un nome **function NOME() { SEQUENZA ; }** (il punto e virgola serve per ricordare l'accapo, se non si mettesse il ; l'ultima graffa dovrebbe andare accapo). Possono ricevere parametri, utilizzabili all'interno allo stesso modo dei parametri posizionali degli script $1,$2… ma appunto per questo i valori passati allo script non possono essere visti internamente alla funzione, solo $0 resta impostato al nome dello script (si possono passare per valore all'interno: se si passa $1 del main come arg funzione, copia il valore $1 del main). Il nome della funzione viene posto in $FUNCNAME.

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut
Se invocate semplicemente, sono eseguite nel contesto del chiamante. Al contrario di C e assembly, non c'è creazione di contesto diverso: non viene creato altro processo, è lo stesso che devia e va ad eseguire prima di riprendere da dov'è arrivato.

Le variabili sono “globali”: una variabile A modificata in una funzione cambia valore nello spazio di memoria generale, si possono dichiarare variabili locali con **local**. Per le invocazioni in pipeline la funzione verrà eseguita dalla shell figlia creata in automatico: scrivendo *cat miofile* | *function* | *sort* si ha creazione automatica di 3 shell; **in questo caso non è più vero** che lo spazio di memoria è lo stesso del chiamante! La seconda fork non fa exec , è una subshell che interpreta direttamente la funzione (shell diversa dalla shell che prepara la pipeline).

Valutazione delle condizioni

I comandi di controllo di flusso non fanno valutazione logica delle espressioni, decidono il percorso in base all’exit code di un processo (0==true, altri valori==false). L’exit code di un processo si trova nella variabile speciale **$?** settata dalla shell al termine del processo. E’ comodo ma abbiamo interpreti di espressioni logiche per quando serve, sia come builtin che come comandi esterni:

- Builtin **test, [ ], [[ ]]**
- Comandi **test, [ ]**

Essendo i builtin eseguiti prima di cercare i comandi equivalenti faremo riferimento a quelli, ma è utile ricordare che se serve portabilità degli script, i comandi esterni sono standard su molti sistemi indipendentemente dalla shell usata, tipicamente hanno sintassi identiche.

test e [sono lo stesso builtin (solo, [richiede] come ultimo parametro) e seguiti da un’espressione la valutano e ritornano 0 o 1 in base al risultato (0 TRUE, 1 FALSE)

- Test unari, un solo parametro: **test OP ARG**
  - su stringhe
    - -z true se stringa è vuota
    - -n true se string è non vuota
  - su file
    - -e true se file esiste
    - -f true se è file regolare
    - -d true se è directory
    - -s true se file non è vuoto
  - altri 20 su **help test** (è builtin, quindi o help o **man bash** e ricerca)
- Test binari, due parametri: **test ARG1 OP ARG2**
  - confronto lessicale (alfabetico) tra stringhe
    - =, !=, <, > per qualche motivo < e > non sono per verifiche numeriche tra stringhe ma per verifica lessicale (alfabetico)
  - confronto numerico tra stringhe
    - -eq, -ne equal, not equal

        - -lt, -le less than, less or equal
        - -gt, -ge greater than, greater or equal
    - confronto tra file
        - -nt newer than
        - -ot older than

[[ ]] è builtin presente nelle versioni più recenti di bash, in più di **test** / **[** offre

- op binari == e != matchano param. Sinistro con pattern espresso dal parametro di destra con stessa sintassi di pathname expansion ([[]] equivale a quoting, i metacaratteri vengono protetti)
    - Es. ***[["ciao" == c?a[l-z]]] → true***
- op binario =~ matcha param. Sinistro con una regular expression specificata dal param, destro
    - Es. ***[["ciao" =~ ^c.{2}o$]] → true***

  Simboli speciali uguali hanno significati diversi nei due casi (sintassi pathname exp != sintassi regex)

E' possibile combinare controlli elementari:

- con test / [ ]
    - operatori AND, OR, NOT sono rispettivamente -a, -o, !
    - si possono fare raggruppamenti con ( ), vanno protette con backslash , si usano \(\)
- con [[ ]]
    - operatori AND, OR, NOT sono rispettivamente &&, ||, !
    - si possono fare raggruppamenti con (), protette già dalle [[ ]], si usano così come sono

Gli operatori **&&, ||, !** si possono usare sulla riga di comando per combinare logicamente gli exit code di qualsiasi processo. Sia in questo caso che quando usati dentro [[ ]], valutazione shortcut: se il primo risultato è sufficiente a sapere il risultato, il secondo non viene valutato. Nel caso dei processi è come un if, quindi con AND se il primo comando torna false il secondo non viene eseguito, viceversa con OR solo se il primo torna false il secondo viene eseguito. Utile per avere check:

cd "$MYDIR" || echo ("Errore") ; exit(1)	il ramo destro viene eseguito solo se fallisce il sinistro, con messaggio e bloccante (uscita)

if è utile tutte le volte che voglio esecuzione condizionata: la keyword **if** vuole un COMANDO come argomento (esterno eseguito da shell figlia, builtin, funzione… ecc) e solo se il comando ritorna 0 viene considerato come TRUE, qualsiasi altro ritorno viene visto come FALSE. **elif** per seconde verifiche, **else** per caso alternativo globale (questi due vanno racchiusi tra [ ]), **fi** chiude la sequenza. In una pipeline (o anche funzione senza nome con molti comandi con ; ; ;) l'exit code è quello dell'ultimo comando eseguito. I COMANDI possono essere composti (sequenze, subshell, combinazioni logiche...)

if COMANDO1

then

comandi eseguiti se COMANDO1 ritorna true

[elif COMANDO 2

then

comandi eseguiti se COMANDO2 ritorna true]

[else

comandi eseguiti se nessun ritorno true]

fi

case permette di esprimere condizioni multiple in modo più leggibile che in una catena di **elif.** Si usa la stessa sintassi della path exp per i casi, (ma non viene davvero eseguita espansione del nome del case, ovviamente). Se nella variabile c'è valore corrispondente alla sintassi, si entra nel caso. Vedi esempio

case "$variabile" in

nome1) echo vale nome1 ;;

nome?) echo vale nome2, nomea, nomez ;;

nome*) echo vale nome11, nome, nomepippo ;;

[1-9]nome) echo vale 1nome, 2nome, …, 9nome ;;

*) echo non cade in nessuna delle precedenti ;;

esac

for itera su una lista di elementi, sintassi ***for*** NAME ***[in*** WORDS … ***]*** ; ***do*** COMMANDS; ***done***

Casi d'uso:

- WORDS = pattern di pathname expansion
  - itera direttamente sui nomi di file prodotti dall'espansione

for F in /tmp/*.bak ; do rm -f "$F" ; done

valuta condizione for: prima viene espansa la sequenza, al posto di /tmp/*.bak appaiono tutti i file corrispondenti all'espressione. Solo dopo la shell sa quante iterazioni dovrà fare. A questo punto viene eseguito il codice tra **do** e **done** aggiornando il valore di F per ogni iterazione.

La shell evita espansione caratteri speciali se sono nei nomi di file, ma per ogni iterazione poi ci sarà una nuova espansione! Es. se assegno filename "Marco Prandini" (con spazio) a F, viene preso come un solo elemento ma nell'iterazone, se non metto quoting intorno a $F, cambia il significato: viene espanso in ***rm -f Marco Prandini*** e il comando vede due parametri file separati, provando a eliminarli ma non esistono. Occhio quindi ad usare "$@" che protegge ognuno dei parametri con lo stesso quoting

- WORDS = parametri della command line
  - ***for PAR in "$@" ; do echo "$PAR" ; done***
- WORDS = *command substitution*
  - ***for USER in $(cat /etc/passwd | cut -f1 -d:)***

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut
vedi pattern comune per processing righe

- WORDS = brace expansion
  - ***for ITEM in item_{a..z}***
- Si possono impiegare sequenze numeriche usando NON MOSTRATO
  - *for BACKWARDSTENTHS in $(seq 1 -0.1 0)* dove (start incremento end)
- Nelle versioni recenti di bash sintassi C-like NON MOSTRATO
  - ***for ((i=0, j=0 ; i+j < 10 ; i++, j+=2))***
  - Espressioni di inizializzazione, test di terminazione (espressione logica uguale a true o aritmetica che dà 0. Se c'è più di un test vengono eseguiti tutti ma solo l'ultimo determina se il ciclo prosegue), espressioni eseguite ad ogni iterazione

while permette cicli indefiniti basandosi su un comando da eseguire per verificare se l'exit code è TRUE (solo 0, tutto il resto FALSE), (**until** itera quando si ha FALSE) sintassi

while COMANDO (oppure ***until*** COMANDO) ; ***do*** LISTA ;COMANDI; ITERATI; **done**

Naturalmente COMANDO può essere composto (sequenze, subshell, combinazioni logiche…)

Pattern comune usato per fare processing di dati:

- **cat /etc/passwd | while read riga ; do echo $((counter++)) $riga ; done**
  - **cat** legge tutto e mette su stdout, pipeline porta stdout su stdin del processo dopo, figlio (shell figlia che fa il while) **read** mangia una riga di input e restituisce true; **while** ha ricevuto true e fa iterazione: **echo** di valore di counter incrementato dopo averlo espanso (espansione non da errore perché parte da 0 essendo counter non definito) seguito dal valore di riga.
  - Tenere a mente importanza di aprire subshell esplicitamente per il while in pipe, così da garantire corretto spazio di memoria per eventuali handler e variabili
  - Se necessario input interattivo dentro al ciclo, il secondo read va alimentato da terminale es. *T=$(ps h $$ | awk '{ print $2 }') ; cat /etc/passwd | while read U ; do { read VAR < /dev/$T ; echo $VAR } ; done* oppure esempio messo nella desc. del builtin *read*
- Versione con for: **for riga in $(cat /etc/passwd) ; do echo $((counter++)) $riga ; done**
  - Il for è vantaggioso perché non c'è pipeline, quindi tutto quello che avviene nelle iterazioni è nella stessa shell che lo lancia: si vede che se si rilancia il comando due volte, parte con il counter dallo stesso valore, la variabile è dello stesso ambiente.
  - Se l'elemento informativo è la riga evito di usare un for, con while possiamo sfruttare read che ha il concetto di riga. Fintanto che nelle righe ci sono solo spazi non c'è problema ad usare for, ma se ci sono spazi e accapo invece è un problema perché perdiamo la differenza tra i due separatori.

break [N] esce da un ciclo for, while o until (se specificato, esce da N cicli annidati)

continue [N] salta alla successiva iterazione di un ciclo (se specificato, riparte risalendo di N cicli annidati)

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut

Accorgimenti utili

source : utile per condividere parametri tra script correlati tra loro e per creare librerie di funzioni importabili. Può essere usato per eseguire uno script nel contesto di un altro (inclusa riga di comando interattiva). Es. supponiamo che lo script common.sh contenga assegnamenti di valori a variabili, definizioni di funzioni e alias. Dopo l'esecuzione di ***source common.sh*** le variabili, funzioni e alias saranno definite anche nello script chiamante.

su è utile da root (da altri utenti chiederebbe password, non pratico da scriptare) per eseguire comandi con altre identità, ma non è ***exec***: non si mettono comandi a seguire, apre una shell e finché non si esce da quella non prosegue nel contesto chiamante. Sintassi ***su -c "COMANDO" - UTENTE***.

sudo COMANDO è utile ad usare comandi come root, ma richiede la configurazione di sudoers (possibilmente con NOPASSWD)

- Sintassi per /etc/sudoers: https://toroid.org/sudoers-syntax
  - formato delle direttive: User Host = (Runas) Tag: Command
    1. "User puo eseguire Command coi privilegi di Runas su Host"
       1. %sudo ALL=(ALL:ALL) ALL riga base tipica, per gruppo sudo
       2. usermod -aG sudo las Aggiunge "las" al gruppo sudo, efficace solo dal successivo login
    2. Runas si può specificare come utente[:gruppo] ; se manca, è implicito root (e nessun altro)

date ha molte opzioni:

- con -s può impostare orologio di sistema
- può restituire l'orologio in formati diversi
  - ***date +FORMAT*** permette di selezionare cosa e come visualizzare
  - ***FORMAT*** è una stringa in cui vengono interpretate sequenze speciali contrassegnate dal carattere ***%*** (vedi ***man date***) es. ***date +"%Y%m%d %H:%M:%S"*** AnnoMeseGiorni Ora:Minuto:Secondo; ***date +"%A %e %B"*** Giornosettimana Giorno Mese (locale)

%s numero secondi da epoca Unix (1/1/70), (***date +s%***) ; con %s%n appende i nanosecondi

  - con -d si può fornire un timestamp da usare al posto del tempo corrente (es. convertire timestamp tra diversi formati: ***N=$(date -d '2020-05-15 10:01' +%s)*** dà ora di evento interessante convertito in epoch, ***((N+=1800))*** modifica tempo, ***date -d "@$N"*** stampa in modo leggibile: ***@*** operatore per cambiare da epoch a timestamp

mktemp è utile per creare file temporanei, restituisce il nome del file creato (default formato /tmp/tmp.XXXXXXXXXX. Opzioni

- -d crea una directory
- -p DIR crea all'interno di DIR invece che in /tmp

basename rimuove path ed eventualmente suffisso da un nome di file.

- Es. ***basename s .h include/stdio.h ritorna*** stdio

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut
dirname rimuove l'ultimo componente del percorso da un nome di file (se non contiene /, restituisce "." directory corrente)

- Es. ***dirname /usr/bin*** ***ritorna*** /usr ; ***dirname data.txt*** ***ritorna*** .

eval permette di processare un file come se fosse uno script, sottoponendolo ai 12 passi di valutazione. Questo permette ad uno script di generare altri script ed eseguirli correttamente. Utile per rivalutare separatori: es. *listpage="ls | more"; $listpage* param. Exp avviene dopo la tokenization, quindi i caratteri "|" e "more" vengono intepretati come argomenti di ls. Con ***eval*** *$listpage* si ha invece una valutazione corretta.

Funzionamento in rete

Configurazione di rete

Richiami di reti, reti locali

Internet è una grande "rete di reti", la cui componente base (isola) è la singola rete IP; che contiene calcolatori che fanno da nodi terminali detti host. Gli indirizzi IP si dividono in globali (validi per tutta la rete, devono essere univoci e quindi vanno assegnati da una procedura di gestione globale che garantisca l'univocità) e locali, validi limitatamente ad una certa sottoparte della rete, che possono essere non globalmente univoci. (vedremo un ibrido tra queste due visioni quando vedremo che tra i vai metodi di assegnazione degli indirizzi IP ci sono metodi che partono da univocità di indirizzo locale (livello 2, MAC di scheda di rete) per derivare indirizzo di liv 3,IP che abbia univocità locale (senza considerare univocità globale) così si può ignorare network fisico, consentendo a questi di scoprire con automatismi come collegarsi con esterno (si sfrutta univocità indirizzi locali))

Viene detta rete fisica la rete (tipicamente LAN) a cui un host è effettivamente connesso; mentre è detta rete logica la network IP o (subnet) a cui un Host appartiene logicamente: sulla stessa LAN (fisica) possono esserci più subnet. A livello logico queste penseranno di non poter comunicare tra loro pur essendo sulla stessa struttura fisica. Tutti gli host appartenenti alla medesima network IP sono in grado di parlare tra loro grazie alla tecnologia con cui essa viene implementata. Per raggiungere hosts che si trovano sulla stessa LAN non è necessario sapere indirizzo di ognuno di loro, basta sapere come raggiungere la subnet.

Per il sistemista diventa necessario saper configurare un computer diversamente dal semplice host nodo terminale in una subnet: esempio, un router è semplicemente un computer con 2 interfacce (in/out) specializzato nello smistamento pacchetti.

Gli indirizzi IPv4 sono formati da 32bit divisi in 4 byte, ogni indirizzo fa parte di una subnet che inizia con l'indirizzo di network; l'estensione della subnet era implicita in origine (mediante classi di indirizzi, in cui i byte erano divisi tra net id e host id (qui indicata come *): Classe A da 0.*** a 127.*** -128 reti, fino a 16M hosts-; Classe B da 128.*** a 192.255.** -16K reti, fino a 65K hosts-, Classe C da 192.0.0.* a 223.255.255.* -2M reti, fino a 254 hosts-; indirizzi da 224.*** a 239.*** sono per il multicast, da 240.*** a 255.*** per usi futuri), oggi l'estensione è specificata da una netmask.

Avere poche classi con dimensioni fisse porta a spreco di indirizzi (es. chiedere una subnet di classe A ricevendo 16 milioni di indirizzi anche se ne servono solo 100.000…), quindi con CIDR (Classless Internet-Domain Routing) la divisione tra net-id e host-id è in un punto arbitrario (es. con 2^6=64 indirizzi, 6 bit di host id: 144.156.166.151 10010000.10011100.10100110.10 010111 unico vincolo è che la dimensione della rete sia potenza di due), mentre nel caso delle classi bastava guardare il primo numero per identificare la rete locale, ora non è più sufficiente: è necessario conoscere la netmask. Questa è un valore a 32 bit composto da tanti 1 quanti sono i bit della subnet e tanti 0 quanti sono i bit che specificano l'host (nell'esempio precedente, 26 1 e 6 0 11111111.11111111.11111111.11000000 = 255.255.255.192). Si può fare AND tra netmask ed

indirizzo, per confrontare indirizzi di destinazione e verificare se è nella stessa subnet di indirizzo di partenza. In ogni subnet, due indirizzi hanno significati speciali e non possono essere assegnati ad un host: quello con host-id con tutti 0 identifica la subnet (network address), con tutti 1 è indirizzo broadcast della subnet (se supportato, fa giungere pacchetto a tutti gli host della rete).

In ogni LAN ogni dispositivo ha un indirizzo MAC (mediante cui si ha inoltro fisico del traffico tra schede di rete), ma ha anche un indirizzo IP della rete. La traduzione di uno nell'altro avviene mediante ARP (Address Resolution Protocol). ARPsfrutta broadcast della rete fisica per chiedere chi possiede un certo indirizzo IP: se un host sa che appartiene ad una subnet, e vuole parlare con un host nella stessa subnet, SA ANCHE CHE è sulla stessa LAN, quindi gli basta chiedere in broadcast (tutti faranno caching opportunistico della richiesta, che contiene IP-MAC a cui rispondere) alla LAN chi possiede un certo IP per ottenere da lui indirizzo fisico a cui inviare pacchetti (risposta in unicast con coppia IP-MAC).

Nel livello fisico concreto (sul ferro), l'interfaccia di rete è un vero dispositivo, che ha un MAC tipicamente cablato (MAC modificabili sono eccezione più che regola). Far parte di una LAN cablata è determinato da connessione fisica di un cavo, oppure da gesto attivo di connettersi a wireless LAN, l'indirizzo IP va configurato. In una VM, l'interfaccia di rete è un artefatto gestito dall'hypervisor, mentre il SO guest la percepisce come un dispositivo e fornisce strumenti per configurarlo. L'hypervisor può impostare MAC (chiaramente è configurabile da chi gestisce hypervisor, non essendo cablato in una scheda fsica), definire a quale subnet è connessa la VM (instradamento è definito dall'hypervisor, il guest consegna a lui i pacchetti), può a volte gestire indirizzamento logico in modi particolari: può dire "Se esiste protocollo standard per configurare rete (esiste, DHCP), allora non chiedo al sysadm di farlo ma sfruttando il protocollo lascio che venga configurato in automazione".

~~In particolare in VirtualBox è permesso attivare molteplici interfacce per ogni VM, ognuna delle quali può connettersi in diversi modi, i principali: (comando VBoxManage utile per gestire varie VM da script)~~

- ~~**NAT**~~
  - ~~A default c'è una sola interfaccia in NAT: permette al guest di chiedere direttamente all'hypervisor un indirizzo per uscire sulla rete. Vedi dopo -->~~
- ~~**Bridged**~~
  - ~~Si comportano come se fossero connesse all'interfaccia dell'host attraverso un bridge/hub, quindi come se attestate sulla stessa LAN dell'host. Accesso materiale alla stessa rete offre grande vantaggio in termini di flessibilità , ma va fatta attenzione all'isolamento (la VM ha accesso diretto alla stessa rete delle macchina fisica: visibile dall'esterno come se fosse una macchina fisica connessa alla stessa rete dell'host).~~
    - ~~Vagrant può assegnare un'interfaccia bridged ad una VM con direttiva *config.vm.network "public_network"* Nel vagrantfile, entro il ciclo di configurazione do|config| se c'è questa direttiva, viene matchata con interfaccia bridged di virtualbox. A default assegna in automatico tutti i parametri, ma volendo con~~
      1. ~~*config.vm.network "public_network", ip: "192.168.0.17"* si può assegnare IP statico che verrà dato alla VM in fase di provisioning; usando vagrant per configurare il guest, assegnandogli direttamente un indirizzo IP~~
      2. ~~*config.vm.network "public_network", bridge: "eth0"* si può selezionare a quale interfaccia host agganciarsi~~

3. ~~*config.vm.network "public_network", auto_config: false* si può disabilitare automatismo per lasciare che il guest si occupi della configurazione, utile ad esempio nel caso nella VM ci sia un client DHCP: servirà un server DCHP sulla rete reale che fornisca configurazone alla VM.~~

- ~~**Host-only**~~
  - ~~Hypervisor genera interfaccia virtuale sull'host e le assegna un IP di una specifica subnet, poi connette alla LAN virtuale la VM, così che questa possa comunicare solo con l'host.~~
    - ~~Vagrant assegna interfaccia host-only a VM con direttiva *config.vm.network "private_network"*~~
      1. ~~*config.vm.network "private_network", name: "vboxnet3"* si può creare più di una rete host-only specificando il nome~~
      2. ~~valgono opzioni *ip* e *auto_config* come per bridged~~
- ~~**Internal**~~
  - ~~Hypervisore assegna queste interfacce a LAN completamente virtuale, e fa sì che solo interfacce della stessa internal network possano comunicare tra loro~~
    - ~~Vagrant le tratta come caso speciale di private_network *config.vm.network "private_network", virtualbox__intnet: "LAN1"*~~
      1. ~~Valgono opzioni *ip* e *auto_config* (queste sono opzioni specifica del provider Vbox, (dal punto di vista di vagrant, il provider è chi fornisce supporto alla virtualizzazione), potrebbero essere diverse con altri provider)~~

Reti globali

Le isole sono interconnesse da apparti che fanno da ponte (computer specializzati detti router o gateway) spesso realizzati con tecnologie diverse da quelle dell'isola. L'obiettivo di IP è rendere possibile il dialogo tra network a prescindere dall'implementazione, è realizzato per lavorare indifferentemente su tecnologie diverse. LAN con switch separati, interconnesse mediante un router, hanno possibilità di broadcast separato e possono contare su maggiore sicurezza (dovendo ogni host comunicare con il router per raggiungere le altre LAN interconnesse). I router (gateway) devono poter parlare sulle tecnologie di entrambi le network. Per capire se un dato pacchetto in uscita da un host deve andare sulla rete locale o se deve passare da un gateway, ogni nodo ha una base dati con le destinazioni (route) possibili, su una macchina linux è consultabile con ip route (abbreviato ip r).

La scelta del percorso su cui inviare i dati è detta routing. I router formano una struttura interconnessa, i datagrammi passano da uno all'altro fino a raggiungere quello che può consegnarli direttamente al destinatario: L'instradamento di un pacchetto si dice diretto quando IP sorgente e IP destinatario sono sulla stessa rete fisica, indiretto in caso contrario. Quindi da mittente a destinatario c'è sempre almeno una consegna diretta, e possono esserci zero (sorgente-destinazione sulla stessa rete fisica) o più (numero variabile di hop tra router) consegne indirette.

Sulla LAN le macchine si possono parlare elettronicamente con i MAC address. Su Internet le macchine possono parlare facendo dei salti fino al nodo destinazione.

A livello di trasporto, ulteriore indirizzamento in base alla porta; i pacchetti vengono spediti ad una porta perché una data applicazione li riceva, discriminando tra applicazioni in esecuzione sullo stesso host.

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut

NAT

La tecnica NAT (Network Address Translation) permette di usare un solo indirizzo pubblico per realizzare una rete privata anche di grandi dimensioni. La sua efficacia si basa sul fatto che la maggior parte degli host è solo client e non server, quindi originano richieste e devono poter essere raggiunti dalle risposte ma non devono poter essere raggiunti da richieste.

Posto un solo indirizzo pubblico fornito dal provider; nella rete locale il router che fa NAT si occupa, per le richieste in uscita, di sostituire indirizzo locale di origine con quello globale fornito dal provider, e per la risposta si ricorderà che deve consegnarla a quella una specifica macchina. Quindi molti IP sorgente vengono sostituiti da unico IP pubblico del router. Per connessioni che differiscono unicamente per l'IP sorgente, il router può anche fare sostituzione della porta sorgente per distinguerle (anche in questo caso memorizza traslazione per riconoscere destinatario risposte).

Gli IP della rete risultano completamente nascosti, scegliendoli arbitrariamente si potrebbe violare principio di univocità o tentare di connettersi a sé stessi. Per evitare il problema dell'oscuramento di IP validi, per standard si definiscono tre intervalli di indirizzi che non possono essere usati su Internet:

- 10.0.0.0/8 (da 10.0.0.0 a 10.255.255.255)
- 172.16.0.0/16 – 172.31.0.0/16 (da 172.16.0.0 a 172.31.255.255 = 172.16.0.0/12)
- 192.168.0.0/24 – 192.168.255.0/24 (da 192.168.0.0 a 192.168.255.255 = 192.168.0.0/16)

Questi indirizzi privati sono usati solo localmente, dietro il proprio router che fa NAT.

Il NAT può fornire una semplice conversione di indirizzo IP (statica o dinamica), le conversioni contemporanee sono limitate dal numero di IP pubblici a destinazione del gateway NAT. Può inoltre fornire conversione di IP + porta TCP o UDP (noto come PAT), in questo caso le conversioni contemporanee sono possibili anche con un unico indirizzo IP pubblico del gateway NAT.

Il NAT si applica di solito da rete privata verso rete pubblica (Outbound NAT): si tiene memoria delle connessioni e/o dei flussi di traffico (intrinsecamente stateful), le traduzioni in corso sono messe in cache (traduzioni con tempo di vita limitato), si preoccupa di fare conversione inversa quando arrivano pacchetti in direzione opposta relativi ad un flusso già attivo.

È anche possibile contattare dalla rete pubblica un host sulla rete privata (Bi-directional NAT), ma bisogna configurare in modo esplicito il NAT (**port-forwarding)**, e riutilizzare le porte se indirizzi sono limitati.

Per usare una rete di client con un solo IP pubblico si modifica l'IP sorgente (SNAT, Source NAT), con il traffico che fluisce attraverso il default gateway in modo trasparente ed automatico. Lo stesso gateway permette di rendere raggiungibili dall'esterno degli host della rete privata modificando l'indirizzo della destinazione, quando riceve richieste su di una specifica porta del proprio IP pubblico (DNAT, Destination NAT): la mappatura tra porta (servizio) e host interno a cui inoltrare la richiesta va configurata in modo esplicito (port forwarding, scelta specifica della porta da usare: si dice al router di sostituire indirizzo di destinazione globale dal suo (del router) con quello di un particolare host interno).

- ~~Vagrant stesso da fa SNAT Router:~~
  - ~~la VM viene configurata per usare come default gateway un'interfaccia virtuale che consegna i pacchetti al processo VirtualBox. Questo li propaga all'host come se li avesse generati lui, e quindi vengono etichettati con IP sorgente dell'host.~~

        - I parametri dell'interfaccia sono assegnati automaticamente, ma si possono configurare con *config.vm.base_mac* e *config.vm.base_address*

- Si può configurare VirtualBox perché si comporti anche da DNAT-Router:
    - Il processo Vbox si mette in ascolto su una porta TCP o UDP dell'host, il traffico entrant viene modificato assegnando come destinazione l'IP della scheda virtuale NAT del guest (impostando port-forwarding: vagrant fa questo da porta 2222 dell'host a porta 22 del guest, per permettere ssh).
        - Le mappature si configurano con, ad esempio *config.vm.network "forwarded_port", guest: 80, host: 8080*
        - il parametro opzionale *host_ip* può essere usato per limitare raggiungibilità della porta, es. *host_ip: "127.0.0.1"*

Configurazione

La configurazione di un'interfaccia di rete richiede come minimo IP e netmask; in più se la rete è connessa ad altre, gateway specifici e default gateway. Se è disponibile un sistema di risoluzione dei nomi, indirizzi dei server DNS e domini di default per costruire i FQDN.

Queste informazioni possono essere assegnate manualmente, da un server DHCP o localmente in modo automatico. Ancora una volta ci sono sia comandi per la modifica istantanea (runtime) della configurazione, sia altri per modifiche persistenti da applicare all'avvio.

L'approccio classico prevede editing di file di testo, seguendo lo standard di qualsiasi altro servizio. Le modifiche sono persistenti, ma è necessario che venga preso in considerazione il file (riavvio networking con *systemctl restart networking* o equivalenti). In molte distro c'è NetworkManager, che automatizza la rilevazione di nuove interfacce. Si può pilotare da GUI o con *nmcli*, e può sovrascrivere le modifiche runtime in qualsiasi momento, avendo un approccio dinamico. Il sistema di networking ha un approccio più semplice, tradizionale, con meno feature avanzate, ma ha grande pregio del determinismo: legge il file di configurazione e finché non si cambia il file e si riavvia, non cambia niente. Invece networkmanager viene risvegliato da eventi (es. rilevazione nuove schede di rete) e dinamicamente applica modifiche: importante sapere bene dove sono i file di configurazione per poter intervenire run-time.

- Configurazione runtime
    - suite iproute2 formata da comando ***ip*** e sottocomandi, permette controllo completo di tutti gli aspetti più avanzati (link-layer, interfacce virtuali, tunnel…).
        - sottocomando address (a) manipola interfacce:
            - visualizzazione ***ip a***
            - assegnazione ***ip a add <address>/<mask> dev <interface>***
            - rimozione ***ip a del <address>/<mask> dev <interface>***
        - sottocomando route (r) manipola tabelle di indirizzamento:
            - visualizzazione ***ip r***
            - routing via gateway ***ip r add <dst_net>/<mask> via <gw_addr>***
              (passa attraverso un router)

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut

- routing via dev ***ip r add <dst_net>/<mask> dev <interface>*** (passa attraverso interfaccia fisica (device))
- rimozione ***ip a del <address>/<mask>*** ***(****is this a correct?)*

- Configurazione persistente (classica, DEBIAN)
  - file /etc/network/interfaces vedi ***man 5 interfaces***
  - snippet nella cartella /etc/network/interfaces.d/
  - Esempio: *auto eth0* *# attiva con ifup -a*

iface eth0 inet static *#con dhcp al posto di static, non serve altro*

address 192.168.56.203

netmask 255.255.255.0 *# se omesso, class-based ip*

Opzionalmente

gateway 192.168.56.1 *# uno solo, non per interfaccia*

up /path/to/command arguments *# eseguito dopo configurazione*

Abilitazione packet forwarding: il kernel non inoltra i pacchetti a meno che non impostiamo il flag per dirgli di farlo.

- nel file **/etc/sysctl.conf** , decommentando la riga *#net.ipv4.ip_forward=1* e poi dando ***sysctl -p*** per applicare le modifiche
  - Commentare nuovamente l'opzione non resetta il valore, in quanto semplicemente si copre l'istruzione. Se si vuole riportarlo a 0, è necessario farlo esplicitamente
- Configurazione instantanea (non persistente) ??? ??? ***sysctl -w net.ipv4.ip_forward=1***

Tool per il monitoraggio: useremo i primi due tool per verificare la raggiungibilità delle reti che configuriamo

- ***ping <IP>*** verifica base della connettività
- ***traceroute <IP>*** verifica del percorso dei pacchetti
- ***ss*** verifica stato delle connessioni (socket status)
  - ***-t / -u*** TCP/UDP only
  - ***-l / -a*** stato LISTEN (default è ESTABLISHED)/ALL
  - ***-n*** non risolve indirizzi/porte in nomi simbolici
  - ***-p*** mostra processi che usano la socket
  - **-e** mostra info dettagliate sulle socket
- ***tcpdump*** e ***wireshark*** per intercettare contenuto dei pacchetti: lo stato delle socket è su ogni endpoint, con questi due invece possiamo vedere proprio i pacchetti in transito.
  - ***tcpdump*** tcpdump -nvl
    - -n output numerico, non converte indirizzi in nomi

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut

- -v output verboso, meglio toglierlo quando si mette in uno script ma utile per uso interattivo
- -l output line-buffered: fa output una linea alla volta, molto importante metterlo
- -c NUM esce appena ha ricevuto NUM pacchetti
- -i [any|INAME] specifica su quale interfaccia ascoltare [tutte oppure INAME]
- udp specifica il protocollo
- -w dumpa i pacchetti così come sono su un file invece di parsarli e stamparli su stdout (utile per creare file PCAP)
- poi con "and" concatena i filtri: (sicuro? lezione del 24-05 contiene riferimenti a tcpdump)
- con "src" imposta sorgente dei pacchetti (e con "net"? intende tutta una rete [mettendo quindi anche netmask], e non un singolo host?)
- con "dst" imposta destinazione dei pacchetti (con "net"? stesso dubbio di sopra. con "port" specifica porta destinazione, vale anche per source volendo)
- sudo tcpdump -nv -c 10 -i any udp and src net 127.0.0.0/24 and dst net 127.0.0.0/24 and dst port 514

Servizi di rete di base

NSS

La risoluzione dei nomi di host ad indirizzi IP e viceversa è uno dei vari casi in cui il sistema necessita di un dizionario di nomi. In GNU/Linux primo strato di risoluzione è NSS, che si occupa della scelta della sorgente di informazioni. NSS (Name Server Switch) è parte della libreria C di sistema, e supporta un set fisso di possibili DB; le categorie di nomi sono configurabili tramite il file **/etc/nsswitch.conf**

Il contenuto del file segue la sintassi *<entry>::= **<database>":"[<source> [<criteria>]]****

<criteria>::="[" <criterion> + "]"

<criterion>::=<status> "= " <action>

<status>::= "success" | "notfound" | "unavail" | "tryagain"

<action> ::= "return" | "continue"

A default, success (risposta ricevuta) ha come action *return* (ritorna un risultato ora), mentre notfound (sorgente esiste ma non sa rispondere), unavail (sorgente è irragiungibile) e tryagain (sorgente esiste ma è occupata) hanno *continue* (chiama la prossima funzione di lookup)

Esempi di entry: *passwd: files nis ldap*

group: files ldap

Esempio su record *hosts:ldap [NOTFOUND=return] dns files*

Il database hosts è quello dei nomi di host (siamo abituati a chiamarlo DNS ma è un'astrazione maggiore). Qui c'è scritto che per risolvere gli host, si va nella sorgente ldap, che può fare 4 cose:

- success, rispondere che lo trova, e la catena di chiamate torna all'origine col risultato
- unavail, risultato preliminare che vuol dire che ldap non riesce a rispondere: si passa a controllare dns
- tryagain, riprovare più tardi
- not found, per com'è scritta la regola, se ldap dà not found, c'è il return e si ferma la ricerca; quindi si dà massima autorità a ldap stesso.

Se <source> è *files* , si indica che NSS va a cercare un file locale con lo stesso nome della entry. Per questo record di esempio, quindi, la sorgente di informazioni sarà il file **/etc/hosts**

/etc/hosts ha formato <IP> <FQDN> [<ALIAS>…] fully qualified, con dominio per intero

esempio: 8.8.8.8 dns.google.com gnds

Altrimenti se <source> è *dns* , la sorgente di informazione è il DNS. L'interrogazione a questo è svolta da un altro set di funzioni della libsys C, il **resolver**. Questo si configura attraverso il file **/etc/resolv.conf** record di esempio *nameserver 137.204.58.1*

domain disi.unibo.it

search ing.unibo.it

Per il principio di caching multilivello (per migliorare le prestazioni) del DNS, avremo che il primo livello di cache è locale alla macchina: questa userà sé stessa come name resolver mediante un server DNS locale. Guardando nel file fondamentale lato client (**/etc/resolv.conf**) vediamo che è indicato *nameserver 127.X.X.X* poiché tutti gli IP che iniziano per 127 puntano a localhost.

Il DNS è un protocollo C/S basato su UDP che risponde sulla porta 53, potremo verificare la presenza del servizio con ***sudo ss -naup | grep 127.X.X.X:53*** , il comando mostra che c'è una socket unconnected (udp) legata alla porta 53, su cui gira ***dnsmasq*** pronto a ricevere.

Risoluzione nomi via NSS

Comando ***getent*** permette di interrogare i database del NSS: è il metodo per passare correttamente attraverso lo strato di astrazione fornito da NSS, che si occupa di scegliere la fonte delle informazioni restituite. Sintassi:

getent <dbname> <keyword> nome del db da interrogare e keyword su cui fare la ricerca

Esempi: *getent passwd las* se voglio vedere cosa c'è su etc/passwd, getent fa lookup di fonte dati "passwd" alla ricerca di una entry che abbia come keyword "las", passando da NSS per selezionare la fonte più adatta.

Risoluzione nomi DNS diretta

Per interrogare direttamente il DNS e avere più controllo sulle query, si usano ***host*** e ***dig***, che non considerano nsswitch e usano i nameserver di resolv.conf di default, ma possono interrogare un server specifico.

- ***host*** si usa tipicamente per conversioni IP ← → nome
  - query di un nome:*host www.unibo.it* interroga il server indicato in /etc/resolv.conf e restituisce risultato, se lo trova

- query di server specifico: *host www.unibo.it 8.8.8.8* così salta il file resolv.conf e va a chiedere direttamente al name server indicato (8.8.8.8) (citazione dei glue record - da ricordare per quando si diventa matti sulla risoluzione DNS, da sistemisti)

- ***dig*** per ottenere informazioni legate a un dominio, diverse da nomi host: i record ns dicono qual è il name server di un dominio, i record mx quali sono i mail exchanger.
  - conoscere i Mail eXchanger: *dig mx example.com*
  - conoscere i Name Server: *dig ns example.com*

~~Zeroconf~~

~~Se DNS pemette di associare nomi e indirizzi, ed è quindi un database compilabile, a mano ma anche in maniera automatica; lo **zeroconf** è uno standard per far funzionare un servizio appena connesso, automaticamente, senza configurarlo a mano. Questo non considera la comunicazione standard a livello applicativo con periferiche (come fa UpnP, Universal Plug and Play), ma i layer comuni a tutti:~~

- ~~**link-local addressing:** per determinare automaticamente un indirizzo di rete, assegnazione di indirizzi validi sulla LAN: gli indirizzi di layer 2 per definizione valgono solo localmente (MAC della scheda di rete è univoco sulla LAN ma non globalmente, non sono instradabili). Il link-local addressing si pone l'obiettivo di avere lo stesso approccio per il layer 3, per generare spontaneamente una subnet; ottenendo un IP univoco senza farselo consegnare da un nameserver o un DHCP.~~
  - ~~Link-local IPv4: riservata a questo scopo la classe 169.254/16, best practice per l'uso degli indirizzi sono: evitare l'assegnamento a interfacce che hanno indirizzi instradabili, non distribuirli con DHCP, non associarli stabilmente a nomi DNS (siccome servono solo sessione per sessione, per parlare IP tra le macchine). Assegnazione viene fatta solo se interfaccia non ha già indirizzo assegnato staticamente o via DHCP:~~
    - ~~Viene scelto IP random nel range 169.254.1.0 – 168.254.254.255, con seed legato a caratteristica univoca (es. MAC, per ridurre probabilità di conflitto e tendere a riassegnare IP stabili).~~
    - ~~Poi si verifica che qualcuno non abbia già l'indirizzo con ARP probe (IP disponibile)~~
    - ~~Infine si annuncia acquisizione con gratuitous ARP~~
  - ~~Link-local IPv6: gli indirizzi IPv6 sono divisi in 64bit di Subnet Prefix e 64 bit di Interface ID. IPv6 definisce un range per indirizzi link-local (logicamente equivale a 169.254/16), con prefix FE80::/10. (:: significa che ci sono solo zeri). Ogni interfaccia può costruire un suo IPv6 link-local seguendo lo schema:~~
    - ~~Si prende il MAC (, si allunga di 2 byte mettendo in mezzo i due byte fissi :ff:fe:~~
    - ~~Si inverte il 7° bit del primo byte (trasformato in binario ovviamente)~~
    - ~~Il risultato è l'Interface ID da concatenare a FE80:: (il subnet prefix)~~
  - ~~IPv6 ha sistema flessibile per determinare se indirizzo è libero e valutare se rete locale è raggiungibile dall'esterno: SLAAC è l'algoritmo per costruire indirizzi link-local validi globalmente configurando host in automatico (se possibile).~~
    - ~~Definizione token identificativo interfaccia (da MAC o valore random)~~

        - generazione indirizzo IPv6 link-local con prefix FE80::/10
        - invio di Neighbor Solicitation (algoritmo di scoperta dei vicini), se nessuno risponde si avanza, altrimenti indirizzo è occupato e processo automatico si ferma
        - invio di Router Solicitation all'indirizzo multicast di tutti i router (serve per scoprire se sulla rete ci sono router disposti a instradare il mio traffico).
            1. Se c'è risposta, si valuta la risposta:
                1. Risposta M: è necessario DHCP, si va nello stesso stato di mancanza router
                2. Risposta A (+subnet prefix): concatenazione del prefisso con indirizzo globale ottenuto
            2. Se non c'è risposta (nessun router), si fa richiesta in broadcast per scoprire se c'è server DHCPv6 che può darmi un indirizzo
                1. Risposta ricevuta, si assegna e usa l'indirizzo ricevuto
                2. Nessuna risposta, si mantiene indirizzo link-local.

- **multicast DNS:** per la traduzione di nomi in indirizzi in mancanza di DNS unicast configurato manualmente. L'associazione nomi-indirizzi viene gestita secondo standard, che definisce il TLD *.local.* come riservato ad host appartenenti a una rete link-local; e impone che le richieste di risoluzione per nomi che terminano in .local. siano inviate all'indirizzo link-local di multicast 224.0.0.251 (IPv4) o FF02::FB (IPv6), porta 5353. Inoltre lo standard raccomanda di strutturare i nomi in modo flat, sconsiglia domini di secondo, terzo... livello.
- **service discovery (DNS-SD):** basato su server DNS aggiornabile dinamicamente per registrare servizi, per scoprire nomi di servizi anche senza un DNS indicato specificamente. Offre la rilevazione automatica di servizi disponibili in rete, stabilendo un formato di entry DNS per descrivere la collocazione in rete, i protocolli applicativi ed eventuali parametri da utilizzare.

Implementazioni comuni di zeroconf sono Apple Bonjour (mDNS e SD), e Microsoft APIPA (solo link-local addressing).

Sincronizzazione

L'allineamento dell'ora di un sistema ad un orologio di riferimento è molto importante, ad esempio per la diagnostica dei problemi (timestamp su log), o per i protocolli di autenticazione e autorizzazione (spesso i protocolli di sicurezza si basano sulla validità temporale dei messaggi, token: es. per sapere se c'è un dato utente dietro alla tastiera, si usa un token per certificare che sia ancora là, ed è importante verificare il tempo in maniera corretta). E' possibile usare un protocollo, detto NTP (Network Time Protocol), che compensa i ritardi di rete per ottenere informazioni precise via internet. NTP è preciso (poche decine di ms di scarto su WAN, <1 ms su LAN, supporta sorgenti HW come oscillatori e GPS); standard (portato su ogni architettura), scalabile e affidabile (è multi-server, prevede stratificazione per permettere servizio a un grande numero di nodi (tipicamente non più di 2-3 strati; inoltre si può avere peering -i server allo stesso strato verificano l'uno con l'altro di essere allineati-).

Strumenti Linux per i servizi di rete

Linux offre diverse possibilità per la gestione dei servizi di rete: (nelle slide, link ai vari manuali)

- Client side:

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut

- Avahi (link-local, mDNS, DNS-SD)
- ISC dhcp (client DHCP per acquisire info su servizi di rete (e info simili))
- systemd.network (ogni possibile modo di configurare la rete, incluso link-local)
- systemd-resolved (DNS/mDNS resolver, sostituisce resolvconf)
- ntpd /ntpdate

- Server side:
  - dnsmasq (DCHP, DNS)
    - inoltre PXE (Pre-boot eXecution Environment) e TFTP (in laboratorio dell'uni c'è un server dnsmasq che permette di avere avvio sistemi diskless mediante PXE, TFTP viene usato per trasferire l'iso)
  - systemd.dnnsd (DNS-SD)
  - ntpd

Ci occuperemo principalmente di configurare **dnsmasq**: su reti di dimensioni ridotte è una scelta pratica per fornire i servizi necessari all'avvio zeroconf. Configurazione generale si fa su file ***/etc/dnsmasq.conf*** , dove possiamo specificare su quali interfacce esporre dei servizi.

(se all'esame, ci viene chiesto di configurare dnsmasq, consegnare un file con SOLO le righe utili, invece di quello con 8k righe dove controllare quali sono state scommentate e quali sono commentate...)

Opzioni base:

- *bind-interfaces* evita conflitti in caso si vogliano usare più istanze di dnsmasq per reti diverse connesse al server
- *interface=<interface name>* e
- *listen-address=<ipaddr>* mettono dnsmasq in ascolto solo sull'interfaccia o indirizzo indicati (anche più di una/uno)
- *user / group / pid* utente e gruppo UNIX del processo, file in cui salvare il PID

dnsmasq può fare da DCHP, il server DHCP è disabilitato se non sono specificate le opzioni descritte di seguito (altrimenti eroga indirizzi DHCP in un certo range o per un certo host):

- *dhcp-range=<start-addr>[,<end-addr>|<mode>][,<netmask>[,<broadcast>]][,<lease time>]*
  - questo specifica la rete da assegnare mediante indirizzo inizio, fine e netmask. Mettendo più di una riga, si specificano range per più reti/interfacce
  - indirizzi tra <start-addr> e <end-addr>, se specificato imposta il <lease time>
  - la <netmask> è opzionale per reti connesse direttamente al server
  - al posto di <end addr>, <mode> può essere *static* per abilitare il server sulla rete indicata senza servire indirizzi dinamici, ma solo quelli specificati con opzione *dhcp-host* (di seguito)
- *dhcp-host=[<hwaddr>][,<ipaddr>][,<hostname>][,<lease_time>][,ignore]*

- questo caso serve per assegnare indirizzi statici (sempre stesso indirizzo per stessa macchina)
- assegna <hostname>,<ipaddr> e <lease time> stabili all'host con MAC=<hwaddr>
- con *ignore* non fornirà mai un lease all'host indicato

- *dhcp-hostfile* permette di specificare una directory di file contenente informazioni formattate come la parte a destra dell'= di *dhcp-host*
- *dhcp-option=[<opt>|option:<opt-name>|option6:<opt>|option6:<opt-name>],[<value>[,<value>]]* uso più comune: *dhcp-option=<opt>,<value>*
  - dnsmasq fornisce un sacco di info, come convenzione si indica il gateway (solitamente il primo o ultimo indirizzo del range) con opzione -3, per indicare rotta di default. Es. riga del file /etc/dnsmasq.conf dhcp-option=3,1.2.3.4 indica che il gateway è su 1.2.3.4
  - Opzioni comuni:
    - 1 netmask
    - 2 fuso orario (offset da UTC)
    - 3 default gateway
    - 4 time server
    - 6 DNS server
    - 12 host name
    - 15 domain name
    - 121 static route (parametro:network/netmask,gateway)
      1. possibile definire anche rotte statiche: es. per andare alla rete XXXX si passa COMUNQUE da quel router lì

dnsmasq può fare da DNS, opzioni base sono

- *port=<dns server port>*
  - di default 53, se impostato a 0 disabilita il server DNS
- *local-service*
  - accetta query DNS solo dagli host delle subnet locali al server: tipica di default, previene DNS amplification attacks
  - ha effetto solo se non è specificata nessuna delle opzioni seguenti
    - *interface*
    - *except-interface*
    - *listen-address*
    - *auth-server options*

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut
~~Non fa da resolver ricorsivo, deve appoggiarsi ad uno esterno (prende coppie nome-indirizzo da qualcun altro, è un cached DNS e non un DNS autorevole): di default prende indirizzi dei nameserver upstream da /etc/resolv.conf (che viene aggiornato in automatico dai demoni es. dhcp e dnsmasq si accorge automaticamente delle modifiche).~~

~~Per usarlo localmente, bisogna aggiungere *nameserver 127.0.0.1* al file /etc/resolv.conf, inoltre è necessario configurare gli upstream server perché non ci sia loop di richiesta a sé stesso:~~

- ~~con *resolv-file=<file>*~~
  - ~~sopprime l'uso di /etc/resolv.conf~~
- ~~con *server=[/<domain>/]<ipaddr>*~~
  - ~~si deve aggiungere *no-resolv*, per evitare uso di /etc/resolv.conf~~

ntpd è client e/o server in funzione della configurazione, fatta su **/etc/ntp.conf** .

Criteri di sicurezza da usare con NTP: non avrebbe senso lasciarlo aperto; viaggia su UDP quindi qualcuno potrebbe inviare un pacchetto per far cambiare l'ora: mediante direttive si specifica di accettare l'ora solo da alcuni server.

Il tool ***ntpdate*** permette di sincronizzare l'orologio locale ad un server NTP, non rimpiazza ntpd, che usa algoritmi sofisticati per compensare errori e ritardi dei pacchetti dai server. ~~(dice che tutti parametri i configurazione li vedremo in laboratorio ma così non è stato?)~~. Senza parametri usa i server in /etc/ntp.conf (ntpd non deve essere attivo), ma accetta come parametro un server specifico. L'ora viene modificata con step se errore > 0.5sec, con slew (mediante *adjtime()*) se <0.5 sec.

Lato client: ~~(non mostrato???)~~ 17-06 FOR ALL I KNOW, THAT'S WHATS NEEDED FOR DHCP?

- DHCP
  - Pacchetto *isc-dhcp-client* fornisce comando **dhclient** , che lanciato senza parametri avvia un demone che tenta di configurare tutte le interfacce. Di solito è avviato da **interfaces** (/etc/network/interfaces) con

auto <ifname>

iface <ifname> inet dhcp

  - File **/etc/dhcp/dhclient.conf** per parametri impostabili
  - Hook per esecuzione automatica di script al cambio di stato interfaccia, nelle dir.
    - /etc/dchp/dhclient-enter-hooks.d/*
    - /etc/dhcp/dhclient-exit-hooks.d/*
- ~~zeroconf~~
  - ~~framework *avahi* offre stack completo mDNS/DNS-SD con API per integrazione delle funzionalità con qualsiasi programma C; un demone per gestire registrazione nuovi servizi in modo orchestrato da qualsiasi programma non scritto in C (via D-bus); un client/wrapper C che semplifica uso di D-Bus; adattatori per integrare le API di avahi negli event loop dei sistemi grafici come GNOME e KDE~~
  - ~~il demone è responsabile ad esempio della scoperta automatica di stampanti in una rete locale~~

- ~~sono disponibili pacchetti con strumenti per svolgere funzioni singole specifiche~~

- ~~link local~~
  - ~~pacchetto *avahi-autoipd* fornisce comando omonimo che implementa IPv4 Link Local, mediante demone indipendente, oppure, nel file */etc/network/interfaces*~~

~~*auto <ifname>*~~

~~*iface <ifname> inet ipv4all*~~

  - ~~ad ogni cambio stato dell'interfaccia invoca /etc/avahi/avahi-autoipd.action~~
  - ~~può essere usato come fallback se DCHP fallisce (come plugin per **dhclient**, usando hook nelle cartelle sopra indicate)~~

Sicurezza di rete

Sono possibili vari tipi di attacchi che sfruttano le debolezze dei protocolli di rete (sia a livello applicativo, che a livello di rete), ad esempio dirottamenti del traffico attraverso sistemi compromessi.

- **DNS spoofing:** Quando un utente effettua una query DNS, l'attaccante la cattura e manda alla vittima una risposta fasulla, differente da quella che avrebbe fornito il DNS. Questo attacco può anche essere portato a termine tramite pharming, quanto la vittima visita un sito compromesso uno script esegue una riconfigurazione del DNS locale del router redirigendo tutte le successive query DNS verso un name server scelto dall'attaccante. Questo attacco è molto semplice ma richiede l'accesso ad un name server e la possibilità di modificare direttamente alcuni record. L'impatto di questa categoria di attacchi può essere mitigato tramite l'uso di HTTPS.
- **HTTPS (Sicurezza a livello applicativo)**
  - Https è un protocollo per la comunicazione sicura attraverso reti internet, consiste nell'utilizzo del protocollo HTTP all'interno di una connessione cifrata da TLS (o dal predecessore SSL) fornendo autenticazione dei siti web visitati, protezione della privacy durante la comunicazione ed integrità dei dati. Il protocollo garantisce una protezione accettabile da eavesdropper e da attacchi della tipologia man in the middle. Grazie a TLS vengono cifrati tutti i dati contenuti nei messaggi HTTP, quali URL, parametri della query, cookies, header della connessione. Quando un browser si connette ad un server, la verifica della prova fornita dal server (un attaccante può provare a spedire certificato diverso, con sua chiave pubblica, che renderebbe inutile la prova crittografica) avviene mediante i certificati conservati nel Certificate Store.
  - **Certification Authority / Certificati digitali** Una certification authority è un soggetto terzo fidato abilitato ad emettere un certificato digitale, ovvero un documento elettronico che attesta l'associazione univoca tra una chiave pubblica e l'identità di un soggetto. L'infrastruttura PKI è costituita da varie CA organizzate gerarchicamente al cui vertice si trova una CA-root, il cui certificato solitamente è auto-firmato, che certifica le sub-CA. Una CA ha, tra i suoi compiti, il rilascio dei certificati, previa identificazione e verifica del richiedente, la manutenzione del registro delle chiavi, la revoca/sospensione dei certificati in caso di abusi/falsificazioni, la pubblicazione di liste sempre aggiornate di certificati. All'interno di un certificato digitale sono contenute varie informazioni tra cui la chiave pubblica del proprietario del certificato, l'identità del proprietario. Esse sono digitalmente firmate da parte della CA per garantirne l'autenticità e l'integrità. Il formato

più comune per i certificati è definito dallo standard X.509, che però, essendo molto generale, ha la necessità di essere ulteriormente specificato per i vari casi d'uso.

- **SSL/TLS:** Sono protocolli crittografici progettati per fornire sicurezza nelle comunicazioni attraverso reti internet, in particolare per fornire caratteristiche di confidenzialità ed integrità dei dati. La connessione è privata grazie all'uso della crittografia simmetrica, la cui chiave viene scambiata durante l'handshake iniziale (a sua volta protetto dalla crittografia asimmetrica). Inoltre grazie all'utilizzo della crittografia asimmetrica durante l'handshake iniziale viene garantita l'autenticazione delle parti comunicanti. L'handshake è composto da più fasi:

• Negoziazione: vengono concordate le caratteristiche della comunicazione, come protocollo di comunicazione, protocollo crittografico

• Scambio dei certificati

• Inizializzazione della connessione cifrata

A causa di alcune vulnerabilità, sia a livello di protocollo sia a livello di implementazione, SSL è stato sostituito con TLS.

- **Sicurezza a livello di rete:** Il protocollo IP non può garantire nessuna proprietà di sicurezza per nessuna parte del pacchetto.
  - **IP hijacking:** Si opera in modo tale da informare internet che la rotta per una data subnet passa attraverso la propria rete. Questo è possibile perché chiunque teoricamente può diffondere informazioni circa la viabilità delle rotte in internet e permette di realizzare vari tipi di attacco più o meno dannosi (DoS, man in the middle, spamma e fuggi).
  - **IPSec:** IPSec non è un protocollo singolo ma un insieme di algoritmi per la sicurezza ed un framework per la negoziazione di algoritmi che permettono di ottenere autenticazione e crittografia direttamente a livello di rete. IPSec ha varie applicazioni tra cui l'interconnessione sicura di reti remote attraverso internet e l'accesso sicuro di client ad una rete privata. Rispetto ad altre soluzioni IPSec presenta il vantaggio di essere trasparente alle applicazioni, tuttavia lo stack di IPSec è differente da quello standard di TCP/IP, pertanto la macchina va configurata per poter usare questo stack specifico invece di quello standard. IPSec è basato su tre componenti:

• AH (Authentication Header): fornisce un servizio di autenticazione dei pacchetti

• ESP (Encapsulating Security Protocol): fornisce un servizio di autenticazione e cifratura dei pacchetti

• IKE (Internet Key Exchange): fornisce un servizio di negoziazione dei parametri necessari al funzionamento dei precedenti componenti

Nel funzionamento di IPSec giocano un ruolo chiave le Security Association (SA) che identificano un canale di comunicazione unidirezionale diretto dal nodo locale verso un altro nodo e definito destinatario, protocollo utilizzato (AH o ESP), modalità di funzionamento (tunnel o trasporto) ed un Security Parameter Index (SPI, usato per distinguere tutti i canali aventi gli stessi estremi e lo stesso protocollo). Essendo una SA unidirezionale ne servono sempre due per definire un canale di comunicazione bidirezionale. La modalità di funzionamento transport prevede la comunicazione diretta tra due stazioni, ciascuna delle quali tratta i pacchetti e li spedisce tramite IPSec; la modalità tunnel invece prevede la presenza di almeno un router (il cosiddetto Security Gateway) che faccia

da tramite per la comunicazione con le macchine poste nella rete per la quale agisce da gateway, questa modalità è quella tipica delle VPN.

Il protocollo AH serve solamente per autenticare i pacchetti (eccezion fatta per i campi variabili come TTL). In caso venga usato in modalità trasporto ci si limita a frapporre tra l'header IP ed il payload l'intestazione di AH, che contiene tutti i dati relativi all'autenticazione del pacchetto; il pacchetto risultante mantiene l'header originale, anch'esso autenticato nei campi non variabili. In caso venga usato in modalità tunnel invece si crea una nuova intestazione IP, che conterrà solamente gli estremi del tunnel, e si autentica integralmente il pacchetto originale (i campi variabili della nuova intestazione non vengono autenticati), la nuova intestazione viene posta in testa al pacchetto, seguita dell'intestazione di AH e infine dal pacchetto originale replicato integralmente; quando il pacchetto verrà ricevuto dal security gateway esso provvederà a controllare l'autenticità del pacchetto ricevuto, a rimuovere l'intestazione usata per la trasmissione ed infine ad inoltrare il pacchetto al destinatario.

Qualora si voglia cifrare il contenuto del pacchetto invece occorre ricorrere all'uso del protocollo ESP. In caso si usi la modalità trasporto ci si limita a cifrare il payload del pacchetto IP interponendo tra l'header IP ed il payload l'intestazione di ESP; contestualmente viene fornita autenticazione sull'intero contenuto del pacchetto (ad esclusione dell'header IP). In caso venga usata la modalità tunnel invece, si genera una nuova intestazione IP ed il pacchetto originale viene cifrato integralmente, in questo modo non si potranno vedere gli indirizzi originali (che possono così essere anche indirizzi di reti private), contestualmente viene fornita autenticazione per quanto riguarda l'intero pacchetto originale e l'intestazione ESP. In ogni caso in coda al pacchetto vengono aggiunti due footer. Sebbene IPSec sia uno standard usato da molto tempo per la creazione di VPN la sua diffusione è molto limitata a causa dei problemi derivanti dall'interazione con il NAT. AH semplicemente non può funzionare in presenza di NAT in quanto esso autentica l'intero pacchetto, compresi gli indirizzi di sorgente e destinazione, che vengono modificati all'attraversamento di un NAT portando ad un fallimento del controllo sull'integrità del pacchetto. Con ESP il problema è più sottile e riguarda il checksum dei pacchetti TCP ed UDP, in questo caso non vi sono problemi di autenticazione in quanto l'header IP esterno non viene mai autenticato, tuttavia il checksum del pacchetto è calcolato sugli indirizzi originali e quindi se il pacchetto è cifrato con IPSec questo valore non può essere aggiornato facendo fallire i controlli sugli errori a destinazione. Per risolvere questi problemi è stata proposta un'alternativa al NAT classico chiamata NAT-T che prevede che ogni volta che un pacchetto raggiunge un NAT esso venga incapsulato all'interno di un pacchetto UDP per poi venir inviato tramite lo stack TCP/IP standard. Questa soluzione tuttavia pone dei problemi per quanto riguarda l'efficienza e rende l'utilizzo di IPSec equivalente a livello pratico a quello di altre soluzioni come OpenVPN.

- **VPN:** Una Virtual Private Network è una rete di telecomunicazioni privata, instaurata tra soggetti che utilizzano, come tecnologia di trasporto, un protocollo di trasmissione pubblico e condiviso, come ad esempio la rete Internet. Lo scopo delle reti VPN è quello di offrire alle aziende, a un costo minore, le stesse possibilità delle linee private a noleggio, ma sfruttando reti condivise pubbliche: si può vedere dunque una VPN come l'estensione a livello geografico di una rete locale privata aziendale sicura che colleghi tra loro siti interni all'azienda stessa variamente dislocati su un ampio territorio, sfruttando l'instradamento tramite IP per il trasporto su scala geografica e realizzando di fatto una rete LAN, detta appunto "virtuale" e "privata", equivalente a un'infrastruttura fisica di rete (ossia con collegamenti fisici) dedicata.

Tuttavia l'uso di una VPN non è trasparente all'utente in quanto è necessario istruire il sistema affinché invii i dati attraverso un'interfaccia di rete virtuale offerta dal programma che gestisce la VPN stessa. Per l'autenticazione degli host si può ricorrere ad una modalità "statica", con scambio

manuale delle chiavi crittografiche, oppure ai protocolli SSL/TLS, affidandosi all'architettura PKI per lo scambio delle chiavi.

- **Firewall:** Un firewall può essere paragonato ad una porta blindata che divide l'interno (da proteggere) dall'esterno (da cui provengono delle minacce), in questo senso un firewall permette di bloccare gli accessi indebiti alla rete. Così come una porta blindata un firewall può bloccare certe minacce, ma diventa inutile nel momento in cui vengono lasciate altre falle aperte, pertanto è buona norma porlo al di sopra di un sistema sicuro in modo da ridurre le vulnerabilità del firewall stesso. Le tecniche di controllo prevedono l'esame di:

• Traffico: esaminare di indirizzi, porte ed altri indicatori relativi al tipo di indirizzo che si vuole esplicitamente autorizzare.

• Direzione: discriminare il traffico in base alla direzione a parità di indirizzi e porte

• Utenti: differenziare il traffico in base a chi lo genera

• Comportamento: valutare come sono usati i servizi ammessi per identificare le anomalie

    - **Tipologie:** Si identificano tre tipi fondamentali di firewall:

1. Packet filter: sono in grado di esaminare unicamente l'header dei pacchetti applicando in serie un insieme di regole, generalmente raggruppate in degli elenchi corrispondenti a punti di controllo differenti (pacchetti in ingresso, in uscita, ...), del tipo "se condizione allora azione". Normalmente la prima condizione ad essere verificata decide il destino del pacchetto ed interrompe la scansione dell'elenco. Di base le azioni sono accettare il pacchetto o scartarlo, ma possono anche essere intraprese azioni più complesse come loggare i dettagli del pacchetto oppure modificarlo in qualche modo. Se nessuna regola è applicabile viene attivata una policy di default. Questo tipo di firewall porta con sé alcuni vantaggi, ad esempio è molto veloce e semplice (tanto che è implementato in moltissimi router) ed è trasparente all'utente (non occorre modificare il sistema in nessun modo, il firewall si occupa autonomamente di scansionare il traffico di rete); tuttavia le regole sono spesso di basso livello, facendo sì che per ottenere comportamenti complessi occorra costruire set di regole molti articolati, e mancano del supporto alla gestione degli utenti. Inoltre i PF presentano alcune vulnerabilità e limitazioni:

• i pacchetti frammentati possono comportarsi in maniera non prevedibile (si potrebbe riassemblare ogni pacchetto all'interno del firewall, ma questo fa sì che il carico computazionale cresca molto rapidamente)

• poiché i controlli vengono effettuati solamente sulla base dei dati presenti all'interno degli header i PF soffrono in presenza di spoofing. Per limitare i problemi è necessario controllare la coerenza tra gli indirizzi e le interfacce e controllare certi indirizzi particolari (indirizzi multicast, illegali, broadcast, riservati, …)

• il filtraggio non è in grado di lavorare in presenza di protocolli che negoziano l'apertura di porte dinamicamente (ad esempio connessione dati FTP)

• non è possibile implementare difese contro attacchi data-driven, poiché viaggiano all'interno del payload del pacchetti

Tipicamente i PF sono stateless, tuttavia in alcuni casi è possibile usufruire di PF stateful; essi sono in grado di analizzare il traffico sulla base dei pacchetti già visti. È possibile anche avere dei *Multilayer protocol inspection firewall* che sono in grado id garantire la coerenza del protocollo tenendo traccia dell'intera storia della connessione

2. Application-level gateway (anche chiamato proxy server): può essere definito come un man in the middle" buono, in grado di propagare il traffico verso i server effettivi. In questo modo è in

grado di comprendere i protocolli applicativi e di controllare anche il payload (ad esempio per bloccare spam/virus per quanto riguarda i server di posta o per bloccare malware/phishing per quanto riguarda i server web). Tuttavia i firewall appartenenti a questa tipologia sono moltopiù pesanti di un PF e specifici per un singolo protocollo applicativo. Poiché agiscono come server nei confronti dei client non è detto che siano trasparenti, ma è possibile che sia necessario un minimo livello di configurazione.

3. Ciruit-level-gateway: spezzano la connessione a livello di trasporto diventando endpoint del traffico ed inoltrando i payload senza esaminarli. Tipicamente sono utilizzati per determinare quale sia il traffico ammissibile dall'interno verso l'esterno. I firewall appartenenti a questa categoria hanno il vantaggio di poter essere configurati per essere trasparenti agli utenti per autorizzare solamente le connessioni fidate, inoltre possono agire da intermediario generico (non sono limitati ad un singolo protocollo) e possono essere estesi con altri tipi di firewall per ottenere servizi più complessi. Tuttavia le regole sono limitate ad indirizzi, porte ed utenti.

- **SOCKS:** SOCKS (Socket Secure) è un protocollo di livello 5 per lo scambio di pacchetto tra un client ed un server per mezzo di un proxy server. In aggiunta fornisce l'autenticazione dei pacchetti, pertanto solamente gli utenti autorizzati sono in grado di accedere al server. Questo protocollo è lo standard de facto per l'implementazione dei CLG (Curcuit-level-gateway).
- **Tor:** Il protocollo di Tor permette di realizzare connessioni cifrate in cui il legame tra chi effettua richieste e il contenuto delle stesse è profondamente oscurato. Per prima cosa si ottiene un elenco di nodi Tor da un directory server, successivamente ne vengono scelti alcuni casualmente per creare un percorso tra mittente e destinatario su cui inviare un messaggio cifrato a "cipolla", ad ogni hop viene rimosso uno strato di cifratura fino a che il server non è in grado di leggere il traffico in chiaro. I nodi Tor formano un overlay rispetto alla rete internet, pertanto è possibile che tra un nodo e l'altro il traffico attraversi vari router, tuttavia questo non presenta problemi grazie ai vari livelli di crittografia.

Ogni relay conosce solamente il nodo prima di lui e quello dopo, pertanto viene reso molto difficile scoprire chi sia il vero mittente. Tuttavia il protocollo presenta alcune debolezze intrinseche:

• è possibile correlare il traffico dei vari nodi per capire quale percorso facciano i pacchetti, sebbene siano cifrati

• gli exit node vedono il traffico in chiaro, è sebbene non siano in grado di capire automaticamente chi sia il mittente i payload dei pacchetti potrebbe contenere informazioni ben più rilevanti ai fini dell'identificazione

• è sufficiente che all'interno della rete vi sia anche un solo nodo compromesso per compromettere l'intera rete

Packet Filtering

netstat è il comando utile a verificare le informazioni relative alle connessioni di rete.

- **netstat -na --ip** visualizza le informazioni relative alle connessioni TCP attive
- **netstat -s** visualizza una serie di dati di tipo statistico sul traffico relativo ai diversi protocolli

nmap è il comando per fare port scanning su un host e visualizzare le porte TCP corrispondenti a processi in ascolto (server). Il port scanning consiste nell'invio di un SYN verso ogni porta: se si

riceve un SYN+ACK la porta è attiva, se si riceve un RST+ACK la porta è chiusa. Se non si riceve nulla, la porta è filtrata.

Un firewall è un filtro software che serve a proteggersi da accessi indesiderati provenienti dall'esterno della rete, può essere un programma locale all'host (protegge questo da attacchi esterni e la rete da attacchi generati da potenziali virus presenti sull'host) o una macchina dedicata che filtra tutto il traffico in/out su una rete locale.

Il traffico tra la rete locale ed Internet deve essere filtrato dal firewall, solo quello autorizzato deve attraversarlo. In fase di configurazione di un firewall, per prima cosa si decide la politica di default per i servizi di rete:

- **default deny**: tutti i servizi non esplicitamente permessi sono negati
- **default allow:** tutti i servizi non esplicitamente negati sono permessi

Un firewall può essere implementato come

- **Packet filter:** si pone tra rete locale e Internet, filtrando i datagrammi IP da trasferire sulle interfacce scartandoli in base a interfaccia (source e/o dest), MAC e/o IP (source o dest), tipo di servizio (campo PROTOCOL o porta TCP/UDP). Può essere **stateful/stateless** nel caso in cui tenga memoria o meno delle connessioni e/o dei flussi di traffico in corso.
- **Application layer firewall** o **proxy server**: filtra i dati in base ai protocolli applicativi (HTTP, FTP…) e l'accesso alla rete esterna è possibile solo attraverso il proxy.

IPTABLES implementa uno stateful packet filter nei kernel Linux 2.4 e successivi, lavora a livello di kernel ed ha il controllo dei pacchetti IP in transito sulle interfacce di rete (loopback compreso). I pacchetti processati sono sottoposti a diverse modalità di elaborazione chiamate **table**, ciascuna delle quali è composta da gruppi di regole dette **chain**. IPTABLES definisce quattro table principali

- **filter** filtraggio di pacchetti
- **nat** sostituzione di indirizzi IP
- **mangle** manipolazione ulteriore di pacchetti (TOS, TTL..)
- **raw** esclusione di pacchetti dal connection tracking

- Tabella **FILTER:** contiene le vere funzionalità di firewall. Si occupa di filtrare i pacchetti in base all'interfaccia di provenienza e dei parametri contenuti nell'header IP e TCP. Ha 3 chain predefinite, ed è possibile definirne ulteriori:
  - **INPUT**: contiene le regole di filtraggio da usare sui pacchetti in arrivo al firewall (destinati all'utente locale). (quali pacchetti destinati a processi locali possono raggiungerli)
  - **OUTPUT**: contiene le regole da usare sui pacchetti in uscita dal firewall (originati dall'host locale) (quali pacchetti generati da processi locali possono raggiungere la destinazione)
  - **FORWARD**: contiene le regole da usare sui pacchetti in transito nel firewall (inoltrati tra interfacce diverse) (quali pacchetti destinati a destinazioni esterne possono passare)
  - Regole della tabella FILTER: quando un pacchetto viene processato da una chain, è soggetto alle regole specificate in essa, secondo l'ordine di inserimento. Una regola può decidere se scartare (DROP), rifiutare esplicitamente (REJECT) o accettare (ACCEPT) un pacchetto in base a interfaccia coinvolta, IP source/dest, protocollo (TCP, UDP, ICMP),

porta source/dst, tipo di messaggio ICMP. Se un pacchetto non soddisfa nessuna regola, viene applicata la regola di default, o **policy**, di quella chain.

- Opzioni per gestione della tabella FILTER:
  - ***iptables -L [-nv –line-num]*** per visualizzare le regole attualmente in uso da ogni chain della tabella filter. -n risultato numerico (non c'è reverse lookup del nome), -v verboso
  - ***iptables -L <chain>*** visualizza regole in uso da una chain specifica
  - ***iptables -P <chain> <policy>*** imposta la policy di default di una chain
  - ***iptables -A <chain> <rule-specs> -j <policy>*** aggiunge una regola in coda ad una chain.
    - <chain> è INPUT | OUTPUT | FORWARD | …
    - <policy> è ACCEPT | DROP | REJECT | … (REJECT non è ammessa come policy di default)
      - policy **LOG** indica al kernel di loggare i dettagli del pacchetto. Opzioni
        - --log-level <priority> indica una priority
        - --log-prefix <prefisso> indica un prefisso
        - --log-uid inserisce user-i???
      - ad ogni regola sono associati due contatori che vengono incrementati ogni volta che un pacchetto fa "match" (un contatore di pacchetti, uno di byte cumulativamente trasportati da essi). Una regola senza policy permette di conteggiare il traffico con alcune caratteristiche senza interferire con transito pacchetti
    - <rule specs> specifica su quali pacchetti applicare la regola
  - ***iptables -I <chain> <N> <rule specs> -j <policy>*** inserisce regola in posizione N nella chain indicata
  - ***iptables -R <chain> <N> <rule specs> -j <policy>*** sostituisce regola in posizione N nella chain indicata
  - ***iptables -D <chain> <N>*** elimina la regola in posizione N di una chain
  - ***iptables -F <chain>*** flush di tutte le regole di una specifica chain, o omettendo<chain>, da tutte quante (non agisce su policy di default)
- Specifica di regole <rule specs>
  - **-i** <interface> e **-o** <interface> specificano interfaccia di ingresso/uscita
  - **-s** <address>/<netmask> e **-d** <address>/<netmask> specificano IP (host o rete) di origine/destinazione (netmask può essere in formato decimale puntato o CIDR
  - **-p** <tcp|udp|icmp|...> specifica il protocollo, /etc/protocols contiene lista supportati

- **--sport** <port> e **--dport** <port> specifica porta di origine/destinazione (/etc/services contiene elenco porte corrispondenti ai servizi)
- **--icmp-type** <typename>specifica tipo di messaggio ICMP (***iptables -p icmp -h*** mostra elenco tipi riconosciuti da IPTABLES)
- **--mac-source** <MAC_address> specifica MAC di origine
- **!** per negare un'opzione (es. *! -s <address>/<netmask>)*
- **-m state --state <NEW|ESTABLISHED>** per specificare pacchetti di connessioni TCP o flussi UDP nuovi o già attivi (stateful packet filter). Stati di flusso/connessioni definiti da IPTABLES (conntrack):
  - **NEW:** generato da un pacchetto appartenente a un flusso/connessione non presente nella tabella conntrack
  - **ESTABLISHED:** associato a flussi/connessioni dei quali sono stati già accettati pacchetti precedenti, in entrambe le direzioni
  - **RELATED:** associato a flussi/conness. Non ESTABLISHED ma che sono correlati con flussi/conness. ESTABLISHED (es. connessioni FTP control e FTP data)
  - **INVALID:** associato a pacchetti con stato non tracciabile
  - **UNTRACKED:** associato a pacchetti non soggetti alla modalità stateful (tabella **raw)**
  - ***conntrack -L*** comando per visualizzare stato di flussi/conness.

- Tabella **NAT:** implementa le funzionalità di NAT in IPTABLES. Ha quattro chain predefinite
  - **PREROUTING**: contiene regole da usare prima dell'instradamento per sostituire indirizzo destinazione dei pacchetti (policy = Destination NAT o **DNAT**)
  - **POSTROUTING:** contiene regole da usare dopo l'instradamento per sostituire l'indirizzo di origine dei pacchetti (policy = Source NAT o **SNAT**)
  - **OUTPUT/INPUT:** contiene regole da usare per sostituire l'indirizzo di pacchetti generati/ricevuti localmente

La policy **ACCEPT** indica assenza di conversione, la policy **MASQUERADE** indica conversione implicita dell'IP assegnato all'interfaccia di uscita

- Opzioni per gestione della tabella NAT:
  - ***iptables -t nat -L [-nv --line-num]*** visualizzare regole attualmente in uso da ogni chain della tabella NAT
  - ***iptables -t nat -L <chain>*** visualizza regole attualmente in uso da una chain specifica
  - ***iptables -t nat -A<chain> <rule-specs> -j <policy>*** aggiunge regola in coda ad una chain
    - <chain> è POSTROUTING | PREROUTING | OUTPUT | INPUT | …

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut

- <policy> è ACCEPT | MASQUERADE | SNAT --to-source <addr> | DNAT --to-destination <addr>
  - <addr> è <address> | <address>:<port>
- <rule specs> uguale a tabella FILTER

- Case study (ESEMPIO config firewall e NAT)
  - Tutte le LAN aziendali che usano IP pubblici devono essere accessibili senza restrizioni da parte di connessioni provenienti da altre LAN aziendali. **[1]**
  - I server nella LAN 1 devono essere accessibili da parte di connessioni non provenienti da LAN aziendali, solo se dirette verso i servizi HTTP e HTTPS. **[2]**
  - Almeno un host della LAN 2 deve poter essere raggiungibile via SSH dalla LAN 3. **[3]**
  - Le altre LAN aziendali non devono essere accessibili da connessioni originate dall'esterno, ma tutti i loro host e server devono poter accedere all'esterno senza limitazioni. **[4]**
    - LAN 1: config firewall
      - *iptables -P FORWARD DROP #policy di default*

iptables -A FORWARD -i eth0 -m state --state NEW -j ACCEPT #accetta pacchetti per nuove connessioni su eth0 **[4]**

iptables -A FORWARD -i ppp1 -s 192.168.0.0/24 -m state --state NEW -j ACCEPT #accetta nuove connessioni su ppp1 provenienti da LAN2 **[1]**

iptables -A FORWARD -i ppp0 -s 155.108.131.0/24 -m state --state NEW -j ACCEPT #accetta nuove connessioni su ppp0 provenienti da LAN3 **[1]**

iptables -A FORWARD -i ppp0 -d 87.15.12.0/24 -p tcp --dport 80 -m state --state NEW -j ACCEPT #accetta nuove connessioni su ppp0 verso i server (dest LAN 1), a patto che siano prot TCP verso porta 80 (servizio HTTP) ***[2]***

iptables -A FORWARD -i ppp0 -d 87.15.12.0/24 -p tcp --dport 443 -m state --state NEW -j ACCEPT #accetta nuove connessioni su ppp0 verso i server (dest LAN1), a patto che siano prot TCP verso porta 443 (servizio HTTPS) ***[2]***

iptables -I FORWARD 1 -m state --state ESTABLISHED -j ACCEPT #accetta pacchetti di connessioni già stabilite, pone regola in posizione 1

    - LAN 2: config firewall e NAT
      - *iptables -P FORWARD DROP #policy di default*

iptables -A FORWARD -i eth0 -m state --state NEW -j ACCEPT #accetta nuove connessioni su eth0 **[4]**

iptables -A FORWARD -i ppp1 -s 87.15.12.0/24 -m state --state NEW -j ACCEPT #accetta nuove connessioni su ppp1 provenienti da LAN1 **[1]**

iptables -A FORWARD -i ppp0 -s 155.108.131.0/24 -m state --state NEW -j ACCEPT #accetta su ppp0 pacchetti per nuove connessioni provenienti da LAN3 **[1]**

iptables -I FORWARD 1 -m state --state ESTABLISHED -j ACCEPT #accetta pacchetti di connessioni già stabilite, pone regola in posizione 1

iptables -t nat -A POSTROUTING -s 192.168.0.0/24 -o ppp0 -j SNAT --to-source 87.4.8.21 # sostituisce indirizzo di provenienza dei pacchetti (originati in LAN2) in uscita su ppp0 con "87.4.8.21" **[3]**

iptables -t nat -A PREROUTING -i ppp0 -d 87.4.8.21 -p tcp --dport 2222 -j DNAT --to-destination 192.168.0.1:22 # sostituisce indirizzo di destinazione dei pacchetti in ingresso su ppp0 (destinati a "87.4.8.21"), se hanno prot. TCP e porta 2222, con "192.168.0.1:22" (servizio SSH su host 192.168.0.1) **[3]**

    - LAN 3: config firewall
      - *iptables -P FORWARD DROP #policy di default*

iptables -A FORWARD -i eth0 -m state --state NEW -j ACCEPT #accetta nuove connessioni su eth0 **[4]**

iptables -A FORWARD -i ppp0 -s 87.15.12.0/24 -m state --state NEW -j ACCEPT #accetta nuove connessioni su ppp0 provenienti da LAN1 **[1]**

iptables -A FORWARD -i ppp0 -s 87.4.8.21 -m state --state NEW -j ACCEPT #accetta nuove connessioni su ppp0 provenienti da "87.4.8.21" **[3]**

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut
iptables -I FORWARD 1 -m state --state ESTABLISHED -j ACCEPT #accetta pacchetti di connessioni già stabilite, pone regola in posizione 1

SNMP

La gestione di apparati in rete è problematica in quanto ognuno di questi dispone, per configurazione e monitoraggio, di strumenti proprietari, quindi c'è necessità di interfacciarsi con linguaggi diversi a seconda che si voglia gestire ad esempio computer di SO diversi, apparati di rete di produttori diversi. Per la configurazione persistente, i tool sono inevitabili (anche se oggi il mondo del software-defined network fa sì che si usino apparecchi generici per la costruzione delle reti, e le regole vengono decise da un control plane (a livello più alto); facendo sì che ci siano protocolli standard) ma per il monitoraggio di base sono un ostacolo all'automazione (infatti spesso si ricorre al wrapping degli strumenti proprietari per avere un unico linguaggio per interfacciarsi con vari di questi). Ci sono inoltre svantaggi di sicurezza nei protocolli di accesso generici (es. TELNET offre canale non sicuro, shell interattiva lascia disponibili troppe funzionalità). La risposta a questi problemi è SNMP (Simple Network Management Protocol), standard molto diffuso per sorvegliare grande quantità di macchine con approccio centralizzato, invece di usare comandi locali. Si desiderava un protocollo che di mestiere facesse solo il management degli apparati connessi in rete: "Simple" perché basilare, con due scopi:

- Essere meno invasivo possibile, aspetto di sicurezza: certe cose non si possono fare quindi certi errori non si possono fare, anche volendo
- Costi: ogni genere di apparato di rete (telecamere, stampanti...) oggi disponibile a basso costo, quindi senza un microprocessore con SO. La semplicità deve consentire di scrivere firmware sui microcontrollori senza aumentare i costi di produzione.

SNMP standardizza:

- il modello dei dati, per cui qualunque proprietà di rete mi interessi identificare, viene rappresentata come oggetto. Es. posizione di un'apparato: potrebbe essere utile sapere dove si trova fisicamente, se arriva un notifica in ufficio che apparato sta andando a fuoco.
    - La proprietà ha ricevuto nome univoco (1.3.6.1.2.1.1.6) e ha ricevuto una sintassi standard (stringa di non più di 255 caratteri). Sintassi e nome univoco possono essere letti online, la definizione formale è parte della dotazione di SNMP.
- Il modello di interazione: questi dati dove stanno? Sull'apparato; questi dati chi li usa? il gestore, che sta nel suo ufficio e controlla centinaia di apparati. Ci vuole un protocollo per recuperare queste informazioni, ed SNMP ne offre uno applicativo veicolato su UDP che gestisce la comunicazione tra dispositivi ed entità che li gestisce

Modello dei dati: OID e MIB

Alla base di SNMP è un modello generico per inquadrare qualsiasi oggetto concreto, proprietà di un oggetto, o concetto astratto: ognuno di questi è identificato univocamente da un OID (Object IDentifier). Il modello è stato concepito dall'ITU, con l'obiettivo di fare una tassonomia di qualsiasi proprietà/oggetto che fa parte del mondo delle telecomunicazioni. La gerarchia parte da un radice anonima ("."), da cui discendono tre archi (0 ITU-T, 1 ISO, 2 joint-iso-itu-t). Dalla radice alla foglia, si ha sequenza di numeri che porta all'oggetto: ogni nodo ha id numerico e simbolico (es. 1.3.6.1 == iso.identified-organization.dod.internet). La sequenza dei numeri è propriamente la tassonomia delle proprietà e non ne individua un valore (qualsiasi, colore di tubo per gas pericoloso, numero di bulloni passati da settore produzione…): quando dico 1.3.6.xxx non sto dicendo che conto quanti pacchetti passano, sto dando un nome alla proprietà di quanti pacchetti passano.

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut
SNMP è sotto internet, che a sua volta è sotto dod (department of defence) essendo questo stato il principale finanziatore di internet: iso.identified-organization.dod.internet...

...internet.mgmt.mib.system.syslocation quindi 1.3.6.1.2.1.1.6 è il nome dato all'oggetto "posizione dell'apparato", e la descrizione di questo oggetto contiene la sintassi della proprietà, oltre allo stato… Stesso discorso per ogni proprietà che mi interessi codificare.

Viene detto MIB (Managed Information Base), la collezione degli oggetti gestiti da un apparato o da un sistema di monitoraggio. E' la collezione degli oggetti che voglio avere a disposizione per descrivere le proprietà di un dispositivo:

- da un lato è una collezione teorica (es. dizionario, che contiene "tutte" le parole) che mi dice come sono identificati (con stringa di numeri) e definiti (con la loro sintassi) anche tutti gli oggetti del mondo (il MIB globale contiene identificazione e definizione di tutte le proprietà possibili!); quindi descrizione dell'intero albero globale degli OID;
- dall'altro lato, comprato un apparato, mi chiedo qual è il MIB che descrive proprietà di quell'apparato: sarà un sottoalbero di quello globale, fatto di pezzetti di quello globale. Le proprietà dell'apparato potrebbero anche non essere direttamente nel MIB, perché potrebbero essere generate dinamicamente: nel MIB globale ci sarà scritto allora che c'è una tabella con i dati. Quando vado a controllare contenuto MIB del mio dispositivo, mi accorgo che ci saranno più proprietà di quelle prevedibili: nel MIB ci sono le colonne definibili, ma non le righe possibili!

In sostanza un MIB è un catalogo che associa ad ogni oggetto un OID, una sintassi (tipo di dato) una codifica (descrive la rappresentazione materiale, per rendere possibile la comunicazione tra architetture diverse); e formalmente usa linguaggio SMIv2, sottoinsieme di ASN.1

Tutte le volte che viene fuori un apparato, questo mi metterà a disposizione le sue proprietà attraverso SNMP, mediante un sottoinsieme di MIB, consultabile online. Altro caso è quello in cui è il venditore a dover dare informazioni (che non si troveranno sul sito globale): esistono aziende private che hanno comprato un sottoalbero, in cui possono fare cosa vogliono senza chiedere all'ISO. Es. venditore di telecamere IP si inventa nuovo modo di commutare tra visione diurna/notturna; la proprietà che permette di interrogare la telecamera per sapere quale mode sta usando sarà codificata e interrogabile via SNMP, ma il modo in cui interrogarla dovrà esserci detto dal venditore, non sarà sul sito globale delle proprietà. Il modulo MIB relativo a un certo set di informazioni deve essere noto a chi vuole interrogare il dispositivo, per sapere quali informazioni sono disponibili e come interpretarle.

Le due sintassi dati supportate sono SMIv1 e v2, quest'ultima aggiunge alcune migliorie (sottolineate):

- Tipi di dato semplici:
  - interi a 32 bit con segno
  - stringhe di byte (lunghezza max. 65535)
  - OID
- Tipi di dato application-wide:
  - network addresses: come IPv4, come generiche stringhe di byte (anche IPv6, in v2)
  - counters: interi a 32/64 bit positivi e crescenti (a differenza degli interi possono solo salire), rollover a zero

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut

- gauges: interi non negativi con limiti minimo e massimo
- time ticks: centesimi di secondo trascorsi da un dato evento
- opaques: stringhe arbitrarie senza controllo di sintassi
- integers e unsigned integers: ridefiniscono gli interi per avere precisione arbitraria
- bit strings: stringhe di bit singolarmente identificati

Le uniche strutture dati supportate sono scalari e tabelle (array bidimensionali). Avremo 3 varianti sintattiche dell'OID:

- l'OID rappresenta in astratto la proprietà (nodo dell'albero): es. quando dico (1.3.6.1.2.1.1.5) sto intendendo la proprietà "nome dell'host"
- caso scalare: alla proprietà è associato un valore scalare (per questo computer qui, qual è valore associato a 1.3.6.1.2.1.1.5 ?) Per esprimere il concetto di VALORE della proprietà si aggiunge zero all'identificativo: quindi per interrogare il dispositivo, si scrive "nome proprietà.0", che sarà OID per letture/scritture dispositivo
- caso tabella: alla proprietà è associata una tabella, per indicarne un valore si aggiunge *.colonna.riga* per leggere valore di una cella. Es. se voglio sapere la terza proprietà della seconda interfaccia di rete (terza colonna, seconda riga) scrivo "nome proprietà.3.2" - che sarà OID per letture/scritture dispositivo, (non potremo mai leggere una tabella intera e farcela restituire, dovremo sempre ottenere uno scalare)

MIB notevoli sono:

- MIB-2: e' il sottoinsieme che inizia in 1.3.6.1.2.1; includeva tutti i dati essenziali agli apparati di rete. Oggi è un po' obsoleto, è stato modularizzato (diviso in TCP-MIB, UDP-MIB ,IP-MIB, IF-MIB; e i suoi pezzi delegati, così che possano essere più gestibili, riscrivendone solo una parte). Esempi a noi utili di managed object del MIB-2 sono
  - Il gruppo system 1.3.6.1.2.1.1, usato per memorizzare dentro l'apparato stesso
    - sysDescr(1), descrizione libera
    - sysContact(4), persona responsabile (email,telefono)
    - sysName(5), nome apparato
    - sysLocation(6), posizione apparato
  - Il gruppo IP 1.3.6.1.2.1.4, formato da 19 scalari + 4 tabelle che descrivono com'è configurato stack IP dell'apparato (es. qual è TTL di pacchetti prodotti); usato per memorizzare parametri generali del protocollo e tabelle con parametri specifici di ogni interfaccia, regola di routing, mappatura MAC. Es. tabella ifEntry (1.3.6.1.2.1.2.2.1), dove per ogni interfaccia sul sistema, ci sarà una riga con queste info (colonne della tabella)
    - ifindex(1) indice, numero ordinale dell'interfaccia
    - ifDescr(2) descrizione
    - ifType (3) tipo
    - ifMtu (4)

        - ifSpeed (5) velocità
        - ifPhysAddress (6) indirizzo MAC
        - ifOutOctets (16) quanti pacchetti sono usciti dall'interfaccia
- PEN (Private Enterprise Numbers): il sottoalbero su 1.3.6.1.4.1 è dedicato a moduli specifici richiesti da enti privati (possono essere richiesti gratuitamente, l'elenco è consultabile online). Due PEN sono importanti per monitoraggio sistemi operativi, entrambi estendono il MIB con oggetti generati dinamicamente:
    - UCD-SNMP (1.3.6.1.4.1.2021), sotto University of California Davids: in questo modulo hanno definito tutte le proprietà tipiche di un sistema operativo. Quindi tutta la parte di rete sta in MIB2; qui tutte le proprietà di un sistema (stato dischi, memoria, processi, carico, log).
    - NET-SNMP-EXTEND-MIB (1.3.6.1.4.1.8072) è l'output della direttiva extend, qui possono essere fatti comparire dinamicamente nuovi oggetti (OID generato sul momento) in base all'output di script di un managed object. Utile perché posso fare script che conta es. quante volte utente ha aperto un file; e poi dico ad SNMP di estendere questo MIB, ovvero dico al sistema di generare un nuovo oggetto (con nuovo MIB) che verrà posto sotto questo nodo di partenza (1.3.6.1.4.1.8072.xxxxxxx) ed essendo chiaramente uno scalare, accederò accodando al valore con 0

Modello di interazione e protocolli

I **managed object** sono le varie proprietà di un dispositivo, come finora descritte. Il dispositivo prende il nome di **network element (**l'oggetto fisico connesso alla rete); su questo è in esecuzione un **agent.** L'agent è un componente software o firmware che accede a memoria e registri dei dispositivi fisici per renderne visibili i contenuti come managed object; fa da interfaccia tra protocollo SNMP e strato di configurazione/funzionamento del network element, quindi espone via rete una possibilità di comunicare con linguaggio standard. Il componente che si occupa di interrogare gli agent (e organizzare informazioni ricevute) è detto **manager**: è l'altra componente della comunicazione, fa tipicamente parte di un NMS (Network Management System) ma non necessariamente: noi in laboratorio useremo comandi che producono un risultato, senza essere integrati in un sofisticato sistema di monitoraggio che produce grafici, tabelle.

Ci sono somiglianze tra modello di interazione manager-agent e C/S: manager sarebbe client e agent sarebbe server, ma con risorse hw e numerosità invertite: su internet pensiamo a pochi server molto potenti, per servire tanti client leggeri; nel manager-agent è il contrario. UN solo manager (robusto per raccogliere info da molti dispositivi, di frequente, tenerle in memoria, elaborare statistiche, generare grafiche) da cui interrogo una grande quantità di agent (aggeggi molto semplici, poche migliaia di righe di codice implementato in firmware). Inoltre c'è anche differenza perché a volte l'agent prende iniziativa per comunicare con manager, ad es. quando c'è informazione critica.

Il modello di interazione base è a polling, e un giro di polling dura un certo tempo: se c'è un apparato critico per il quale non posso aspettare quel certo tempo, l'interazione deve essere fatta ad interrupt (nel gergo di SNMP, trap). Una **trap** è una notifica asincrona inviata da un agent senza aspettare di essere interrogato dal manager. Se il modello tipico a polling invece prevede che manager invia richiesta e riceve risposta (GET/SET), in SNMPv1 non è prevista risposta da parte del manager alla trap. E' prevista la possibilità di creare una sorta di proxy (il **master agent**), che si comporta come agent nei confronti di un manager e come manager nei confronti di altri agent: richieste a due strati sono utili per gestione di apparati distribuiti geograficamente; così che il

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut
numero di agent da interrogare per ogni giro di polling sia minore (suddivisi tra più master agent); va tenuto conto che oltre alla latenza del polling si aggiunge anche la latenza delle interrogazioni fatte dai master agent.

SNMP è un protocollo a livello applicativo, trasportato su UDP, con agent in ascolto su porta 161 e manager su porta 162 per ricevere le trap (rispettive versioni sicure basate su TLS, sono su porta 10161/10162). Ci sono più versioni (v1,v2,v2c,v3), noi useremo la 1 per semplicità anche se oggi tutti i dispositivi supportano la 3. Tutte le versioni sono accomunate dalla struttura del pacchetto PDU (Protocol Data Unit, diviso in: versione, community, PDU-type, request-id, error-status, error-index, variable bindings).

Ci sono più tipi di PDU (quelli sottolineati sono solo in v2 e v3):

- Le 4 possibili richieste che un manager può fare ad un agent
  - **GetRequest** richiesta base che un manager fa ad un agent per sapere quanto vale un OID; richiede il valore associato al managed object
  - **SetRequest** richiede di settare il valore associato ad un managed object (es. se sposto apparato da secondo a terzo piano, via SNMP vado a modificare la proprietà systemLocation)
  - **GetNextRequest** richiede di scoprire qual è OID del managed object successivo a quello specificato. Come faccio a sapere sul manager qual è identificativo di un oggetto che agent ha creato dinamicamente (perché è stato istruito così)? Managed non può certo chiedere tutto gli OID possibili.
    - **Dettagli**: Getnextrequest implementa la visita dell'albero: questa si può fare in diversi modi. A partire dalla radice, si può sapere quali nodi sono attaccati; poi scendo in un nodo e chiedo quali sono sotto questo; a questo punto posso
      1. o restare al primo livello e chiedere gli altri attaccati alla radice, scoprendo quali sono tutti i nodi del secondo prima di scendere e scoprire tutto il terzo;
      2. oppure scendere al primo nodo di questo livello e continuare a scendere finché non trovo una foglia, prima di risalire ed esplorare orizzontalmente
    - GetNextRequest permette di dire: caro agent, so che gestisci OID con ident. 1.23.12.3; qual è il prossimo? L'agent (che gestisce il proprio MIB) sa qual è l'oggetto successivo, e risponderà con qual è il prossimo nodo. A questo punto il manager può porre la stessa richiesta al nodo che gli è stato dato come risposta, scendendo in profondità o meno; finché non esiste più un next: allora manager avrà quindi scoperto tutti i nodi che fanno parte dell'albero.
  - **GetbulkRequest** da v2 in poi è versione ottimizzata di GetNextRequest: si evita di avere spedizione pacchetto -> elaborazione -> pacchetto di ritorno; con getbulk si richiede di recuperare tutti gli oggetti successivi a quello indicato, fino a riempire il pacchetto UDP.
- Poi ci sono le 4 risposte possibili dall'agent al manager:
  - **Trap** notifica asincrona dall'agent al manager
  - **InformRequest** da v2, come sopra ma con conferma di ricezione: richiedere risposta di conferma non è aggiunta da poco, specie perché SNMP è su UDP (non ci si accorge di perdita pacchetti). Ma non avrebbe senso pensare di tirare su connessione TCP: in

generale, ogni volta che abbiamo protocollo req-resp con dati piccoli non ha senso avere overhead degli handshake.

- **Response** è usata dall'agent per rispondere a Request (o dal manager per rispondere a trap/inform)
- **Report** solo v3, usata per comunicazione inter-engine, principalmente per segnalare problemi con elaborazione messaggi ricevuti

I protocolli v1 e v2c offrono il concetto di **community**, ovvero un' etichetta per stabilire il livello di fiducia tra manager e agent. Ci sono 3 livelli di autorizzazione: una che può solo leggere info (read-only), una che può fare le set (read-write), una autorizzata a ricevere le trap (trap). Ogni livello è identificato da una stringa che fa sia da nome che da password (es. se so che la read-write community di un agent si chiama *private*, questo è tutto ciò che mi serve per avere accesso R/W al suo MIB). Questo mette in evidenza una debolezza di sicurezza: se non usiamo versione su TLS di SNMP, i pacchetti sono in chiaro; quindi qualcuno sulla rete che può vedere i pacchetti, può scoprire la community per poi fare liberamente query SNMP. Soluzione per essere sicuri che l'infrastruttura di gestione non sia attaccata è o usare le versioni cifrate, o raddoppiare l'infrastruttura di rete per separare quella di gestione da quella operativa.

SNMPv1 è limitato a 32bit e ha gestione errori minimale; SNMPv2 aggiunge:

- nuovi data type a 64bit
- comandi PDU GetBulk e Inform (per gestione interi sottoalberi e trap con risposta)
- party-based security system, molto macchinoso e di scarsa adozione, tanto da portare alla creazione di due sotto-versioni:
  - SNMPv2c (c per community, è v2 senza party-based security, quindi stessa sicurezza di v1)
  - SNMPv2u (user based, tentativo di migliorare la sicurezza rispetto a v1 senza complicarlo, poi usato anche in v3).

SNMPv3 è una vera rivoluzione dello standard, in cui vecchi comportamenti sono ridefiniti con termini totalmente diversi (niente più "manager" e "agent", unificati come **entities**). Le entity sono

- un engine: componente software dedicato a invio/ricezione messaggi, estrarre i dati e gestire la sicurezza
- più application: implementano algoritmo che permette di confezionare la risposta giusta

Dal punto di vista della sicurezza, per gli utenti si segue il modello User-based Security Model (USM, in cui gli utenti sono autenticati con password protette da HMAC e il canale è cifrato). Per le informazioni il modello View-based Access Control Model (VACM) è a metà strada tra un DB e un sistema operativo: gli utenti sono mappati in gruppi (che permette gestione in modo semplice, aggiungendo nuovi utenti a gruppi gestiti) e le porzioni di MIB sono dette viste, permettendo la definizione di matrici di controllo accessi: queste possono dire che l'intero sottoalbero che si genera in un dato nodo è gestibile da un dato gruppo (sia in lettura che scrittura).

Comandi SNMP e info sui file di config (extra info):

- Lato manager
  - ***snmpget*** recupero di un solo oggetto
    - es. ***snmpget -v 1 -c public 192.168.56.203 .1.3.6.1.2.1.1.6.0***

- ***snmpset*** impostazione del valore di un oggetto
  - es. ***snmpset -v 1 -c supercom 192.168.56.203 .1.3.6.1.2.1.1.6.0 s "proprio qui"***
    - ***s*** indica stringa da inserire nella proprietà, altri type possibili sono
    - ***i*** integer, ***u*** unsigned, ***a*** ipaddress… vedi ***man snmpset***
- ***snmpwalk*** usa ricorsivamente la PDU *getNext* per navigare un intero sottoalbero del MIB
  - Fare *walk* di un OID per scalare (es. .1.3.6.1.2.1.1.6.0 sysLocation) ha lo stesso risultato di una *get*, MA costa molti più pacchetti (agent fa esplorazione dell'albero, invece di richiedere direttamente proprietà che ci interessa). Importante non usare una walk quando basta una get
- Questi comandi hanno molti parametri in comune, le man page di ognuno documentano solo le opzioni specifiche. ***man snmpcmd*** documenta quelle comuni, essenziali in ogni invocazione:
  - ***-v*** versione
  - ***-c*** community
  - indirizzo del network element
  - OID del managed object
  - Quindi linea base è ***snmpXXX -v 1 -c <nomeCommunity> <IPelement> <OID>***
    - Per *walk* e *get* basta community read-only
  - Nel man ci sono opzioni con ***-O*** utili per formattare output:
    - Opzione **-On** mostra gli OID in output in formato numerico (meno leggibile ma più processabile) invece che testuale
- Il file *etc/snmp/snmp.conf* contiene alcuni parametri per configurare il comportamento di default dei tool manager
  - commentando la riga "*mibs :* " abilitiamo l'uso dei MIBs (essendo questi disabilitati di default per questioni di licenza)

- Lato agent
  - Gli agent possono essere in firmware, ma noi vedremo come funziona un agent software (che gira all'intero del SO). In questo caso l'agent è il demone ***snmpd***
  - ***systemctl start snmpd*** lancia il demone secondo la configurazione attuale
  - Il file *etc/snmp/snmp**d**.conf* configura l'agent:
    - sostituendo *agentAddress udp:127.0.0.1:161* con *agentAddress udp:161* poniamo agent in ascolto su porta 161 per qualsiasi indirizzo (?) (è l'indirizzo su cui si mette in ascolto l'agent (può essere elenco di indirizzi) agentaddress 127.0.0.1,[::1] indica sia IPv4 che IPv6 che l'agent ascolta su localhost. A default agent ascolta traffico su porta 161 da qualsiasi interfaccia).

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut

- direttive *rocommunity public* e *rwcommunity supercom* definiscono 2 community separate, una per la lettura e una per la scrittura: comm di lettura si chiama public, comm di scrittura si chiama supercom
- può settare alcuni managed object di sistema (es. *sysLocation*, *sysContact*): **questi oggetti sono RW SOLO SE non sono settati nel file di config! Altrimenti read-only.**
- È possibile inserire direttive di monitoraggio dei parametri base (carico macchina, uso dei dischi, presenza di processi), estensione UCD-SNMP doc at http://net-snmp.sourceforge.net/docs/mibs/ucdavis.html
  - ***load [max-1] [max-5] [max-15]***
    - tabella .1.3.6.1.4.1.2021.10
    - i tre valori indicano il massimo tollerabile negli ultimi 1-5-15 min: tre righe (carico negli ultimi 1-5-15- min); le colonne hanno il carico effettivo, e il flag di superamento delle rispettive soglie.
      - scrivendo ***load 0.1*** nel file conf e riavviando il demone, all'interno della tabella indicata (.1.3.6.1.4.1.2021.10) dovrebbe comparire lo stato del carico di sistema (si indica che il limite è 0 nell'ultimo minuto).
      - *walk* su OID della tabella(***snmpwalk -v 1 -c public 192.168.56.203 .1.3.6.1.4.1.2021.10 | less***) restituisce tutti gli elementi come lista unidimensionale, come se fosse una tabella serializzata. Vediamo output con OID che si ripetono, e non finiscono in 0: i nomi prima degli OID numerici sono i nomi delle colonne.
      - la prima colonna contiene il numero delle righe (sono 3 perché load offre 3 risultati, una riga per 1, una riga per 5, una riga per 15)

UCD-SNMP-MIB::laErrorFlag.1 = INTEGER: error(1)

      - flag di errore viene settato perché abbiamo superato la soglia di 0.1 nell'ultimo minuto (carico simulato con find /)
      - Leggere il flag permette di usare una definizione più astratta (il manager non deve sapere quanto è il limite)
      - per usi in script, utile OID *UCD-SNMP-MIB::laLoadInt.1 = INTEGER: 30* che fornisce intero già moltiplicato per 100, ci toglie di mezzo il .
  - ***disk [partizione] [minfree|minfree%]***
    - tabella .1.3.6.1.4.1.2021.9
    - una riga per ogni partizione messa sotto controllo da una direttiva disk
    - colonne hanno tutti i dettagli della partizione e flag di spazio sotto il minimo
  - ***proc [nomeprocesso] [maxnum [minnum]]***
    - tabella .1.3.6.1.4.1.2021.2
    - una riga per ogni processo messo sotto controllo da una direttiva proc

- colonne hanno numero di istanze, flag di superamento delle soglie
  - non specificando minimo e massimo si hanno default di 1 – illimitato
  - es. aggiungiamo ***proc sshd*** al file di configurazione: facendo walk della tabella indicata, si ha una tabella con una sola riga (perché abbiamo messo sotto controllo un solo processo).
    - *UCD-SNMP-MIB::prCount.1 = INTEGER: 3* è il numero di processi sshd running, flag errore non è settato essendo sotto la soglia max (non avendola specificata, è illimitata)
    - POI se aggiungiamo riga proc atd e modifichimo la prima in proc ssh 2 1 la tabella diventa di due righe (una per ogni processo) e otteniamo flag settato *UCD-SNMP-MIB::prErrMessage.1 = STRING: Too many sshd running (# = 3)* perché minimo è 1 e massimo è 2

- È possibile inserire direttive per eseguire codice il cui output è reso accessibile come managed object. Estensione NET-EXTEND: doc at http://www.oidview.com/mibs/8072/NET-SNMP-EXTEND-MIB.html
  - tabella NET-SNMP-EXTEND-MIB:: .1.3.6.1.4.1.8072
    - ***snmpwalk -v 1 -c public 192.168.56.203 .1.3.6.1.4.1.8072 | less*** per visualizzare intera tabella
  - righe con nome = etichetta della direttiva *extend-sh* etichetta di una riga sarà il nome simbolico che ho dato alla mia direttiva extend
  - diverse colonne, la più comune è ***nsExtendOutputFull*** mostra l'output intero dello script associato alla direttiva
    - ***snmpwalk -v 1 -c public 192.168.56.203 .1.3.6.1.4.1.8072 | grep "nsExtendOutputFull" | less***
  - formato sarà ***extend-sh <etichetta> <riga di comando>*** (ovviamente riga di comando potrà avere pipe e quant'altro)
  - es. aggiungiamo ***extend-sh test1 echo HelloWorld*** al file conf. OID corrispondente sarà *NET-SNMP-EXTEND-MIB::nsExtendOutputFull."test1"* **(notare i doppi apici, indicano all'agent che è un nome da risolvere, non un segmento di OID standard – attenzione all'espansione bash, devono arrivare al comando snmp!). Importante usare path assoluto per i comandi inseriti**
  - Quando agent riceve una *getRequest* all'OID corrispondente, esegue il comando e restituisce l'output nella Response
  - **Per eseguire comandi privilegiati**, dovremo autorizzare utente dell'agent (nelle VM 9CFU utente si chiama *snmp*) nel file sudoers, ad eseguire quello specifico comando.
    - Con ***visudo*** aggiungiamo al file /etc/sudoers riga

snmp ALL=NOPASSWD:/absolutePathComando -opzioni

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut

- nel file /etc/snmp/snmpd.conf, direttiva sarà

extend-sh <etichetta> /usr/bin/sudo <riga di comando>

- es. sull'agent
  - file /etc/sudoers snmp ALL=NOPASSWD:/sbin/iptables -vnL
    - si può testare con
      - # las@Router:~$ sudo -H -u snmp /bin/bash

snmp@Router:/home/las$ sudo /sbin/iptables -vnL

  - file /etc/snmp/snmpd.conf extend-sh ipt /usr/bin/sudo /sbin/iptables -vnL

DETTAGLI: SNMP non è molto comprensibile, è efficiente: risale a quando i byte avevano un alto costo. Oggi nuovi standard, come YANG, sono indipendenti da protocolli di trasporto, offrendo strutture dati definite in stile XML. Ci sono nuovi protocolli che introducono 2 grandi novità:

1) transazionali: modifiche da remoto garantendo o che sono riuscito a fare tutto quello che serve, o che non è cambiato niente

2) non più comunicazioni C/S, ma si usano piattaforme di comunicazione per realizzare modelli pub/sub; con maggiore flessibilità per chi vuole ricevere date notifiche (divise per argomento e contenuto, non per topologia dell'apparato)

LDAP

Per configurare un sistema sono necessari diversi file di configurazione, e in una rete con molte macchine sarebbe necessario cambiare centinaia di file per compiere coerentemente operazioni come l'aggiunta di un utente, di un host; la modifica di un parametro di un demone. Da qui nasce l'esigenza di condividere queste informazioni in maniera centralizzata superando l'approccio a file: LDAP è una directory centralizzata dove si possono conservare informazioni che servono a tutti i calcolatori della rete. ~~(Abbiamo già visto Vagrant e Ansible per configuration as code, ma essendo pensati per essere usati in momenti specifici del ciclo di vita di una VM -si pensi a boot e provisioning- non tutti i parametri sono impostabili con questi: ad esempio risoluzione dei nomi e lista utenti, va bene che siano impostate già al provisioning; ma se poi c'è bisogno di creare un sistema multiutente dove vanno modificati molte volte in un giorno, ci vuole un protocollo in tempo reale, non basta il provisioning).~~

~~Nel corso abbiamo visto un minimo di sistemi di config. Management, come Vagrant e Ansible, pensati per fornire modelli in grado di descrivere lo stato di configurazione per un determinato ruolo -VM, servizio...- e di istanziare il modello sulla base delle specifiche del target (hostname, ip, layout dischi);~~ LDAP è in grado di fornire sia servizi di templating e configuration management (config management ovvero fornire modelli per descrivere stato di configurazione per un ruolo -VM, servizio...-, e istanziazione del modello sulla base delle specifiche del target -hostname, ip, layout dischi...-), che autenticazione centralizzata.

Per fornire autenticazione centralizzata è necessario specificare due aspetti:

- Come scegliere localmente la sorgente dei dati:
  - NSS è un sistema generale per selezionare i servizi di nomi, abbiamo già visto che permette di scegliere quale fonte usare per individuare concretamente un DB indicato come astratto. NSS si aspetta di trovare libnss-ldap.so per sapere come interfacciarsi con

LDAP: è una libreria che fa da commutatore, sa trovare le implementazioni concrete che servono, grazie a file di configurazione di NSS.

- **PAM** è un sistema modulare per implementare *authentication modules*. Permette di implementare in modo flessibile le politiche di autenticazione (es. interfacciamento con lettore impronte digitali, scrittura di algoritmo complesso per valutare molti parametri sull'utente da autenticare...), con un sistema simile ad NSS: ha un commutatore che va a cercare le implementazioni dei metodi astratti descritti in un file di configurazione (esempio: *account required pam_nologin.so* lo shared object implementa in binario la funzione necessaria a gestire quel caso). Le librerie modulari possono essere usate in maniera flessibile dalle applicazioni (da una parte ho librerie che implementano tutte le politiche, dall'altra le app che ne hanno bisogno; in mezzo c'è file di configurazione che dice quale libreria usare per una data applicazione). C'è una libreria libpam-ldap.so che permette a PAM di interfacciarsi con LDAP.

- Come trasferire i dati dal server centrale ai client
  - LDAP è il protocollo di distribuzione delle informazioni. Attraverso NSS ottengo una lista di utenti in modo configurabile, poi PAM fornisce i metodi per l'autenticazione: per LDAP sarà necessario che funzionino entrambi, e che ci sia lo strato intermedio delle librerie libnss-ldap (per usare db remoto LDAP come sorgente di nomi utenti, gruppi, host…) e libpam-ldap (per accedere a db remoto LDAP per autenticare utenti e ricavare regole di autenticazione).

LDAP implementa un directory service, ovvero un database specializzato basato sul modello dell'elenco telefonico: permette di cercare informazioni in base ad attributi, ma è importante che sia specializzato e non generico perché

- fornisce grande velocità di lettura ma minore velocità di scrittura. Come per elenco del telefono, si prevedono molte letture e poche scritture (es. in azienda, ogni mese assumo 1 persona, ma ogni giorno 1000 fanno accesso e devono autenticarsi).
- E' un DB gerarchico, al suo interno è diviso in diverse parti (es. ogni nazione ha il suo namespace (o anche ogni regione, in italia)). La gerarchia ha pregi e difetti:
  - Pregi:
    - se ho modo semplice di immaginare la posizione dell'informazione; questa diventa criterio di ricerca iperefficiente; navigare una gerarchia è semplicissimo.
    - molto facile creare delle deleghe nella gestione di sottoalberi: nessun problema di collisione, i nomi devono essere univoci globalmente solo nel loro livello di sottoalbero.
  - Svantaggi:
    - una volta individuata la struttura della gerarchia, è difficile cambiarla: si ha una visione rigida, con il tradeoff dello scegliere a priori la vista che mi interessa, e avere una ricerca meno efficiente se ce ne interessa un'altra dopo.
- Non è relazionale, si possono evitare le transazioni e si può usare meccanismo di locking semplice, perché le scritture sono rare e quindi che ce ne siano due nello stesso momento è difficile; un'ipotetica collisione data da modifica dello stesso dato, andrà gestita a mano, il protocollo non se ne occupa.

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut

- Deve essere altamente disponibile e quindi poter essere facilmente replicato; avere molte copie è importante.

LDAP esiste da 30 anni, usato su Internet per i sistemi UNIX-like (prima UNIX era fortemente multiutente, poi è diventato principalmente server) e da quando Windows è diventato il principale sistema client per reti professionali, ha implementato una sua versione di LDAP sotto Active Directory.

Modello dei dati

LDAP non si preoccupa di come vogliamo memorizzare la base di dati (noSQL, SQL, file di dati complessivo, un file per ogni record…) la cosa importante è che ci sia un modulo che LDAP possa usare per accedere fisicamente ai dati, **DBI** (**DataBase Interface**) è un'interfaccia per accedere al backend più adatto (Si pone tra LDAP Server e driver file/SQL...). DBI gestisce una serie di **entry**, anche dette **object**; ogni entry ha una collezione di **attribute**, ognuno dei quali ha un **type** e può avere diversi **value** (diversamente da un DB tradizionale, dove ogni cella ha un solo valore; in DBI, una entry che rappresenta un utente che può avere attributo "telefono", può avere più valori per questo attributo). Ogni entry ha un identificativo gerarchico detto **DN** (**Distinguished Name**)(gerachico, come gli OID per SNMP).

Le entry sono legate in gerarchia, il DN dice quali rami seguire per arrivare al nodo che mi interessa. La struttura risultante è chiamata **DIT** (**Dir Information Tree**) e rappresenta il modo in cui viene esposta la base dati gestita dalla DBI: questa dice come accedere alle entry (che possono essere in qualsiasi formato); ma dal lato di chi usa LDAP le entry risultano ordinatamente organizzate in un DIT. Quindi i dati possono avere una rappresentazione fisica arbitraria (es. anche come file flat, oppure su disco i dati sono visti come blocchi); ma da un punto di vista logico, mediante LDAP si possono vedere come ordinati gerarchicamente, come un albero.

Replica

Essendo un servizio essenziale, LDAP prevede nativamente meccanismi per l'alta disponibilità. Implementare la ridondanza è semplice, siccome scritture sono rare, (e quelle incompatibili eccezionali). Il modello più usato è quello multimaster, in cui si copia il DB su vari server e si permette ad ognuno sia accesso in lettura che in scrittura; con protocolli che controllano periodicamente le inconsistenze tra i server e valutano se propagare modifiche da uno all'altro, o come gestire manualmente le inconsistenze. L'alternativa è il modello master-replica, in cui c'è un solo server che accetta scritture dai client e molti server replica che offrono solo lettura (ha senso avere solo letture sulle tante repliche, essendo l'operazione che è più necessario scali). I modelli master-replica possono essere anche by referral (in cui il client diventa più complicato, perché prova a scrivere su un server, e se incontra replica read-only; riceve informazioni sul master a cui rivolgersi) e by chain (in cui il client è il più semplice possibile, e i server sono più complessi: i replica fanno finta di essere scrivibili ma propagano richiesta al master, facendo da proxy).

Sezionare un DIT è vantaggioso (ad esempio in termini di prestazioni, per la località delle risorse rispetto a sedi geograficamente distanti, o per delega amministrativa per sezioni di un'azienda) e avere una struttura ad albero rende facile distribuire i dati, ponendo i sottoalberi in diverse posizioni. La distibuzione può essere fatta mediante foglie LDAP che sembrano entry normali, ma in realtà contengono puntatori alla posizione di un sottoalbero:

- dal lato del server principale, un nodo link (detto reference) permette di sapere a quale altro server subordinato rivolgersi per scendere nella gerarchia e avere i dati del sottoalbero non locale; (Subordinate knowledge link)
- dal lato del server che ottiene la delega, il nodo link contiene l'indicazione del server a cui riferirsi per la parte superiore dell'albero: così il server subordinato sa che ha una sua radice

ma che non è assoluta; ed in realtà c'è un genitore a cui può rivolgersi per conoscere il resto del DIT. (Superior knowledge link)

Formato entry

Una entry è una collezione di attributi. Questi non sono definiti con un nome come nei linguaggi di programmazione, ma si dichiara direttamente il valore etichettato dal tipo (In C: int i = 1 dice tipo, nome e valore; in LDAP variabili sono come "int 1", coppia tipo-valore). Tra gli attributi, due sono presenti in tutte le entry: ***dn*** (distinguished name) è nome univoco della entry, ***objectClass*** (una o più) è la classe a cui appartiene la entry. Esempio entry:

dn: dc=labammsis DN è "dc=labammsis", perché c'è attributo dc: labammsis

objectClass: dcObject

objectClass: organization

dc: labammsis

o: università

Le parti blu sono i tipi di attributo (etichette che dicono il tipo), la parte in verde è il valore dell'attributo. Alcuni attributi sono univoci (es. dn deve essere univoco), altri possono essere multipli (es. attributo di tipo objectClass ha 2 valori, o se preferiamo ci sono due attributi dello stesso tipo). Questo formato è usato per scambiare entry tra client e server e viene detto **LDIF** (LDAP Interchange Format), ogni entry è sequenza di righe, e in ogni riga c'è "tipo:valore" dei diversi valori; per mettere il DB in un solo file LDIF, basta scrivere ogni entry così e separarle con una riga vuota.

Lo **schema** garantisce che ogni entry inserita sia ben formata e che tutti gli utilizzatori sappiano come vedere l'insieme delle entry (possano vedere l'organizzazione del DB per interpretare correttamente le informazioni); mediante un insieme di regole che descrivono i dati immagazzinati, usando due definizioni: le ***objectClass*** (una o più per entry, specifica i tipi di attributo obbligatori e facoltativi -e per esclusione, quelli vietati-) e gli ***attributeType*** (definiscono i tipi di dato e le regole per compararli).

Lo schema è composto da tante definizioni di classi di oggetti; le objectClass, per definire cosa è valido in una data classe, usano le definizioni dei tipi di attributo. Gli attributeType, per essere definiti, si avvalgono della sintassi (es. attributeType numero di telefono, potrà dire ha sintassi di stringa di max 12 caratteri). In LDAP; qualcuno ha definito quali sintassi esistono (volendo un esempio sono come i tipi di dato di base, interi stringhe float booleani etc.).

Quanto appena visto è lo schema, che dice solo cosa è valido e cosa no, mentre il contenuto effettivo deve essere modellato seguendo lo schema: ogni attributo avrà un valore, che deve essere compatibile con la sintassi (es. non posso assegnare valore di 15 caratteri ad attributo che prevede sintassi di 12 caratteri), quindi i valori devono rispettare i vincoli degli attributiType. Ogni attributo deve rispettare i vincoli di numerosità imposti dall'objectClass di ogni entry (quali attributi devono/possono essere presenti). Le objectClass seguono le definizioni dello schema del DIT.

Spazio dei nomi

Per ogni entry è necessario scegliere un attributo rappresentativo per individuarla. C'è libertà di scegliere quale attributo darà nome alla entry; nell'esempio di prima si è deciso che attributo *dc* con valore *labammsis* viene usato come nome: quindi il nome (***dn*** , distinguished name) è

dc=labammsis perché tra gli attributi della entry c'è un attributo *dc* che vale *labammsis*; in alternativa il nome potrebbe essere es. *o=università* perché c'è attributo *o* con valore *università*. Essendo la scelta del nome arbitraria, in teoria si può scegliere che le entry siano legate

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut
ad un nodo padre mediante nomi relativi diversi (es. alcune di nome relativo o=qualcosa e altre con nome relativo dc=qualcosa), mischiandole, ma dopo un po' non si capisce più niente: è opportuno seguire linee guida per costruire una gerarchia sensata.

La stringa *tipoAttributo=valoreAttributo* scelta è il **relative distinguished name** (**RDN**) della entry (quello autocontenuto che non mi dice dov'è la entry; poi devo scegliere a quale nodo già esistente agganciarla -con il BDN-); il nome del nodo a cui agganciare la entry è il **base distinguished name** (**BDN**). Il **distinguished name** (**DN**) è il nome univoco globalmente ottenuto concatendando i due, quindi DN=RDN+BDN.

Nella radice, DN e RDN coincidono. Supponiamo che voglia suddividere albero con *organizational unit*, una per docenti e una per studenti: un nodo ha come RDN ou=docenti, un nodo ha RDN ou=studenti; essendo agganciati a quello padre; il DN (nome univoco che individua i nodi nell'albero complessivo) è ou=docenti,dc=unibo . Nel DN l'ordine dei termini è foglia->ramo->radice, da sinistra a destra (quindi top level domain è a destra, come per il DNS; l'opposto del filesystem dove la radice è la prima cosa a sinistra). Il terzo livello avrà RDN cn=mario (c'è docente mario e studente mario): si possono avere più nodi con lo stesso RDN solo se hanno BDN diversi, essendo il DN complessivo di ognuno a dover essere univoco.

Quindi ogni RDN deve essere localmente unico (tutte le entry connesse allo stesso BDN, quindi univocità nel proprio livello). Nel caso in cui un singolo attributo non sia sufficiente per avere RDN unico (ad esempio, *cn common name* può avere omonimie) si può:

- Introdurre id univoco, soluzione "da DB":
  - si può effettivamente aggiungere un attributo come uniqueIdentifier, ma l'idea di LDAP è che gli attributi rappresentino qualcosa di proprio della entry (e un identificativo progressivo non lo è). Così facendo salta il discorso della facilità di navigazione della gerarchia, per ricerca efficiente. (esempio, per trovare un docente, associare un id univoco fa sì che si debba fare ricerca dell'id: per trovare docente prima scendo nella sua università, poi nel settore in cui insegna, ma se poi invece del nome devo sapere il suo numero di serie; tanto valeva fare ricerca globale)
- Concatenare più coppie attributo-valore:
  - Soluzione adottata da LDAP, si concatenano con + più coppie attributo=valore per ottenere RDN localmente univoco (esempio, cn=Marco+sn=Prandini come RDN, per evitare omonimie su *cn* nome, si concatena con *sn* cognome, poi ok possono esserci omonimie nome-cognome ma è un esempio)

Attributi

Un *attributeType* è simile a un tipo di un linguaggio di programmazione, ma a differenza di questi, specifica separatamente:

- **SYNTAX** ,sintassi: Il vero e proprio tipo di dato nativo (documentazione sintassi https://tools.ietf.org/html/rfc4517#section-3)
- Matching rules: Le regole per stabilire i criteri di confronto tra valori, per valutare uguaglianza e ordinamento nelle ricerche. In un linguaggio di programmazione, se ad es. dico che una variabile a è di tipo intero, per fare confronti verrà implementata una logica cablata all'interno del concetto di tipo intero. In LDAP per ogni tipo di dato potrei avere diversi modi di interpretare confronti logici: (doc https://tools.ietf.org/html/rfc4517#section-4)

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut

- **ORDERING** ,ordinamento: es. stringa che rappresenta un numero, se vale 100 è maggiore o minore di 2? dipende se interpreto con criterio lessicografico o numerico
- **EQUALITY** : es. per un tipo attributo che è una stringa, si può dire che il confronto tra stringhe di quel tipo deve essere fatto ignorando maiusc e minusc (come per le email, case insensitive). Per attributo email in una entry, vedrò che attributeType email è definito come sintassi stringa, e con EQUALITY caseInsensitiveMatch (che vuol dire, quando fai il confronto con altro valore, ignora maiusc e minusc). Per confronto tra password, sarebbe caseSensitiveMatch
- **SUBSTRING**: potrei definire che il criterio di confronto è la presenza di una sottostringa (es. in un'azienda con centralino e numero di interno, considero match valido le prime 6 cifre del numero di telefono, perché vuol dire che sono su numeri interni)

- Vincoli d'uso: **SINGLE-VALUE** ammette un solo valore di questo tipo nella entry (es. ***dn*** è single-value), ce ne sono altri come NO-USER-MODIFICATION, USAGE...
- Eventuali dipendenze gerarchiche: con **SUP** <*altro attributeType*> il tipo eredita tutte le proprietà del superiore, può ridefinirne alcune, e nelle ricerche per un tipo superiore vengono restituiti tutti i valori dei tipi basati su quello (es. *cellulare* ha come tipo superiore l'attributeType *telefono* (che è una stringa, ha ordinamento, etc) Quando creo una entry dell'utente marco, se ci metto dentro un *telefono* e un *cellulare*, se qualcuno cerca "*telefono*" verrà fuori sia *telefono* che *cellulare*: questo permette di definire in modo ordinato, separatamente, 1 telefono, 1 cellulare e 1 numero di casa; ma di ottenere comunque tutti e tre i risultati se si richiede "*telefono*", quando non si sa quale si vuole specificamente).

Esistono molti attributeType già definiti, standard per gli usi più comuni (doc. https://oav.net/mirrors/LDAP-ObjectClasses.html sono a questo link, sezione Attributes). Tra i più comuni abbiamo

- ***cn*** ***common name***,
- ***dc*** ***domain component***,
- ***o*** ***organization***,
- ***c*** ***country***

Per definire un nuovo attributeType, si inserisce la sua descrizione nella config. del server, sotto forma di attributo di una entry speciale (quasi mai si ridefinisce un nuovo attributeType, essendo già diffusi molti attributeType standard):

olcAttributeTypes: (1000.1.1.1 NAME ('fn' 'filename') → due nomi alias

DESC 'nome del file'

EQUALITY caseExactMatch

SUBSTR caseExactSubstringsMatch

SYNTAX 1.3.6.1.4.1.1466.115.121.1.15) → questo è come scrivere "string": per definire sintassi, regole di matching, si usano gli OID

Se vogliamo creare attributi nuovi, dobbiamo usare come nome un OID univoco (es. 1000 non esiste di sicuro, essendo la radice OID 0 o 1 o 2)

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut

Classi

Lo scopo essenziale delle *objectClass* è di indicare in una certa entry quali attributi sono obbligatori (MUST) e quali facoltativi (MAY): quando si va a compilare una entry, si è obbligati a riempire attributi MUST, ed è vietato riempire attributi che non siano MUST o MAY. Le classi possono essere di tre tipi:

- ABSTRACT servono per creare tassonomia di oggetti, ma sono troppo astratte per originare entry (non si possono istanziare)
    - es. top è classe predefinita, astratta, che contiene solo distinguished name e object class (dato che tutte le entry dovranno contenerli). Poi classe astratta animale/vegetale aggiunge solo numero di cromosomi (e ancora non è istanziabile).
- STRUCTURAL servono per descrivere categorie di oggetti concreti
    - es. aggiungendo nome, razza e peso si possono specificare classi *cane* e *gatto* (strutturali, istanziabili, che ereditano gli attributi delle classi da cui discendono)
- AUXILIARY possono descrivere collezioni di attributi aggiuntivi che arricchiscono una entry, senza essere collegati a categorie specifiche di oggetti
    - *fascicoloSanitario* può estendere alcune entry di *cane:* per definire un'entry di classe *cane* dovrò/potrò specificare nome, razza,peso; poi potrò dire che una entry *cane* ha ANCHE objectClass *fascicoloSanitario*, e questo mi permetterà/obbligherà (may/must) di inserire anche i dati aggiuntivi.
    - Permette di specificare, ad esempio, una classe ausiliaria che prevede attributo *colore;* senza bisogno di creare due classi separate *AutoConColore*/*AutoSenzaColore.*

Ogni entry **dovrà** avere UNA SOLA classe strutturale, e potrà avere più classi ausiliarie (es. marco prandini ha objectClass strutturale *docente*; potrò aggiungere classe ausiliaria *rettore* (con informazioni aggiuntive, ognuna di esse in MUST o MAY. La classe ausiliaria dovrà comunque essere definita, e non potrà essere usata da sola).

Le classi ausiliarie quindi permettono di creare un modello con una sorta di ereditarietà multipla:

- Se in Java potrei avere il problema di 2 classi che definiscono attributo A con tipi diversi, (una come intero, una come stringa. Quando istanzio oggetto che è istanza contemporaneamente delle due classi, A è intero o stringa?) in LDAP invece attributeType "nome" di classi *docente* e *rettore* non è definito nella entry, ma ESTERNAMENTE, nello schema: quando mi riferisco ad un certo tipo, e' per forza QUEL tipo.
- E' come dire che in JAVA, tutte le variabili che si chiamano (con il nome) A hanno lo stesso tipo. Questo perché in LDAP le variabili non hanno nome, appunto, ma solo tipo. Ogni objectClass specifica attributeType che devono essere definiti nello stesso schema, non ci sono quindi conflitti se più objectClass in una entry menzionano lo stesso attributeType.
- L'effetto è che la entry che usa le diverse classi deve contenere tutti gli attributi MUST di tutte le sue classi, e può contenere tutti gli attributi MAY. In caso di vincoli diversi per lo stesso attributeType nelle diverse classi, vince il più stringente (un attributo sia nell'insieme MUST che MAY, deve essere presente).

Le classi possono essere definite gerarchicamente usando **SUP** <*altra objectClass*> nella definizione. La classe inferiore eredita tutti gli attributi MUST e MAY delle superiori e può solo aggiungerne (e non toglierne) altri.

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut
Esistono molte objectClass già definite, standard per gli usi più comuni (doc. https://oav.net/mirrors/LDAP-ObjectClasses.html). Tra le più comuni

- ***organization***
- ***person***
- ***device***

Per definire una nuova objectClass , si inserisce la sua descrizione nella config. del server, sotto forma di attributo di una entry speciale:

olcObjectClasses: (1000.2.1.1 NAME ('dir') → per definire nomi univoci, si usano gli OID

DESC 'una directory'

MUST fn

MAY fs

AUXILIARY)

In LDAP, tutte le direttive di configurazione sono attributi in entry di un albero separato da quello dei dati veri e propri:

Per riconfigurare un server LDAP si inserisce quindi una entry in formato LDIF, es. per un nuovo schema si può creare il file *filesystem.ldif :*

```
dn: cn=filesystem,cn=schema,cn=config
objectClass: olcSchemaConfig
cn: filesystem
olcAttributeTypes: ( 1000.1.1.1 NAME ( 'fn' 'filename' )
  DESC 'nome del file'
  EQUALITY caseExactMatch
 SUBSTR caseExactSubstringsMatch
 SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
olcAttributeTypes: ( 1000.1.1.2 NAME ( 'fs' 'filesize' )
  DESC 'dimensioni del file'
  EQUALITY integerMatch
  ORDERING integerOrderingMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 )
olcObjectClasses: ( 1000.2.1.1 NAME 'dir'
  DESC 'una directory'
  MUST fn
  MAY fs
  AUXILIARY )
olcObjectClasses: ( 1000.2.1.2 NAME 'file'
```

Version: 1.0
First draft on Sat 5 March 2022, 10:12:58 author: jgabaut
Last edit on Tue 12 July 2022, 10:37:02 author: jgabaut

DESC 'un file'

MUST (fn $ fs)

AUXILIARY)

Solitamente si usa un comando apposito per scavalcare autenticazione LDAP e agire da amministratori locali (admin del servizio): ***ldapadd -Y EXTERNAL -H ldapi:/// -f filesystem.ldif***

Protocollo e operazioni offerte

Il protocollo LDAP è a livello applicativo e si basa su TCP a liv. Trasporto (porta 389 / 636 over TLS), si basa su una sessione (viene autenticato l'utente che vuole effettuare la connessione). Alla *bind* (che riceve risposta positiva o negativa in base al successo dell'autenticazione); segue una serie di richieste di operazione, tutte effettuate a nome dell'utente autenticato, ognuna di queste può essere permessa o meno in base al sistema di controllo degli accessi (in LDAP per ogni entry, per ogni attributo… si possono specificare permessi per ogni utente). Alla fine delle operazioni c'è l'unbind.

Quindi per ogni operazione sarà necessario specificare un ***bind DN*** : equivale all'utente con cui autenticarsi sul server LDAP, la bind dirà alla directory chi è l'utente e la directory deciderà se l'utente ha diritto o meno a fare l'operazione.

Le operazioni offerte sono:

- Search
  - La ricerca deve specificare: come tutte le operazioni un **bind DN**, poi
    - Un ***base DN*** : il punto del DIT da cui iniziare la ricerca (radice da cui cercare)
    - Uno ***scope*** : a partire da *base DN*, indica quanto estendere la ricerca, con tre valori possibili
      - ***sub*** default, indica l'intero sottoalbero
      - ***one*** solo figli diretti del *base DN* (un solo livello in basso)
      - ***base*** solo il nodo base
    - Eventualmente un **filtro** (base e scope obbligatori, filtro è facoltativo): per ricercare in base a contenuto della entry invece che per posizione, usando espressioni logiche in notazione prefissa (operatore precede operandi)
  - Sintassi filtri ed esempi: (operatore(operand1)(operand2)) eventualmente guarda slide
    - <filter>::='('<filtercomp>')'
      - <filtercomp>::=<and>|<or>|<not>|<item>
        - <and>::='&'<filterlist>
        - <or>::='|'<filterlist>
        - <not>::='!'<filter>
          - <filterlist>::=<filter>|<filter><filterlist>
        - <item>::=<simple>|<present>|<substring>
          - <simple>::=<attr><filtertype><value>

- <filtertype>::=<equal>|<approx>|<greater>|<less>
  - <equal>::='='
  - <approx>::='~='
  - <greater>::='>='
  - <less>::='<='
- <present>::=<attr>'=*'
- <substring>::=<attr>'='<initial><any><final>
  - <initial>::=NULL|<value>
  - <any>::='*'<starval>
    - <starval>::=NULL|<value>'*'<starval>
  - <final>::=NULL|<value>

- (cn=Babs Jensen) tutte le entry nello scope, che hanno come attributo cn la string "Babs Jensen"
- (!(cn=Tim Howes)) operatore "diverso da"
- (o= univ*of*mich*) * wildcard, qualsiasi stringa
- (&(objectClass=Person)(|(sn=Jensen)(cn=Babs J*))) due operandi in and (operatore &), ognuno tra parentesi: il primo è valore objectClass uguale a person; il secondo è nuovamente una composizione, ma in or (operatore |), il primo sn uguale a jensen, il secondo cn uguale a babs j* , dove * è qualsiasi stringa
  - Abbiamo quindi possibilità di comporre più espressioni, ponendole come operandi di altre
- (&(|(uid=jack)(uid=jill))(objectclass=posix Account) risultato finale è unica riga con la notazione prefissa

- comando per ricerca: **ldapsearch -x -b dc=labammsis [-s base | one | sub] [filtro]**
  - **-s** indica lo scope
  - ***-b*** indica base DN, punto di partenza della ricerca
  - opzione -h indica server a cui connettersi
    - ldapsearch -x -h 192.168.56.203 -b "dc=labammsis" -s sub 'cn=marco'
  - -x attiva autenticazione base

- Compare
- Add
  - comando per aggiunta: **ldapadd -x -D "cn=admin,dc=labammsis" -w admin [-f file_ldif_da_inserire]**
  - **-f** specifica il file, se omesso usa stdin
  - -D attiva query autenticate, fornendo credenziali amministratore di LDAP:

        - -D <utente> ("cn=admin,dc=labammsis")
        - -w <password> ("admin")
        - Si deve sempre specificare almeno una classe strutturale nell'LDIF !!!

- Modify
    - comando per modifica **ldapmodify**
    - Stessi parametri di ldapadd, ha più casi: si può usare per aggiunta/modifica/rimozione attributo
    - si usa un LDIF con attributo *changetype: modify* seguito da *add* | *replace* | *delete* che specificano su quale attributo agire, e poi
    - Esempi nel prontuario sul sito del corso...
- Delete
    - comando per rimozione **ldapdelete -x -D "cn=admin,dc=labammsis" -w admin *DN***
        - DN è distinguished name della entry da eliminare
- ModifyRDN
- Configurazione server:
    - ***ldapsearch -Y EXTERNAL -H ldapi:/// -b "cn=config"*** visualizza su localhost (con meccanismo di autenticazione "EXTERNAL", ovvero sui nostri sistemi, fidandosi dell'utente UNIX → root è autorizzato a configurare)
    - Modifica del DIT della configurazione si fa comunque con ***ldapadd, ldapmodify, ldapdelete*** (sempre con autenticazione speciale).
    - Un'operazione che può essere necessaria è la definizione di nuove classi di oggetti e nuovi tipi di attributo, se quelli standard non sono adatti alla modellazione del nostro DIT. Vedi guida sul sito del corso