**Ambienti di programmazione per il software di base.**

[michele.ianni@unical.it](mailto:michele.ianni@unical.it) - Cubo: 44Z

Per e-mail sarebbe più comodo: [AMBIENTI] aiuto esercizio.

Prima parte del corso:

**Linux**, com’è formato, come poterlo utilizzare in maniera corretta.

**Command** **line**.

Modalità di interazione con il sistema operativo sia attraverso un’interfaccia grafica (finestre, cartelle), sia attraverso una modalità di tipo testuale ( command line – senza quindi l’utilizzo dell’interfaccia grafica ).

L’utilizzo della command line va a braccetto con la programmazione perché le shell sono questi programmi che permettono di interagire con il sistema operativo che danno la possibilità di automatizzare dei determinati comandi ( possiamo raggrupparle in dei file ).

Seconda parte del corso: **Linguaggio** **C**

Utilizzato nei più svariati contesti ( linguaggio general purpose ) – giochini, meccanismo per comandare un’astronave, controllo sensori.

Che cos’è l’Open Source e il sistema operativo Linux?

Aneddoto.

Anni ’70: Richard Stallman figura emblematica dell’Open Source, è un programmatore del MIT. All’università non si aveva un notebook per ognuno, si condivideva un supercomputer PDP10 connesso con una stampante. Cosa succedeva? Questa stampante, spesso, si bloccava perché rimaneva carta inceppata e quindi non sapendo cosa fosse successo (poiché erano lontani dal terminale) e quindi continuavano a mandare in stampa dei documenti fino a che non si riempiva la coda di stampa e fino a che uno di loro non passava da lì per togliere il foglio inceppato.

Allora cosa hanno pensato di fare al MIT, hanno preso il codice della stampante, che cosa faceva? Nel momento in cui un qualcuno mandava in stampa un documento e il foglio rimaneva inceppato, la stampante mandava un messaggio in broadcast (a tutti i terminali connessi) con su scritto “la stampante è inceppata, sistemala!” per cui in questo modo la stampante non rimaneva inceppata per un tempo lungo. Ciò è potuto accadere perché possedevano il codice del driver (*programma che serve a gestire la comunicazione tra un hardware – dispositivo fisico – e un software*) della stampante, se il codice non fosse a loro disposizione.

Anni ’80: ricevono in donazione una nuova stampante laser, allora Richard Stallman chiede all’azienda il codice del driver della stampante per apportare modifiche. No, il codice non lo possiamo dare perché è un’informazione privata e non è più Open Source. Il codice non è più aperto ad eventuali modifiche. Stallman non si da per vinto, chiama un altro universitario di cui era sicuro sul fatto che lui avesse il codice del driver della stampante laser, ma poiché avevano firmato NDA ( un patto di non disclosure ) non poteva digli il codice. Tutto ciò ha rappresentato un tradimento a quella che è la cultura hacker – il termine hacker non è nato con tono dispregiativo, in realtà rappresenta un insieme di persone che cercano di capire il funzionamento delle cose e cercano di modificarle ed utilizzarle in maniera diversa per migliorare le prestazioni e il loro utilizzo.

Vede questo come un alto tradimento e allora decide di intervenire in questo ambito e, nel 1983, fonda un progetto che prende il nome di GNU (g-nu) che ha come simbolo uno gnu.

GNU (G-nu Not Unix) – acronimo ricorsivo.

Inizia a lavorare full time su questo programma GNU, crea un set di programmi, tra cui un compilatore che utilizzeremo anche nel corso che è GCC ( Gnu C Compiler ).

FSF – Free Software Foundation. Fondazione che legata al Progetto GNU si occupa di diffondere il più possibile la distribuzione del codice Open Source, codice sorgente. Nel 1985 viene teorizzata nel The Gnu Manifesto che possiamo trovare tranquillamente online, e con l’aiuto della FSF.

Nel 1989 si crea una licenza generale e si viene a creare quella che oggi si chiama **Gnu General Public License** – licenza di uso completamente generale che spiega come i progetti protetti da questa licenza possono essere distribuiti, possono essere copiati, modificati ecc.. ma chi effettua i cambiamenti deve condividere il codice modificato con la stessa modalità. Sembra una cosa sciocca, ma è di fondamentale importanza. Dà un potere enorme, un utilizzatore normale non metterà mai mano ad un programma; invece, non è vero e quando si effettua una modifica deve essere un beneficio per tutti gli utilizzatori per questa ragione bisogna condividerlo.

“Il fatto che il progetto sia Open Source non significa che sia Gratis” -- “Free Software is not Free Beer” hanno comunque un modello di guadagno.

Parliamo adesso di Linux: fino ad ora abbiamo parlato di avvenimenti antecedenti alla prima diponibilità di Linux ( versione 0.001 17 settembre 1991 ), quindi cosa c’entrano questi due progetti?

Passiamo dalla storia di Richard Stallman alla storia di un altro importantissimo esponente. Ci troviamo in Finlandia, incontriamo un giovane studente universitario **Linus Torvalds.** – Tanennbaum crea un sistema da zero che si chiama Minix e lo usa nella sua università di Amsterdam per spiegare la materia “sistemi di reti”. Anche in Finlandia, nell’università di Linus Torvalds utilizzano il sistema operativo Minix.

Linus, è insoddisfatto di questo sistema operativo che presentava diverse difficoltà nella gestione dei sistemi di input/output, problemi di licenza e quant’altro. Allora comincia a scrivere un nuovo sistema operativo partendo dallo scrivere un **emulatore di terminale** con il linguaggio C e il linguaggio Assembly che aveva come scopo quello di comunicare con la porta seriale del proprio dispositivo e con processi separati. Scrive allora su uno Gnu Group all’epoca, e scrive: “Sto per iniziare questo nuovo progetto” e nel 17 settembre 1991 lo rilascia con la versione 0.001 che viene messo a disposizione su un programma già esistente che è il **Gnu Bash** – shell che già era stata implementata da Gnu.

*Linux, quindi, è molto legato al progetto Gnu. Linux sta per “Linus Unix”, ovvero la nuova versione di Unix by Linus.*

Quindi diciamo che Richard Stallman preme sul fatto che questo nuovo sistema operativo dovesse chiamarsi Gnu/Linux perché l’uno senza l’altro rappresentava ben poco. (Quando parliamo solo di Linux ci riferiamo solo al Kernel mentre quando parliamo di Gnu/Linux ci riferiamo all’interno sistema operativo).

Chi utilizza Linux?

Applicazioni Web, siti dove girano? Su un milione di dispositivi, dispositivi di Storage, Router, la maggior parte gitano su ambienti Linux.

A livello desktop invece Linux ha la share ( percentuale di utilizzo ) inferiore. Sebbene non sia così diffuso a livello desktop, tecnicamente ci sarebbero anche dei vantaggi in quello. Vediamo quali: uno di questo è che ci sono tante versioni di Linux che girano anche su hardware datati. Immaginiamo 20anni fa, i documenti che venivano scritti su Word giravano su un Pc di 2GB di RAM. Come mai adesso per scrivere lo stesso documento sullo stesso programma mi serve un PC di almeno 8GB di RAM? Perché ci sono aggiunte ovviamente, c’è necessità di rendere hardware sempre più performanti in modo tale che il mio computer non va più bene, lo butto per comprarne un altro con prestazioni migliori (e prezzo ovviamente maggiore).

Una marea di altri ambienti in cui Linux è utilizzato ampiamente: Device Mobile ( i cellulari ) alla nascita di Android che ha un kernel Linux ( è basato su Linux ), alcuni dispostivi Nokia possedevano la variante Linux, tantissimi sistemi IoT (Internet of Things) possiedono la variante Linux. Una marea di Web Server si basano su Linux.

I super computer più utilizzati al mondo sono 500. Di questi 500, 497 (99,4%) aveva una distribuzione Linux e sullo 0,6% girava Unix.

Se prendiamo in conto tutti i dispositivi che hanno una potenza di calcolo che non sono solo computer, e che hanno un sistema operativo integrato (anche frigoriferi smart, sensori, ecc..) ci renderemo conto che l’ago della bilancia pende molto di più dalla parte di Linux che da quella di un altro sistema operativo.

Per cui imparare ad usare un sistema operativo come Linux da sviluppatori è molto molto importante per un ingegnere informatico.

Che cos’è una distribuzione? Linux ha dei momenti importanti. Nel 1991 è stata rilasciata la prima versione (rilascio fatto per gioco), con il tempo poi ci è stato lo sviluppo di questo sisop.

(Debian, Redat, distribuzioni importante che resistono tutt’ora che si differenziano dalla disponibilità di pacchetti – un archivio che contiene dei programmi – differenti. Altre distinzioni erano sulle licenze specifiche.)

Ad oggi abbiamo una marea di distribuzioni (distrowatch.com) che è una collezione di distribuzioni che sono presenti in rete. Per la finalità di questo corso è bene utilizzare **Ubuntu** Linux (basata su Debian, quindi è Debian based). È bene utilizzare Ubuntu poiché utilizzatissimo e quindi nel caso di problemi è facile trovare come risolverli.

Quando si fa una modifica su un progetto Open Source, si fa una modifica e si fa una pull request.

VLC, ad esempio, è gestito da Video Lan e quindi nel momento in cui si prende il codice sorgente e si modifica un qualcosa all’interno del programma vanno rimandati ai tizi che gestiscono VLC. Nel caso di una modifica sul Kernel Linux è un po’ diverso.

Alcune modifiche posso decidere di effettuarle solo in locale, ad esempio se voglio che sul mio Firefox venga sempre visualizzato Federica Guido, posso farlo senza problemi.

La distribuzione si differenzia anche per il Desktop Environment (ambiente desktop), in Windows non cambia la grafica (possiamo cambiare lo sfondo, possiamo cambiare la dimensione delle icone). In Linux esistono diversi ambienti grafici differenti, possiamo utilizzarlo anche senza completamente. Esistono due tipi di interfacce grafiche:

**Window** **Manager** e **Desktop** **Environment**.

* Window Manager è un software di base che si occupa di disegnare semplicemente le finestre sullo schermo. Disegna il contorno della finestra (tasto di chiusura, per ridurre a icona e per ingrandire, alcuni menu a tendina e nulla più).

FluxBox, Erblues.

* Desktop Environment : ambienti completi – non offrono solo quello che fa il Window Manager ma anche delle impostazioni che permettono di modificare l’interfaccia grafica, le calcolatrici, i giochi integrati, gli analizzatori dello spazio su disco, il launcher dei programmi, che non sono propriamente legate al sistema operativo di base.

KDE, XFCE, LXDE, Pantheon. I desktop Environment sono una delle più grandi pesantezze del sistema operativo (chi ha un hardware datato, è meglio utilizzarne alcune più leggere – tipo K-Ubuntu che utilizza KDE e così via…).

TTY: Interfacce di comando che permettono di comunicare in maniera seriale con dei dispositivi specifici. Per questo motivo, quando parliamo di terminale, non parliamo di terminale ma di emulatore di terminale perché è come se stesse emulando il comportamento di un terminale reale.

Per seguire per bene il corso c’è bisogno di un po’ di cose:

* Distribuzione Linux (Ubuntu)
* Shell (Utilizziamo Bash)
* Editor di testo (per scrivere del codice ed interagire con C e Bash) – Emax, Vim, Sublime, VsCode, blocco note.

Distribuzione Live – distribuzione su pennetta.

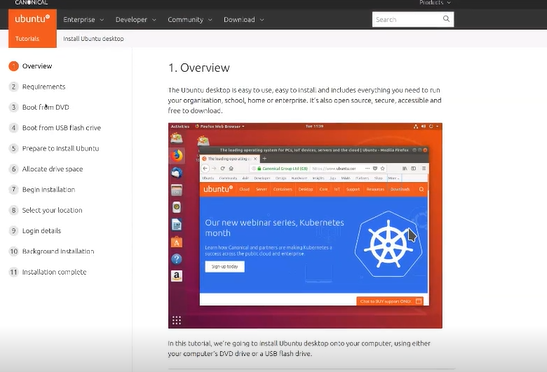
Opzione consigliata: full disk installation.

Dual boot: modalità di installazione che prevede la possibilità di installare due o più sistemi operativi sullo stesso hardisk. All’avvio il boot loader permette, attraverso un’interfaccia grafica basilare, di scegliere quale sistema operativo preferiamo avviare.

Macchina virtuale: creiamo una macchina virtuale e la usiamo su di essa. Sconsigliato: porta ad utilizzarlo poco. Per qualsiasi cosa utilizziamo subito Windows.

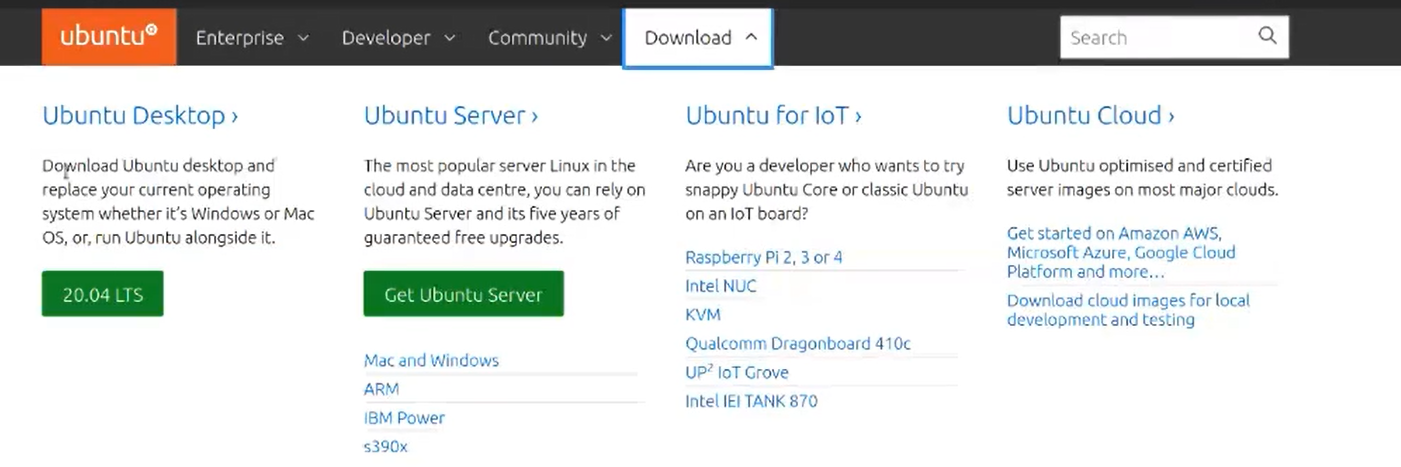
Macchine cubo 40 che hanno delle distribuzioni Linux installate.

Da Windows bisogna ridurre lo spazio occupato dal sistema operativo e lo si fa dal gestore dei dischi di Windows.

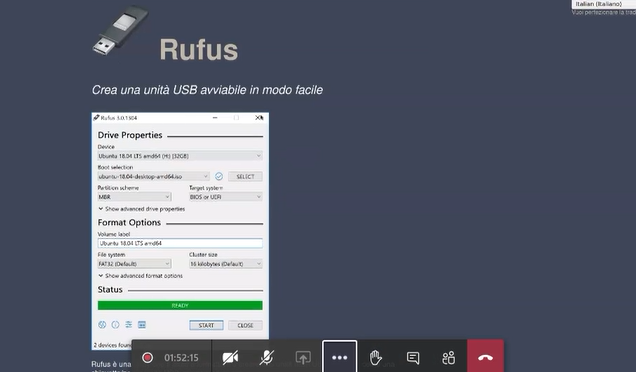


Bisogna avere almeno 25GB su disco.

Successivamente proseguiamo con l’installazione di Ubuntu dal sito (ubuntu.com) e farne il download. ANDARE SU UBUNTU DESKTOP

vengono rilasciate. LTS ogni 2 anni, ci sono pacchetti più stabili, consigliata per un computer che viene aggiornato meno frequentemente. Dopo aver scaricato la ISO come facciamo a copiarla sulla pendrive? C’è la necessità di rendere quella pennetta USB bootable, uno si chiama Rufus. È una piccola Utility in cui inserendo la ISO che abbiamo scaricato dal siro di Ubuntu, scegliamo il device corretto e quando clicchiamo su START, creiamo una pennetta che è in grado di essere avviabile in modo facile.

Dopo aver fatto questo dobbiamo accedere alle impostazioni del BIOS ( Basic Input Output System ) del computer. Dobbiamo dapprima disabilitare una funzione che si chiama “Secure Boot”, ha poco a che fare a dir la verità con la sicurezza, si va a vedere se l’ash del sistema operativo sia uguale ad uno lecito (evita che altri sistemi operativi possano girare sulla nostra macchina). Dobbiamo anche modificare l’ordine di avvio (ordine di boot) – come primo dispositivo si dovrebbe trovare l’hardisk. Dobbiamo fare in modo che parta dalla flash drive e quindi dobbiamo modificare per permettere che il primo componente che venga avviato sia la periferica USB su cui è installata la ISO di Ubuntu.

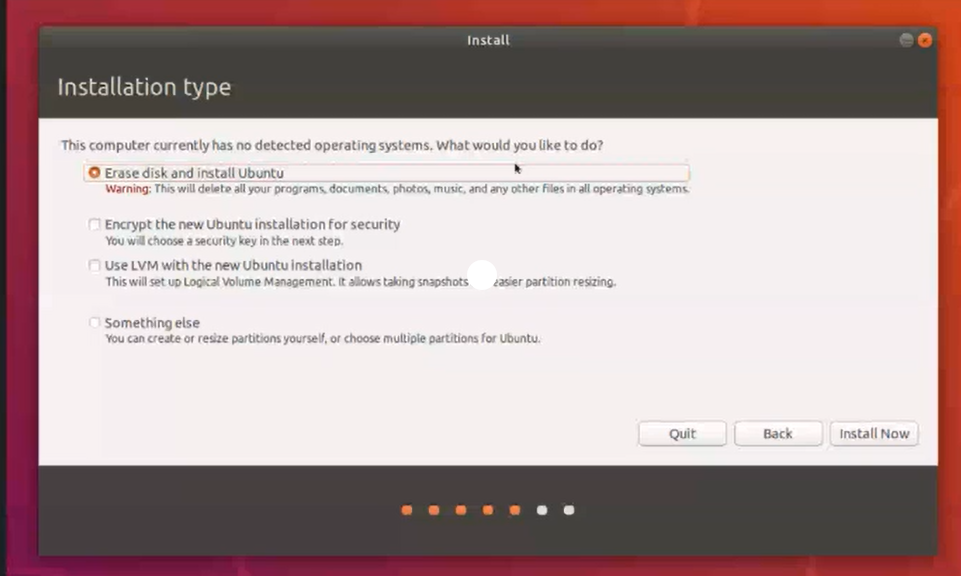


Installazione normale o installazione minimale.

Bisogna installare:



Solitamente viene selezionata.



Comparirà anche Installa Linux al fianco di Windows.

**Lezione 2:**

Prima parte della lezione: Storia di Unix e Linux che si sovrappone con la storia dei computer negli Stati Uniti.

Seconda parte della lezione: ARCHITETTURA DI LINUX (accenni) e alcuni comandi su cui esercitarci.

Excursus veloce:

Negli anni ’40, ’50 tutte le università avevano dei computer dedicati (non in rete), ogni utente aveva il proprio computer oppure più utenti condividevano un computer (che chiaramente poteva essere usato da una persona alla volta).

Questa situazione non era praticabile. Negli anni ’60 sono nati dei computer di tipo BATCH , computer che eseguivano un lavoro (in batch), nel momento in cui si lanciavano i job, avevano pieno possesso della macchina fino alla completa esecuzione del job.

Si usavano le schede perforate, quindi il programma veniva installato su questa scheda, inserito nel computer per essere eseguito. Capiamo le difficoltò nel debugging, immaginiamo se sbagliavano qualcosa e il programma andava in coda, correggerlo e poi lanciarlo il giorno dopo.

Le cose migliorarono quindi negli anni ’60, ’70. Il MIT (Massachusset Institue of Technology) insieme a Ken Thompson, inventarono il time sharing.

Ad un certo punto, nel 1969 nel Bell Laboratories nacque questo sistema operativo chiamato MULTICS che sta per Multiplexed Information and Computing Services, programmati principalmente in Assembler.

Dennis Ritchie insieme a Thompson, progettarono UNIX (Uniplexed Information and Computing Services), scritto in Assembly, nei primi anni ’70.

Lo progettarono su un nuovo calcolatore PDP-11, evoluzione del PDP.7 che si diffuse rapidamente in molte università.

Dal System 3 in poi il sistema UNIX era a pagamento. Una versione commerciale. In contrapposizione a questo sistema UNIX sviluppato dai Bell Labs, l’università di Berkley ne creò un altro che chiamò BSD (Berkley Software Distribution) e ne furono fatte varie versioni.

DISCUTIAMO DEL KERNEL DI LINUX

Linux ha due modalità di utilizzo:

1. User Space
2. Kernel Space

Partendo dal basso in su, supponendo che in basso ci sia l’utente

Esercitazione 2:

Shell share è un sito web che permette la condivisione in tempo reale di un terminale.

La riga di comando è una shell, un eseguibile. Ha come compito prendere dei comandi da testiera e trasferire questi comandi al sistema operativo e di mostrare l’eventuale output relativo.

La shell più comune installata nella maggior parte dei sistemi operativi che ha il nome di BASH = programma che ci permette di comunicare con il sistema operativo sottostante.

**Bash** = *Bourne* *Again* *Shell* → Bourne che fa riferimento ad una vecchia shell che si chiama SH (ancora esistente che era stata creata da un programmatore di nome Steve Bourne. Nel momento in cui si è deciso di ricreare una shell nuova lo si è fatto a partire da SH che è la shell originale di UNIX. Per questo motivo si dice Bourne Again in breve BASH.

Sul terminale c’è scritto **GNOME**. Quando si utilizza un’interfaccia grafica come un Desktop Environment come stiamo facendo noi, per interagire con la shell abbiamo bisogno di un altro programma che prende il nome di Terminale Emulator (emulatore di terminale), quale si sta emulando? L’utilizzo dei computer con interfaccia grafica a finestre è un’invenzione abbastanza recente (prima si interagiva con i computer solamente attraverso la riga di comando, ci si connetteva a dei mainframe (computer potenti) attraverso alcuni terminali. Quando è stata creata l’interfaccia a finestre c’era in qualche modo la necessità di emulare il terminale vecchio e sono stati creati dei programmi completamente grafici*. Il compito degli emulatori di terminali è semplicemente quello di far girare una shell al loro interno.*

Se guardiamo nei nostri Desktop Environment, abbiamo GNOME, potremo avere XTERM, TERMINATOR (che rende facile la creazione delle finestre, splittando quelle già esistenti) ecc..

L’interfaccia è piuttosto “brutta” perché se non conosciamo i comandi non sappiamo cosa fare:

Cosa compare come prima riga?

[michele@void ~]$ poi c’è il cursore lampeggiante.

La parte prima del cursore lampeggiante si chiama Shell Prompt. Può cambiare da distribuzione a distribuzione. Nelle parentesi quadre c’è in nome (utente che utilizza la shell in quel momento).

*Linux è un sistema operativo multiutente. Può essere utilizzato da più utenti anche nello stesso arco temporale. Ognuno di essi avrà uno spazio dedicato e determinati privilegi.*

Per determinare quale utente ha avviato la shell, prima della @ compare il nome utente.

Subito dopo la chiocciola c’è scritto “nome\_macchina” .

Dopo il nome della macchina c’è “~” che ricorrerà molto spesso nel corso. (Su Linux Alt gr + ì )

Questo simbolo rappresenta la HOME DIRECTORY dell’utente corrente.

Ogni utente ha a disposizione dello spazio che è racchiuso all’interno della cartella HOME. Quando avvio la shell, la tilde sta a significare che mi trovo nella cartella home dell’utente che ha avviato il terminale (/home/michele).

$ → Il dollaro sta ad indicare che l’utente che sta utilizzando il computer in quel momento è un utente che non ha privilegi di amministrazione (non è un super-utente). Se fossi stato un superutente (super-user) il simbolo sarebbe stato “#”.

utente root = come abbiamo già detto il sistema operativo Linux prevede la presenza di più utenti che possono operare contemporaneamente. Esiste un utente di default che è il cosiddetto il super-utente che si chiama “root”. Ha qualsiasi privilegio sulla macchina : può accedere a qualsiasi file, cancellarlo ecc..

Siccome root ha grossi privilegi e anche grandi difficoltà. L’utente root può fare danni di sistema irreparabili, cosa che non può fare l’utente michele o l’utente federica (non è super-user). Nel momento in cui si vede quindi il simbolo “#” sappiamo che dobbiamo fare delle operazioni con molta cautela.

Giò vediamo delle modifiche:

[root@void michele]# → adesso al posto della Tilde (non mi trovo nella home dell’utente Root) perché mi trovo nella home di michele.

In qualsiasi momento noi utilizziamo una shell siamo in un luogo specifico del nostro pc (di default è la home directory dell’utente da cui stiamo facendo partire il terminale).

Come si cancella la shell? **Ctrl+L o “clear”.**

I comandi magari non li ricordiamo : la bash ha un sistema che si chiama “history” che è lo storico di tutti i comandi che sono stati eseguiti.

Si ricorda degli ultimi 500 comandi che sono stati eseguiti. Come si fa ad accedere alla history. Con i tasti “GIU” e “SU” (le frecce sopra e sotto).

È buona norma però abituarsi ad utilizzare altre combinazioni di carattere. Per non sollevare la mano e muoverla verso le frecce.

Primo comando:

**Ctrl+P e Ctrl+N**. Si ricordano facilmente perché P = previous e N = next.

Freccia destra o freccia a sinistra per muovere il cursore oppure altri due comandi:

**Ctrl+B** = back e **Ctrl+F** = forward (lettere)

**Alt+B** e **Alt+F** si fa avanti parola per parola invece (parole)

Ctrl+A = inizio riga

Ctrl + E → end = fine riga

Ctrl + K → uccide la riga (kill)

TAB → tasto (premuto 2 volte) serve per fare l'auto-completamento. Se non ci sono tante combinazioni ma solo una, allora completa direttamente il comando.

RTFM →Read The Fucking Manual

Quando facciamo il commando “free” vediamo la Voce **SWAP**.. che cos’è?

Area dell’hardisk che viene utilizzata in sostituzione della memoria RAM nel momento in cui è satura. In Windows c’è un file analogo che è pagefile.sis

È utile avere 0 used di Swap? Se si ha tanta memoria RAM a disposizione, avere la Swap è superfluo. Quando è necessario o quando è utile? È utilissimo quando si ha poca RAM a disposizione, è utile quando si vuole ibernare il computer perché si fa la copia del contenuto della RAM all’interno di questa area di Swap (Partizionamento Automatico che è quello che fa l’Installer, se poi in un futuro ci dovessimo rendere conto che non ci soddisfa possiamo andare a cambiarlo manualmente). La partizione è soggettiva, dipende dal computer e dalla persona.

Posso spostarmi dalla cartella Home, o ,laddove non so dove mi trovo in quel momento.

pwd → print working directory – mostra dove mi trovo.

Output che mostra pwd --- /home/michele → la slash a destra ha due significati:

1. Identificatore delle cartelle – separare cartelle e cartelle
2. sta ad indicare la cartella root.

Immaginiamo la gerarchia delle cartelle come se fosse un albero.

La radice dell’albero in inglese si dice **root**, per cui esiste una cartella chiamata “/” da cui parte tutto, la quale quindi contiene tutte le altre cartelle.

In Windows “C:” che potrebbe essere il “sinonimo di root in windows” non lo è propriamente perché non contiene la totalità delle cartelle, ad esempio se inseriamo una pendrive viene creata un’altra unità ad esempio “D:” che non si trova all’interno di C:

In Linux tutto è contenuto a partire dalla cartella root.

CIO’ CHE VIENE RESTITUITO DA pwd SI CHIAMA PATH ASSOLUTO= percorso

**PATH ASSOLUTO** = *non dipende dalla cartella i cui ci si trova in quel momento. Ci permettono di arrivare a destinazione seguendo il percorso indipendentemente da quale sia il punto in cui ci troviamo. Sono quelli che iniziano con la “/” che è la cartella root.*

**PATH RELATIVO** = *percorso relativo alla cartella in cui si trova in quel momento. Sequenze di cartelle che permettono di arrivare a partire da quel punto di origine alla destinazione che si vuole raggiungere.*

Per capire meglio:

Immaginiamo di trovarci all’Unical e di trovarci all’altezza del cubo 10 e qualcuno mi chiede di arrivare alle pensiline “la risposta è: vai avanti per 300 metri e ti troverai alle pensiline”. Questa informazione va bene se effettivamente ci troviamo al cubo 10, se ci troviamo in Mongolia, non si può rispondere “vai avanti per 300 metri”. Questa è una risposta che va bene relativamente alla posizione in cui ci troviamo in quel momento.

L’equivalente nella vita reale del path assoluto sarebbe:

parti dal mondo, vai nel continente europeo, vai in Italia, vai in Calabria, arriva ad Arcavacata di Rende, via Pietro Bucci, arriva all’altezza del cubo 10, 300 metri e arrivi alle pensiline”. Va bene sempre.

se io scrivessi “*cd /Documenti ”* sarebbe errato perché io non ho in root una cartella che si chiama Documenti. Devo scrivere “**cd Documenti/”**

Ovviamente a cd posso passare sia path relativi che path assoluti.

Usare dei path relativi ci permette di scrivere meno testo, in teoria si potrebbe scegliere di utilizzare sempre path relativi. Funzionano solamente però se ci si trova in alcune cartelle specifiche del computer.

Devo utilizzare sempre path relativi o sempre path assoluti? Quando è utile utilizzare path assoluti e quando path relativi?

Path assoluto = quando abbiamo omonimie

Path assoluto = quando non sappiamo in che directory verrà eseguito uno script. Magari non lo posso conoscere in maniera diretta.

In realtà le cartelle hanno due file nascosti.

Se faccio ls -a mi esce come output:

1. file che si chiama “.” → cartella corrente
2. file che si chiama “..” → cartella che contiene la cartella corrente

Come facciamo a capire con che tipo di file abbiamo a che fare?

Con il comando file nomeFile per vedere di che tipo è un file

La cartella root è una cartella particolare, non è direttamente modificabile. Necessita che si abbiano permessi da super-user.



All’interno della cartella root esiste una gerarchia di cartelle che è + o meno invariata.

La cartella **BIN** sta per Binaryes, all’interno di questo ci sono tutti programmi eseguibili.

La cartella “**BOOT**” contiene il kernel Linux, contiene l’immagine della RAM disk iniziale, contiene il boot loader.

cartella **DEV** = device nodes → concetto che viene conosciuto con un mantra che è “*Everything is a file”* , per come è strutturato Linux, qualsiasi cosa (un dispositivo hardware connesso a un file, lo standard input e lo standard output) è effettivamente un file.

In fase di avvio del nostro computer vengono creati dei file speciali che non hanno un contenuto fissato ma un contenuto che dipende dalle operazioni di lettura e di scrittura che facciamo su di essi. I file nella cartella dev/ specificano delle componenti che sono attaccate in qualche modo al computer, rappresentano un modo per comunicare con i device hardware.

Forniscono una maniera semplice (già conosciuta) per comunicate con qualsiasi cosa.

Cartella **ETC** → contiene file di configurazione

**HOME** → ogni utente ha una cartella home

**LIB** → Shared Libraries = sono delle librerie condivise

**LIB64** → contiene librerie a 64bit

**LOST + FOUND** → contiene dei file che servono a fare il recovery in caso ci sia un evento di corruzione del file system. Generalmente è vuota, se succede qualcosa al file system la cartella viene riempita da quei file che servirebbero in caso di corruzione

**MEDIA** → contiente i "punti di mount" dei dispositivi rimovibili, come le pendrive. Dispositivi che vengono montati automaticamente al loro inserimento. Che cosa vuol dire montare una periferica? "Se la pennetta viene montata automaticamente, il contenuto di essa sta in /media/nomePennetta". Generalmente come punto di montaggio veniva utilizzata mnt/.

**MNT** → veniva utilizzata una volta per il montaggio.

Se sono montati in maniera automatica vanno a finire in media, se sono montati in maniera manuale possiamo scegliere se metterli nella cartella media/ o nella cartella mnt/.

*Montare* = prendere il contenuto del dispositivo rimovibile e renderlo disponibile all’interno della cartella del nostro computer locale.

Quando noi montiamo una pennetta USB ce la ritroviamo sia in media/ che in dev/? Si però in dev/ abbiamo un device virtuale che è utile per la comunicazione con hardware. In media è montato il contenuto della cartella come se stessimo accedendo al contenuto della cartella stessa. Se facciamo un utilizzo normale del computer, in dev/ non vi si accede mai (nessuno ha la necessità di comunicare con l’hardware specifico). Se dobbiamo copiare un file dalla pennetta al pc, quel file lo andiamo a prendere da media/.

Come si fa ad accedere ad una cartella che contiene spazi dato che spazio non è consentito nei comandi?

Esercitazione 3:

Analogia tra l’utilizzo della Shell di Linux e i videogiochi per Bambini. Windows assomiglia al Game Boy, è bello, divertente ma non permette molta personalizzazione e non permette di costruire quello che si vuole manipolando le sue componenti di base.

Viene fatta l’analogia invece tra Linux e i LEGO. Presi da soli sono meno belli, ma l’utilizzo congiunto permette di costruire con poco sforzo qualcosa di più complesso – più vicino alle nostre aspettative ed esigenze.

Dobbiamo imparare un collante tra questi piccoli comandi in modo da utilizzarne alcuni più lunghi e più utile.

Allora:

Tantissimi programmi che utilizziamo producono Output in qualche modo. Può essere di due tipi:

1. Output corretto di un programma. Se io invoco il comando “ls” mi restituisce il contenuto della cartella corrente o il contenuto della cartella “x” se passo “x”
2. Messaggio di errore = sebbene sia equivalente ad un output corretto, in realtà sono due concetti differenti perché uno (il primo) è legato al concetto di *Standard Output*, mentre l’altro è legato ad un concetto che prende il nome di *Standard Error*.

Sia **Standard** **Output** (**stdout**) che **Standard** **Error** (**stderr**) , siccome è utile mostrarli all’utente vengono entrambi mostrati. Per cui dal punto di vista fisico sono legati perché entrambi mostrano output a video.

Se ci si scrive dentro entrambi mostrano a video il risultato. È come se fossero tubi connessi con il nostro schermo.

Accanto a questo concetto c’è quello di **Standard** **Input** (**stdin**) → deve essere collegato con l’input che l’utente dà nell’interazione con un comando specifico. Di defualt lo stdin è attaccato alla tastiera (significa scrivere sulla tastiera).

REDIREZIONE → connettere questi piccoli blocchi. Il primo passo che vedremo per connettere dei blocchi è proprio la redirezione. È quel processo che ci permette di ridefinire dove vanno a finire stdin, stdout o stderr.

Immaginiamo di voler creare un file che contiene il contenuto della cartella 1.

Immaginiamo che il nostro programma sia una scatola che ha tre tubi. Uno per l’input e due per gli output. Per qualche motivo vorremmo non vedere al schermo il risultato ma salvarlo su file, oppure vorremo poter passare un input non attraverso la tastiera ma ad esempio attraverso una periferica di rete. Come facciamo a cambiare questi tubi e collegarli con altri terminali?

*Attraverso gli operatori di redirezione.*

Prendiamo ad esempio il contenuto di una cartella e mettiamolo all’interno di un File.

**File** **Descriptor** = in Linux ogni file ha un codice associato. Mette a disposizione i primi 3 file descriptor che hanno come numeri 0 1 e 2 che sono legati che sono legati rispettivamente a stdin (0), stdout(1), stderr(2). La shell fornisce un operatore per redirigere i file utilizzando il numero di File Descriptor – il comando è ad es (ls -l cartella4/ 2> errore\_ls.txt)

Possiamo pure unificare gli ouput. Tramite “>” redirigo solo lo standard output. Con l’operatore “>>” redirigo sempre solo lo standard output ma senza sovrascrivere.

con “numFileDescriptor>” redirigo lo standard error. Ma se volessi unificarli?

Posso utilizzare /dev/null per buttare errori ad esempio perché nella cartella null nella cartella dev diciamo che la possiamo vedere come un pozzo senza fondo ed è come na sorta di cestino.

*“Parla tanto i tuoi commenti li butto in /dev/null”*

cat sta per concatenate → perché concatenare file?

Come si redirige invece lo standard Input? Tramite il comando “cat” se non passiamo alcun parametro lo ridiregiamo come uno Standard Input, si aspetta che scriviamo qualcosa. Per uscire dalla scrittura premere ctrl+d.

Vediamo due cose che apparentemente sembrano uguali:

Immagine che contiene testo

Descrizione generata automaticamenteFare **cat** con la redirezione dello StandardInput a video di un cavo di rete mi avrebbe permesso di vedere a video tutto il contenuto che proviene dal dispositivo di rete in questione.

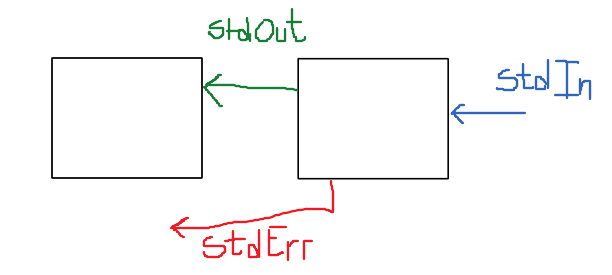
Da dove prende il comando di input come si fa? Andiamo a vedere il manuale.

PIPE → concatenare |

ad esempio comando1 | comando2 succede che il comando1 viene utilizzato come Standard Input del Comando2. Vediamo un paio di esempi.

La barra viene chiamata Pipe.

Immaginiamo due scatole. Lo standard Output del primo parametro diventa lo standard Input del secondo comando :



La connessione tra i due comandi è utilissima.

Pipe o PIPELINE → ad esempio:

Le pipeline sono utilizzate con dei comandi che prendono il nome di *filtri*, perché servono per alterare un output fornito da un terminato comando filtrandolo in base alle esigenze dell'utente.

Esistono diversi comandi che vengono utilizzati come filtri:

ad esempio

cat elenco\_studenti.txt | sort | uniq

Immagine che contiene testo

Descrizione generata automaticamente

Li ordina ed elimina tutti i duplicati.

Non c'è un limite ai comandi che si posso concatenare.

IL RISULTATO DELL'ELABORAZIONE DEL PASSO PRECEDENTE VA A FINIRE NEL PASSO SUCCESSIVO.

Lezione 3:

Cosa sono le pipe in Linux? Servono per eseguire una serie di comandi che si basano su un set di dati e che possono essere eseguiti in sequenza p per spostare i dati in maniera efficiente da un comando all’altro.

**Wild cards** → caratteri speciali che ci permettono di usare in maniera più approfondita i vari comandi di Linux.

**File-system di Linux** →

**Permessi file** →