Breve guida all'uso di Linux e della connessione remota sotto Windows 10

Questa guida viene scritta nell'Ottobre del 2020; e' aggiornata a questa data ed ovviamente alcune cose potrebbero cambiare o diventare obsolete in seguito

Il sottosistema Linux per Windows (WSL)

Windows 10 possiede un sottosistema linux, chiamato WSL. Questo sottosistema consente di far girare una distribuzione linux sotto windows e elimina la necessita' di installare linux in una partizione separata. Inoltre garantisce un certo livello di interoperabilita' tra i due sistemi operativi, ad esempio, si puo' copiare negli appunti qualcosa da un terminale linux ed incollarlo in un applicativo windows e viceversa.

A questo link:

https://docs.microsoft.com/it-it/windows/wsl/install-win10 (https://docs.microsoft.com/it-it/windows/wsl/install-win10)

trovate il documento MS che guida all'installazione. Seguendolo passo a passo non ci dovrebbero essere problemi.

Una volta installato WSL, sara' a disposizione un *terminale linux* con una installazione basica di una distro (consiglio UBUNTU e quanto segue si applica a tale distro).

Attenzione: non avrete una distro completa ed in particolare non l'interfaccia grafico (GUI). Non e' questo lo scopo del WSL. Avrete un terminale, con un certo numero di comandi di linea installati e la possibilita' di installarne altri. Questo e' sufficiente per configurarsi un ambiente di programmazione.

Molti software necessari per utilizzare WSL come ambiente di programmazione vanno installati a mano, a partire.. dal **compilatore**

Il comando per fare cio' e':

In [ ]:

```
sudo apt-get install gcc
```

(in ogni terminale, sudo vi chiedera' la password che avrete fornito in fase di installazione alla sua prima esecuzione)

Prima di usare apt-get e' opportuno assicurarsi di avere a disposizione la versione piu' recente, che punti ai repositories piu' aggiornati. Per ottenere questo il comando e':

In [ ]:

```
sudo apt-get update
```

Alcuni pacchetti utili e consigliati da aggiungere alla distro, qualora non gia' presenti, sono:

- python
- gnuplot

- jupyter
- emacs

Nota per gli utenti python: molti moduli python utili, per esempio numpy, vanno a loro volta installati. Pero' in ambito python di solito il comando per installare tali moduli e'

```
sudo pip install numpy
```

...non fatemi scrivere una guida all'uso di pip, per favore...

Grafica X

Per utilizzare la grafica linux, sia quella di WSL sia grafica da connessioni remote a sistemi linux/unix, occorre installare sotto windows un **server X**. Il server X e' il motore grafico dei sistemi linux/unix. Ne esiste uno in MS store chiamato **X410**, *non* e' gratis ma costa poco (l'ultima volta che ho guardato erano 5 euro). Ve ne sono di free, ad esempio **Xming**. Potrebbero pero' causare noie in fase di installazione e soprattutto di configurazione.

Una volta scaricato ed installato il server, ad ogni accesso a windows occorre lanciarlo, oppure occore automatizzare la sua partenza. Il server **non** ha impatto nullo sulle risorse della macchina percio' personalmente lo lancio quando mi serve.

A server X up-and-running, la grafica di WSL funzionera' se prima viene dato il comando:

In [ ]:

```
export DISPLAY=localhost:0.0
```

nel terminale linux.

Questo comando va dato ad **ogni** esecuzione del terminale, quindi e' meglio automatizzare la sua esecuzione. Cio' si ottiene editando il file

.bashrc

che risiede nella home directory ed aggiungendo il comando di cui sopra alla fine del file

.bashrc viene eseguito all'inizio dell'esecuzione di ogni shell linux. Attenzione che non vedrete comparire il file dando `ls` perche' il . iniziale classifica i files come nascosti, in unix. Lo vedrete pero' (con un certo numero di fratellini suoi) dando `ls -a` .

Su alcuni sistemi, il numero (0.0) potrebbe essere diverso, ma non *dovrebbe* se la configurazione del server X e' corretta.

Si noti che definire una variabile di ambiente ***non*** ha impatto sulle prestazioni della macchina, quindi la procedura puo' essere automatizzata senza effetti collaterali.

Aggiungo che `.bashrc` , venendo eseguito all'inizio di ogni esecuzione di shell, e quindi in particolare, di ogni terminale, e' il posto dove mettere i comandi che devono *sempre* essere eseguiti all'inizio di ogni sessione di lavoro.

Importante

Se vi connettete ad una macchina remota (linux/unix) direttamente da windows (vedi sotto), per attivare le funzionalita' del vostro server X, dovete *comunque* definire la variabile DISPLAY, ma questa volta, in ambiente windows. Questo si puo' fare semplicemente per esempio dal *command prompt* dando il comando

```
set DISPLAY=localhost:0.0
```

e potete vedere le variabili d'ambiente definite dando

```
set D
```

(date "a capo" dopo D; fara' vedere tutte quelle che iniziano per D).

E' pero' scomodo dover settare DISPLAY tutte le volte che accedete. Si puo' settare una variabile d'ambiente permanente nel seguente modo:

```
aprite le impostazioni di windows
andate su "sistema"
scrivete "variabili d'ambiente" nella casella di ricerca (in alto a sinistra)
scegliete "modifica variabili di ambiente per l'account"
si aprira' una finestra di dialogo; cliccate "Nuova"
mettete DISPLAY nella casella di testo "Nome variabile"
mettete localhost:0.0 nella casella di testo "Valore variabile"
```

per attivare la nuova variabile, dovete ***uscire e rientrare*** nell'account.

Visibilita' dei file linux/WSL da windows e viceversa

WSL consente a linux di vedere tutti i files presenti sulla macchina. D'altro canto **non** avrete accesso direttamente da Windows ai files creati all'interno di WSL.

Per accedere ai files windows da linux, date innanzi tutto, all'interno del terminale, il comando:

In [ ]:

```
df -h
```

otterrete un output di questo tipo:

```
Filesystem      Size  Used Avail Use% Mounted on
rootfs          918G  766G  152G  84% /
none            918G  766G  152G  84% /dev
none            918G  766G  152G  84% /run
none            918G  766G  152G  84% /run/lock
none            918G  766G  152G  84% /run/shm
none            918G  766G  152G  84% /run/user
tmpfs           918G  766G  152G  84% /sys/fs/cgroup
C:\             918G  766G  152G  84% /mnt/c
D:\              13G   12G  1.5G  89% /mnt/d
```

la *prima* linea mostra il filesystem dove risiedono i files di linux, sia quelli di sistema che quelli dell'utente. I files utente stanno di solito in

/home/*nomeutente*

La *penultima* linea mostra il disco del computer. Quindi da linux si accede ai dati windows usando /mnt/c; e' l'equivalente del path windows `C:\` Si noti che la dimensione del disco e' sempre la stessa (tranne che per D:\ che di solito e' una recovery partition): questo e' perche' in realta' **tutti** questi filesystems insistono sulla *stessa* partizione dello *stesso* disco.

Per esempio, la cartella "Downloads" si trova in

/mnt/c/Users/*nomeutente*/Downloads

dando il comando

In [ ]:

```
cp miofile.c /mnt/c/Users/nomeutente/Downloads
```

copierete il file miofile.c, presente nella vostra home directory linux, in Downloads, accessibile a Windows. Sara' ad esempio visibile aprendo la cartella "Downloads" con esplora risorse. (Attenzione che in una installazione italiana di Windows potreste trovare /mnt/c/Utenti/nomeutente/Downloads invece di quanto scritto sopra, cioe' Users potrebbe essere sostituita da Utenti)

Il contrario e' anche vero, se scaricate un file da Windows e lo lasciate mettere in Downloads, come e' comportamento di default,

In [ ]:

```
cp /mnt/c/Users/nomeutente/Downloads/file_scaricato .
```

copiera' il file nella home directory linux (supponendo che li' stiate), dove ci potrete lavorare sopra. Si noti che da linux, dando

In [ ]:

```
cd /mnt/c/Users/nomeutente/Downloads
```

si puo' anche andare a lavorare direttamente nella cartella Downloads invece di copiare il file in /home/*nomeutente*. In questo caso qualsiasi cosa facciate, sara' visibile da esplora risorse, Pero' lo sconsiglio per motivi di ordine nel filesystem del PC.

Inoltre linux e' molto piu' permissivo di windows per quanto riguarda le operazioni ammissibili; se andate nella cartella /mnt/c/Windows, potete fare danni inenarrabili, sino a rendere impossibilie la ripartenza del computer.

NON FATELO

Attenzione a non confondervi. Quando usate il terminale linux usate comandi linux, quindi per esempio *ls* per listare i files (invece del "dir/w" del prompt dei comandi win), e la sintassi dei nomi dei files e' linux **non e'** windows. Per fare un esempio, la directory "Downloads" nel prompt comandi di windows ha il nome file

```
C:\Users\nomeutente\Downloads
```

mentre nel terminale linux e'

```
/mnt/c/Users/nomeutente/Downloads
```

MS avvisa che WSL non e' pensato per fornire una distro *completa* di linux ed in particolare non per farci girare un ambiente grafico, per esempio gnome. In rete trovate comunque le informazioni per farlo qualora lo si voglia. Pero', usando un ambiente grafico completamente linux si perde l'interoperabilita' con Windows. Se si ha questa esigenza e' ancora consigliabile installare linux separatamente in una partizione ed usare un dual boot.

Client ssh e sua configurazione

Per connettersi a macchine remote (di solito basate su unix), siano esse supercalcolatori posti al CINECA o macchine virtuali dell'Universita' di Trieste, attualmente il sistema piu' semplice e' usare il comando

```
ssh
```

in ambiente linux (sia standalone che WSL) ma anche in ambiente windows.

Il comando

```
ssh nomeutente@macchina.dominio.it
```

vi connettera' alla macchina.dominio.it con il vostro *nomeutente* chiedendovi una password. (eccezione per win: bisogna configurarlo, vedi sotto)

```
    Nota: (nomeutente, password) insieme costituiscono un identificativo univoco c
hiamato "credenziali"
```

Tecnicamente il software `ssh` e' un *client* che si connette ad un *server* sulla macchina remota. Il client ssh puo' essere configurato in modo da facilitare il suo uso.

Il client ssh e' un software che gira sotto linux *e* sotto windows 10 ed e' pre-installato in ambo i sistemi. Siccome e' lo stesso oggetto, si configura nello stesso modo.

Sotto Windows 10 pero', per quanto ho determinato sin'ora, **deve** essere con configurato, per funzionare. Non escludo che sotto win sia possibile usarlo senza la configurazione che segue, solo non ho avuto tempo di evincere come si fa.

In particolare e' molto utile configurare una coppia **public key/private key** che consente accesso immediato alla macchina bersaglio, eventualmente anche *senza* la necessita' di immettere una password. Nota: questo funziona anche su sistemi Mac, che qui non tratto *(tanto Mac fa tutto lui, no?)*.

Questa configurazione si ottiene mediante i seguenti passi:

- creazione delle keys
- configurazione remota per l'accesso dalla macchina locale a quella remota usando le keys
- configurazione locale per l'accesso alla macchina remota usando le keys

Vediamoli.

Creazione delle keys e configurazione remota

Indipendentemente dal sistema operativo utilizzato, il client ssh ha un tool per creare la coppia di keys:

In [ ]:

```
ssh-keygen -t rsa
```

il comando chiedera' un nome di file (diciamo, nomechiave) per le due chiavi ed una passphrase. Se la lasciate vuota (date due enter) la connessione avvera' **senza** password. E' sicura lo stesso (a livello PGP) ma e' talvolta comodo non avere una password, per esempio per potere automatizzare le operazioni di trasferimento via rete. Questo e' un esempio di uso di `ssh-keygen -t rsa` :

```
murante@DESKTOP-DHR78US:~/Scratch$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/murante/.ssh/id_rsa): /home/murante/.ss
h/nomechiave
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/murante/.ssh/nomechiave.
Your public key has been saved in /home/murante/.ssh/nomechiave.pub.
The key fingerprint is:
SHA256:ILSI6keHsN7GChPRwY906ypgWcdOn3O/wgyvOeJ9aZY murante@DESKTOP-DHR78US
The key's randomart image is:
+---[RSA 2048]----+
| .. .            |
| oo+..           |
|ooo++..          |
|..+.+= .         |
|o.o++.. S        |
|+++ o. = .       |
|=o *    B +      |
|o.=  ....E .     |
| o  ...+* ...    |
+----[SHA256]-----+
```

La chiave **pubblica** (nomechiave.pub) va copiata sulla macchina bersaglio, quindi dovete avervi accesso o qualcuno che vi ha accesso deve fare le operazioni seguenti per voi. Supporro' nel seguito che la macchina bersaglio sia un sistema linux/unix.

La chiave pubblica va aggiunta alle chiavi autorizzate sulla macchina bersaglio, contenute nel file /home/nomeutente/.ssh/authorized_keys, con il seguente comando:

In [ ]:

```
cat nomechiave.pub >> authorized_keys
```

e' **assolutamente fondamentale** usare **>>**. Attenzione che usando solo un segno di maggiore cancellereste tutte le chiavi gia' presenti nel file. Anzi e' forse meglio, prima di dare il comando, eseguire un sano `cp authorized_keys authorized_keys.old` che vi consentira' di ripristinare la situazione precedente se qualcosa va storto.

Una volta fatto cio', e' bene dare il comando:

In [ ]:

```
chmod 600 *
```

questo setta i permessi corretti di tutti i files presenti nella directory .ssh

A questo punto, la configurazione della macchina *remota* (o *bersaglio*) e' completata

---

Configurazione locale

Questa parte della configurazione va eseguita nella directory .ssh, che come detto nei sistemi linux/unix (compreso il sottosistema linux di win) si trova in

```
/home/nomeutente/.ssh
```

ed invece nei sistemi windows in

```
C:\Users\Nomeutente\.ssh
```

(ricordo che "Users" potrebbe invece essere "Utenti" in macchine installate in italiano)

In tale directory si deve creare un file nominato

```
config
```

ed aggiungerci la sezione descritta sotto. Se questo file gia' esiste, occorre semplicemente aggiungere in fondo ad esso la stessa sezione.

Con il vostro editor preferito (su windows il blocco note va bene), si devono inserire in config le seguenti linee:

```
Host etichetta_macchina
     Hostname macchina.dominio.it
        User nomeutente
        IdentityFile C:\Users\Nomeutente\.ssh\nomechiave
```

- etichetta_macchina e' l'identificativo che userete con ssh per accedere alla macchina remota
- macchina.dominio.it e' l'indirizzo internet completo della macchina bersaglio, ad esempio anaconda.oats.inaf.it
- nomeutente e' l'utente con cui accedete alla macchina **remota**
- IdentityFile e' la chiave privata che avete generato. E' *nomechiave* **senza** estensione .pub , e deve vivere nella .ssh

Su linux/unix, l'ultima linea va sostituita con

```
        IdentityFile /home/nomeutente/.ssh/nomechiave
```

Sotto linux e' meglio dare ancora `chmod 600 *` dopo aver eseguito l'operazione. L'installazione base di win non ha ACL (Access Control Lists) quindi questo passaggio non e' necessario.

Questo e' un esempio dal mio config, la sezione che mi consente accesso diretto al mio desktop all'Osservatorio dalla macchina di casa:

```
host anaconda
   Hostname anaconda.oats.inaf.it
        User murante
        IdentityFile /home/murante/.ssh/desk2anaconda
```

Io uso la connessione semplicemente con:

```
ssh -XY anaconda
```

Quindi a questo punto, sia da windows che da linux, potrete accedere alla macchina bersaglio con il comando:

In [ ]:

```
ssh -XY etichetta_macchina
```

da linux il comando va dato da dentro il terminale, da windows, da dentro powershell o command prompt. Vi connetterete alla macchina remota, *senza* password se cosi' avete scelto.

Notate lo switch **-XY** aggiunto ad ssh. Questo abilita il *trasporto X*; vuol dire che i comandi che date sulla macchina remota, e richiedono grafica, verranno reindirizzati al vostro server X locale (naturalmente, perche' funzioni dovete averne uno *up-and-running*; vedere sopra). Ad esempio potete aprire una finestra grafica con gnuplot.

Alcune informazioni utili: potrebbe servire aggiungere all'inizio del file config, le linee

```
Host *
  ServerAliveInterval 60

IdentitiesOnly yes
```

le prime due linee servono ad evitare la disconnessione automatica della sessione se non la si usa per un certo tempo. IdentitiesOnly rinforza il check sulle identita' descritte nel config

Per semplicita' d'uso, sotto linux potete aggiungere questa linea

```
alias ssh="ssh -XY"
```

al vostro .bashrc. Se lo fate, siccome *-XY* verra' aggiunto di sistema, potete evitarvi il *faticosissimo* sforzo di digitarlo ogni volta (ricordate: i fisici e gli informatici sono pigri, lasciamo perdere un fisico informatico). Cioe', `ssh etichetta_macchina` funzionera' come (in effetti, **sara'**) `ssh -XY etichetta_macchina`

Semplici comandi utili di ssh

Comandi utili di ssh, a parte la connessione `ssh -XY etichetta_macchina` , sono:

In [ ]:

```
scp etichetta_macchina:/path/to/a/file/nomefile .
```

questo copia il file chiamato nomefile, che sta sulla macchina remota, nella directory dove vi trovate sulla macchina locale. Naturalmente varrebbe anche

```
scp etichetta_macchina:/path/to/a/file/nomefile /full/path/to/a/local/dir
```

che copierebbe nomefile nella directory locale /full/path/to/a/local/dir

In [ ]:

```
scp -r etichetta_macchina:/path/to/a/dir/directory .
```

questo copia l'intera *directory* che si trova in /path/to/a/dir sulla macchina remota, nella directory dove vi trovate in locale. Esempio:

```
scp -r anaconda:/home/murante/giocattolo .
```

copierebbe dalla macchina "anaconda" l'intera directory "giocattolo" sulla mia macchina locale, tipo, il desktop che ho a casa, ed i files sarebbero nella directory "giocattolo", non nella mia home.

In [ ]:

```
rsync -av [--exclude some-files] etichetta_macchina:/path/to/a/dir/directory .
```

Stesso comportamento che nel caso precedente, con una differenza fondamentale. `scp` copia **sempre** tutta la directory richiesta, dall'inizio alla fine e ripartendo sempre dall'inizio. `rsync` invece **sincronizza** le macchine, quindi se per problemi di rete la copia si interrompe, ridando il comando ripartite col trasferimento dall'ultimo file valido trasferito. Allo stesso modo se alcuni files gia' copiati in locale sono intanto cambiati sulla macchina remota, essi vengono copiati di nuovo.

Attenzione

```
se date
rsync -av etichetta_macchina:/patho/to/a/dir/directory/* .
il risultato sara'  che invece di avere i files copiati in una nuova "directory",
li avrete copiati nella directory dove siete, incasinando tutto terribilmente.
Come per il C, i comandi unix immaginano che sappiate cosa state facendo.
```

mi spiego meglio

```
rsync -av etichetta_macchina:/home/nomeutente/giocattolo .
crea (se gia' non c'e') la directory "giocattolo" SOTTO la dir dove siete adesso,
 per esempio la home,
e ci mette i files che arrivano dalla macchina remota
Alla fine nella vostra home avrete una directory "giocattolo" dentro la quale potr
ete andare e vedere
tutti gli scintillanti nuovi files
```

invece se date

```
rsync -av etichetta_macchina:/home/nomeutente/giocattolo/* .
```

i nuovi files continueranno a scintillare sulla vostra macchina locale ma saranno, nell'esempio di prima,

```
direttamente sotto la home, non nella directory "giocattolo" che NON verra' creat
a,
```

e questo incasinera' notevolmente la vostra home. Cio' vale anche per scp, `scp -r /home/nomeutente/giocattolo/* .` sbatterebbe tutti i files presenti nella directory remota "giocattolo" nella mia *home* incasinandola - dipendendo dal numero di files nella dir remota "giocattolo", in modo piu' o meno catastrofico.

---

rsync si usa per spostare grosse moli di dati. Lo switch tra parentesi quadre puo' essere usato per *escludere* file non desiderati dalla sincronizzazione.

Ogni comando descritto funziona anche al contrario, cioe' potete scrivere ad esempio

In [ ]:

```
scp -r local_dir etichetta_macchina:/path/to/the/place/where/you/want/to/put/it
```

questo copiera' la *vostra* local_dir **dalla** macchina locale **alla** macchina remota.

Ricordate che - sotto linux/unix - potete trovare tutte le informazioni necessarie sui comandi qui accennati con:

```
man ssh
man scp
man rsync
```

Il nuovo terminale MS windows

(per questa sezione ringrazio Fabio Busoni: abbiamo scoperto il terminale MS insieme)

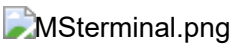
Recentemente MicroSoft ha rilasciato il suo nuovo terminale, di cui suggerisce l'installazione (opzionale) quando si installa WSL. Ho provato l'oggetto e ne sono rimasto convinto. Suggerisco la sua installazione. Nel seguito descrivo configurazione e semplici modi di uso.

La prima cosa da fare e' installare il software. Lo trovate in MS store ed e' gratis. Non sto a dirvi come fare.

Se lo fate partire, e' configurato per aprire una powershell. Iniziamo a guardarlo. Se date ctrl-shift-T, aprira' una nuova *tab*. Trovate le tabs nella barra in alto e passate dall'una all'altra cliccando sulla loro porzione di barra. Se ne aprono di nuove anche cliccando su "+" nella barra in alto. Ogni tab e' un terminale indipendente.

Se date fctn-shift-D, il terminale si dividera' in due *pane*. Lo fa sempre in modo da rendere uguale l'area dei pane creati. Provate a farlo a ripetizione.

I pane sono senz'altro una delle cose migliori di questo oggetto. Considerate che sono tutti indipendenti - in realta' ciascuno e' un terminale a se stante - e potete chiuderli indipendentemente (date "exit" dentro ciascuno di loro) per creare la vostra configurazione geometrica preferita. E' molto carino, ma a parte sembrare (**essere**?)un videogioco, per esempio potete tenere in mezzo terminale il file sorgente su cui lavorate, e nell'altro mezzo la directory dove lo compilate ed eseguite. Puo' sembrare sciocco, ma questi dettagli aiutano a lavorare meglio.


MSterminal.png

```
nota: lo stesso effetto si puo' ottenere in emacs, quando si editano diversi files
sorgenti ad esempio,
con ctrl-X 3 e/o ctrl-X 2
emacs e' un software gnu e come tale, la sua prima versione e' del 198
6..............
(la prima versione in assoluto e' del 1976, quarantaquattro anni fa)
```

Il terminale ha un file di configurazione che si chiama

```
settings.json
```

e sta nella directory

```
C:\Users\Nomeutente\AppData\Local\Packages\Microsoft.WindowsTerminal_8wekyb3d8bbwe
\LocalState
```

Attenzione la directory (cartella) AppData e' *nascosta*

Per vederla dovete andare, in esplora risorse, dentro "visualizza"

e la' abilitare

"elementi nascosti" (terzultima tab da destra)

Il codice esadecimale dopo `Microsoft.WindowsTerminal_` potrebbe essere diverso.

Questo terminale e' completamente configurabile. Dal menu' a tendina cui accedete con il "v", potete aprire il file di configurazione ("settings"). Nel seguito fornisco le istruzioni per configurarlo per usare direttamente la vostra distro WSL linux, e per connettersi automaticamente a macchine remote.

Ammetto di gradire la configurabilita' fatta cosi' che in teoria consente captive terminals (terminali dentro i quali gira un software che decidete voi), multiple-distro, multiple-commands, e molto altro.

---

la configurabilita' del terminale e' *secondo me* la sua caratteristica migliore. Il file di configurazione e' un file di testo e la sintassi e' acquisibile in cinque minuti da qualunque programmatore/sistemista. Qui pero' vi suggerisco alcuni trucchi.

Il seguente file e' il mio settings.json

In [ ]:

```
// This file was initially generated by Windows Terminal 1.3.2651.0
// It should still be usable in newer versions, but newer versions might have additional
// settings, help text, or changes that you will not see unless you clear this file
// and let us generate a new one for you.

// To view the default settings, hold "alt" while clicking on the "Settings" button.
// For documentation on these settings, see: https://aka.ms/terminal-documentation
{
    "$schema": "https://aka.ms/terminal-profiles-schema",

    "defaultProfile": "{61c54bbd-c2c6-5271-96e7-009a87ff44bf}",

    // You can add more global application settings here.
    // To learn more about global settings, visit https://aka.ms/terminal-global-settings
    "wordDelimiters": " `\\()\"'-:,;<>~!@#$%^&*|+=[]{}~?|",

    // If enabled, selections are automatically copied to your clipboard.
    "copyOnSelect": false,

    // If enabled, formatted data is also copied to your clipboard
    "copyFormatting": false,

    // A profile specifies a command to execute paired with information about how it should
    // Each one of them will appear in the 'New Tab' dropdown,
    //   and can be invoked from the commandline with `wt.exe -p xxx`
    // To learn more about profiles, visit https://aka.ms/terminal-profile-settings
    "profiles":
    {
        "defaults":
        {
            // Put settings here that you want to apply to all profiles.
        },
        "list":
        [
            {
                // Make changes here to the powershell.exe profile.
                "guid": "{61c54bbd-c2c6-5271-96e7-009a87ff44bf}",
                "name": "Linux",
                "commandline": "bash.exe -c \"cd $HOME; bash.exe\" ",
                "hidden": false
            },
            {
                // Make changes here to the powershell.exe profile.
                "guid": "{61c54bbd-c2c6-5271-96e7-009a87ff4400}",
                "name": "Windows Power Shell",
                "commandline": "powershell.exe",
                "hidden": false
            },
            {
                // Make changes here to the cmd.exe profile.
                "guid": "{0caa0dad-35be-5f56-a8ff-afceeeaa6101}",
                "name": "Command Prompt",
                "commandline": "cmd.exe",
                "hidden": false
            },
            {
                "guid": "{2c4de342-38b7-51cf-b940-2309a097f518}",
                "hidden": false,
                "name": "Ubuntu",
```

```
                "source": "Windows.Terminal.Wsl"
            },
            {
                "guid": "{2c4de342-38b7-51cf-b940-2309a097f520}",
                "hidden": false,
                "name": "Marconi100",
        "commandline": "ssh -XY m100"
            },
            {
                "guid": "{2c4de342-38b7-51cf-b940-2309a097f522}",
                "hidden": false,
                "name": "anaconda",
        "commandline": "ssh -XY anaconda"
            }
        ]
    },

    // Add custom color schemes to this array.
    // To learn more about color schemes, visit https://aka.ms/terminal-color-schemes
    "schemes": [],

    // Add custom actions and keybindings to this array.
    // To unbind a key combination from your defaults.json, set the command to "unbound".
    // To learn more about actions and keybindings, visit https://aka.ms/terminal-keybindir
    "actions":
    [
        // Copy and paste are bound to Ctrl+Shift+C and Ctrl+Shift+V in your defaults.json.
        // These two lines additionally bind them to Ctrl+C and Ctrl+V.
        // To learn more about selection, visit https://aka.ms/terminal-selection
        { "command": {"action": "copy", "singleLine": false }, "keys": "ctrl+c" },
        { "command": "paste", "keys": "ctrl+v" },

        // Press Ctrl+Shift+F to open the search box
        { "command": "find", "keys": "ctrl+shift+f" },

        // Press Alt+Shift+D to open a new pane.
        // - "split": "auto" makes this pane open in the direction that provides the most s
        // - "splitMode": "duplicate" makes the new pane use the focused pane's profile.
        // To learn more about panes, visit https://aka.ms/terminal-panes
        { "command": { "action": "splitPane", "split": "auto", "splitMode": "duplicate" },
    ]
}
```

..potete copiarlo ed incollarlo e *circa* funzionera' (ma solo se avete installato il sottosistema linux).

Le cose importanti di questo file di configurazione:

- la prima parte del file contiene le configurazioni generali del vostro terminale
- l'hex che c'e' in "default profiles" e' il componente di "lists" che parte all'avvio del terminale, indentificato dal suo "guid"
- occhio a "**wordDelimiters**". Qui mettete i caratteri che definiscono una '**parola**', che e' quello che selezionate col doppio click. Tutti i caratteri tra apici, ad esempio, '(' o ')', delimitano una parola. Loro inizieranno e chiuderanno la parola e non vi saranno inclusi. Attenzione agli escape: per esempio la sequenza \\ dentro gli apici vuol dire che '\' e' un delimitatore. Questo e' necessario perche' alcuni caratteri, come '\' o ", non si possono includere cosi come sono dentro una stringa perche' hanno un loro significato. Chiedetemi se non chiaro che questo e' tricky. Giocateci. Cambiate i delimitatori e guardate cosa cambia quando fate doppio click su un testo in un terminale. Per esempio provate a togliere tutto tranne lo spazio...

- dentro "profiles", e sotto "list", *ogni blocco {} identifica una entrata nel menu' a tendina*
- il blocco che ha la guid corrispondente a quella segnata in "**defaultProfile**" definisce come parte il terminale al doppio click da windows, partira' come descritto dal corrispondente blocco {} in "list"
- le varie configurazioni del terminale sono dettagliate nella lista "list".
- nel caso del mio file di configurazione, ho scelto di partire con il blocco che contiene `"commandline": "bash.exe -c "cd $HOME; bash.exe\" "` . Vuol dire che parto col sottosistema linux (bash.exe) nella sua directory home. Questo e' il mio default, cioe' come parte il mio terminale, perche' la "guid" e' quella scritta in "defaultProfile" sopra. Giocate a cambiare il default profile cambiando soltanto il codice esadecimale.
- **attenzione**: se aggiungete un blocco {} dentro list, il suo hex (quello scritto in guid) deve essere univoco (diverso dagli altri hex presenti nel file)
- l'*argomento di "commandline"* e' il **programma che esegue il terminale quando si avvia** (data la scelta del menu' a tendina). Potete metterci cio' che volete, anche un captive (un codice scritto apposta da voi)
- i tre blocchi successivi al primo, nell'elenco "list", sono di default ed in particolare aprono una **power shell** ed un **command prompt**. Il quarto apre di nuovo un linux, ma notate quale e' la working directory se non la definisco a mano. Questo capita perche il terminale parte di suo da `C:\User\Nomeutente`
- i due ultimi blocchi lanciano **direttamente** da windows una *connessione ssh ad una macchina remota*, nell'esempio, m100 ed anaconda. Tali connessioni ssh devono essere state configurate come descritto sopra. Una volta fatto cio' il terminale MS si connettera' **direttamente** alla macchina remota, usando la etichetta_macchina definita nel config di ssh, con, se sono state seguite le istruzioni di questo primer, il trasporto X abilitato. Si noti che questa connessione *non* passa attraverso il sottosistema linux.
- ovviamente dietro corretta configurazione di ssh, si possono avere tante connessioni remote quante se ne desiderino. Sostanzialmente dopo avere configurato l'accesso ad una macchina remota, come descritto sopra, potete aggiungere il suo blocco e vi accederete direttamente dal terminal

Il terminale consente inoltre molti effetti speciali con cui sbizzarrirsi, sfondi, trasparenze ed altro. Tuttavia le funzionalita' fondamentali sono quelle elencate sopra.

```
Nota. il doppio click su una parola (o l'uso del mouse) EVIDENZIANO un testo.
Questo si copia negli appunti cliccando col tasto destro del mouse.
Si incolla (nello stesso terminale, o in un altro) cliccando di nuovo col tasto de
stro.
Cio' vale anche per il terminale linux.
```

Anche se personalmente consiglio di installare il WSL, devo comunque far notare che la configurazione del client ssh + server X + terminal MS consente, se si accede ad una macchina per esempio di units, di lavorare su un sistema linux remoto in maniera trasparente e qundi *senza* dovere avere installato linux/unix sulla propria macchina.

Credits

Questo primer e' fornito as-is, se qualcosa che c'e' scritto non funziona, *o fa danni*, non me ne assumo responsabilita' :-) . Se volete, segnalate inesattezze od omissioni. Contattatemi pure presso giuseppe.murante@inaf.it (mailto:giuseppe.murante@inaf.it)

In [ ]: