sage

durrendal

Sage is a little shell script I wrote to make managing multiple ssh keys easier. It's pretty simple in nature, but honestly massively helpful if you happen to use password protected ssh keys with strong passwords, and have a nice cli based password manager like `pass`. I imagine that you could sub pass for the bitwarden cli, lastpass cli, or something similar, so long as it can return the credential needed to unlock you key.

Here's the script in all of it's glory, short sweet and to the point!

EDITOR'S NOTE: Where lines in the code snippet below would run off the edge of the page, I have wrapped the line, continuing it with a `->>`. When you see this at the beginning of a line, understand that there is not a literal line break; what you are seeing is a continuation of the previous line. -ed

```
#!/bin/sh
#ssh-agent management script, uses a profile hook to ensure the agent
#exists between sessions, and integrates with pass to unlock ssh keys
#protected with passphrases.

#On Alpine Linux you'll need these packages installed
#apk add util-linux-misc openssh-client-common procps-ng pass sed

#To persist ssh-agent between terminals, add this to ~/.profile.
#Otherwise honestly, this won't work.
#export SSH_AUTH_SOCK=~/.ssh/ssh-agent.$HOSTNAME.sock
#ssh-add -l 2>/dev/null >/dev/null
#if [ $? -ge 2 ]; then
#   ssh-agent -a "$SSH_AUTH_SOCK" >/dev/null
#fi

keys=$@

if { [ -z $1 ]; }; then
    echo "Usage: sage [key]"
    exit 1
elif [ "$1" == "-l" ]; then
    printf "Active Keys:
$(ssh-add -l)

Protected Keys:
$(pass show ssh)
"
```

```
    exit 0
else
    #For each key passed
    for key in $keys; do
        #Check if it's password protected
        protected=$(ssh-keygen -y -P "" -f ~/.ssh/$key 2>&1
        ->>  | grep -o "incorrect passphrase supplied")
        #If it is, "" will not be a valid password
        if [ "$protected" == "incorrect passphrase supplied" ]; then
            #Use script to pass in credentials from pass
            # to a subshell running ssh-add
            { sleep .3; pass ssh/$key; }
            ->>  | script -q /dev/null -c 'DISPLAY= ssh-add ~/.ssh/'$key''
        else
            #Otherwise we can just load the key
            ssh-add ~/.ssh/$key
        fi
    done
fi
```

Now the way this works is by combining our profile settings with the script. When we add this snippet to your .profile or .bash_profile it'll ensure that the ssh-agent is running whenever you open a terminal. If it's already running it just quietly continues.

```
export SSH_AUTH_SOCK=~/.ssh/ssh-agent.$HOSTNAME.sock
ssh-add -l 2>/dev/null >/dev/null
if [ $? -ge 2 ]; then
    ssh-agent -a "$SSH_AUTH_SOCK" >/dev/null
fi
```

The only reason that works is becuase we're exporitng SSH_AUTH_SOCK to a specific static path, normally ssh-agent would just make a random temporary one in /tmp, but doing it this way ensures that the agent communicates the same way each time.

After that we just add our keys and the little {command; command;} piped argument catches the interaction from our password manager and brokers it to the ssh key credential prompt. Here let me show you, we'll add my primary key!

```
~|>> sage neuro
Enter passphrase for /home/durrendal/.ssh/id_ed25519:


-----------------------------------------------------------------
| Please enter the passphrase to unlock the OpenPGP secret key: |
| "Durrendal <...@...>"                                         |
| 4096-bit RSA key, ID ..............,                         |
| created 2023-11-19 (main key ID ..............).             |
|                                                               |
|                                                               |
| Passphrase: _________________________________________________ |
|                                                               |
| <OK>                                              <Cancel>    |
-----------------------------------------------------------------
```

```
Identity added: /home/durrendal/.ssh/id_ed25519 (durrendal@neuromancer)
```

Et voila! By virtue of unlocking my password manager I can import my ssh key into the agent. Now when my keys are at rest I don't have to worry, the passwords to use them can even be absolutely gnarly long random strings generated by pwmake, like this:

```
~|>> pwmake 256
oqkIkASPYms3b=ip%0GitISs4symJ@HJeKFOrJ@c93lYByM1Uk@jIG
```

It feels good to know that my keys are more secure while at rest, and I can utilize a modern authentication workflow to unlock them. Hopefully someone else finds this useful too!