

远程 SSH 配置指南

对于一些 CPU 性能较差的机载计算机，使用远程桌面如 NoMachine 进行远程操作是十分占用性能的。但是，在终端中使用传统 SSH 连接又很不方便：需要输入 IP，进程与窗口绑定，不能使用多开终端/窗口。因此有必要进行优雅而实用的远程 SSH 连接。

准备工作

0. 确保机载计算机和你的主机能够自动地连接到同一个局域网

1. 使用主机名解析对方 IP

虽然将两个路由器设置为同样的名称和密码计算机就能自动连接它们，但 IP 是会变化的。首先安装：

```
sudo apt install avahi-daemon
systemctl status avahi-daemon # 检查运行状态
```

然后设置主机名：

```
sudo hostnamectl set-hostname iusl-xavier-nx
```

这里 iusl-xavier-nx 是你相设置的主机名。设置后需要重启。

如果设置成功，则运行：

```
hostname
```

就能看见自己的主机名是否正确了。

- 注意，局域网内主机名不能冲突，因此设置时谨慎，考虑其他计算机是否在使用该名称。
- 事实上，启动终端时显示的前缀。例如 zzy@zzy-Legion-R9000P-ARX8，这里 zzy 是账户名，而 zzy-Legion-R9000P-ARX8 就是主机名）

现在可通过如下方式测试能否 ping 通对方：

```
ping iusl-xavier-nx.local
```

实际上，.local 方式就解析了对方的 IP。

2. 在机载计算机和你的主机上都安装 SSH 服务

依次执行下面命令即可：

```
sudo apt install openssh-server
sudo systemctl start ssh
sudo systemctl enable ssh
```

3. 安装开源 SSH 客户端 Tabby

Tabby 是一款现代化的终端连接工具，开源并且跨平台。当然你用别的工具也行，甚至直接用终端 ssh 也行（非常不推荐），但就我个人的体验来说，Tabby 确实非常好用且美观。

```
https://github.com/Eugeniy/tabby
```

下载 deb 安装即可。对于 Ubuntu 20.04 系统，请在 Release 中下载 v1.0.216 版本。经测试，更高的版本在 20.04 中会报 NodeJS 相关的错误，无法启动。Windows 下应该无所谓。

启动后应该长这样：

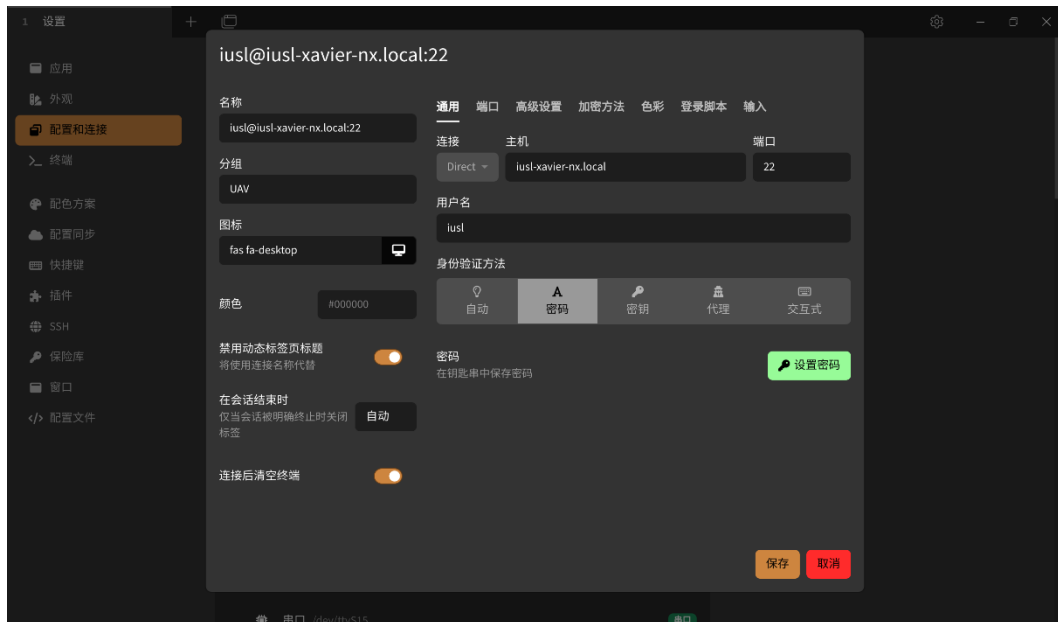


原生支持简体中文，如果没有自动显示为中文则在设置中更改一下就好。

具体步骤：

1. 设置与机载计算机的连接

在 Tabby 中，点击设置-配置与连接-新建-新配置，在弹出的下拉列表中选择“SSH 连接”。



连接：Direct（直连）。

主机：iusl-xavier-nx.local（即：你的主机名.local）

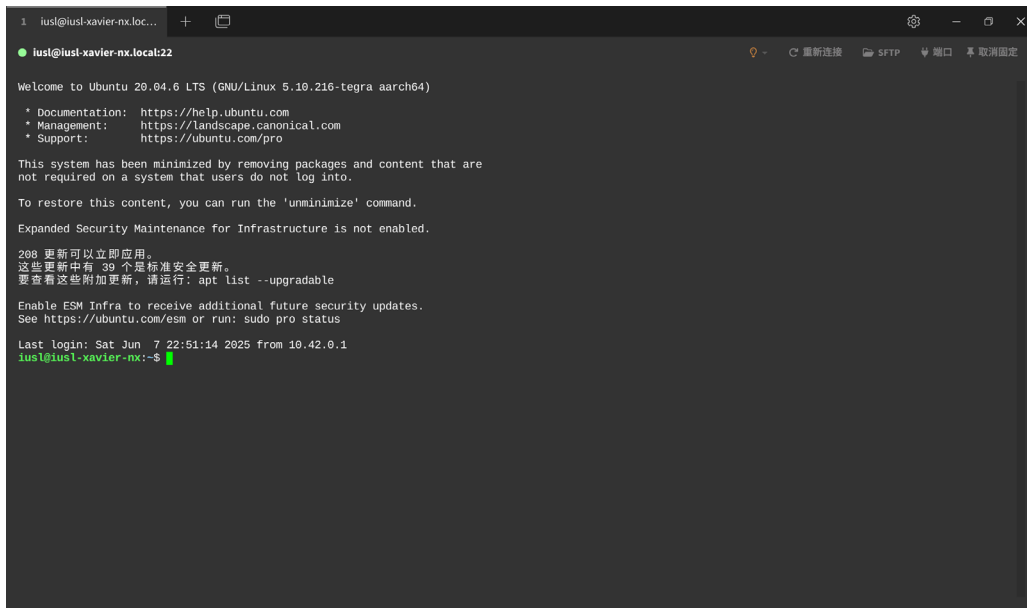
端口：22，不要更改。

用户名：你机载计算机的账户名。

身份验证方法：选择“密码”，设置密码为你机载计算机的密码，并保存。

设置完之后保存即可。

关掉配置页，回到首页，点击“配置与连接”，选择你刚才新建好的 SSH 连接即可。
如果一切正常，则显示：



```
iusl@iusl-xavier-nx.local:22
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.10.216-tegra aarch64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.


To restore this content, you can run the 'unminimize' command.

Expanded Security Maintenance for Infrastructure is not enabled.

208 更新可以立即应用。
这些更新中有 39 个是标准安全更新。
要查看这些附加更新，请运行：apt list --upgradable

Enable ESM Infra to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

Last login: Sat Jun  7 22:51:14 2025 from 10.42.0.1
iusl@iusl-xavier-nx:~$
```

你也可以开启多个窗口。注意，默认情况下窗口旁边的“+”号开启的是自己电脑的终端！
请按旁边的按键，选择刚才新建的 SSH 配置，即可开启机载计算机上的终端页面。
或者你可以在设置-配置与连接-配置中，将“新标签页的默认设置”设置为刚才的 SSH 配置。

- 请注意，这里的窗口与终端窗口类似，如果断开则任务会被中断，因此请注意保持网络连接顺畅。

如果要上传或编辑文件，点击 SFTP 即可打开文件管理器，对于小文件可直接右键-本地编辑，即可在本地编辑保存后自动同步到远端。（注意，需要 sudo 权限编辑的文件不能这样做，保存结果不会生效，你只能在 SSH 中用 vim 编辑）

2. 设置 tmux 终端复用器

为了：(1)能够在 SSH 中配置多开终端页面，(2)防止窗口误关闭导致终端中的进程被杀死，我们使用 tmux 来执行终端命令。

通过以下指令安装：

```
sudo apt install tmux
```

运行 tmux 就可以看见 tmux 会话出现了。

- tmux 的逻辑是基于会话(session)-窗口(window)-窗格(pane)的，一个会话是一组窗口的组合，一个窗口是一组窗格的组合（即窗口分屏），在窗格中可以执行终端命令。tmux 的会话与终端无关，即使你关闭了终端页面（或者断开 SSH 连接），会话中的命令仍然在后台运行（换言之，终端窗口只是用来展示 tmux 会话的界面罢了）。

如果要真正的结束所有 tmux 会话，则需要（在任意终端或窗口中）输入指令：

```
tmux kill-server
```

或者在激活的 tmux 终端中按下 ctrl+b，然后输入英文冒号，然后输入 kill-server 按回车。

关于 **tmux** 的用法有很多，也很实用，这里不展开讲解。为了方便使用（尤其是方便鼠标操作），需要配置 **tmux** 的配置文件。在 SSH 中执行：

```
vim ~/.tmux.conf
```

然后将以下内容复制进去（按 **i** 进入编辑模式，然后 **ctrl+shift+v** 复制，然后按 **esc** 离开编辑模式，最后依次输入 **:wq**，按回车退出）：

```
# ~/.tmux.conf

# 启用鼠标模式，允许用鼠标选择窗格、调整大小、滚动历史记录
set -g mouse on

# 启用 256 色支持，让 vim 等程序的颜色更好看
set -g default-terminal "screen-256color"

# 状态栏设置
set -g status-bg black          # 状态栏背景色
set -g status-fg white         # 状态栏前景色
set -g status-interval 1       # 状态栏刷新闻隔（秒）
set -g status-justify centre   # 窗口列表居中

# 左侧状态栏内容：会话名
set -g status-left '[fg=green]S:#S [fg=white]| '

# 右侧状态栏内容：主机名 | 日期 | 时间
set -g status-right '[fg=white]| [fg=cyan]H:#H [fg=white]| [fg=yellow]%Y-%m-%d %H:%M'

# 窗口列表样式
set-window-option -g window-status-current-style bg=red,fg=white # 当前窗口样式
set-window-option -g window-status-format ' #I:#W '              # 非当前窗口格式
                              （编号:名字）
set-window-option -g window-status-current-format ' #I:#W '      # 当前窗口格式
```

关于 **tmux** 的配置网上也有很多，这里配置了一些实用的功能，你也可以用自己的配置。现在 **tmux** 就可以正常使用了。我已经写好了适用于 **tmux** 的示例启动脚本 **example.sh**：

```
#!/bin/bash

#
=====
===
# 配置区（命令索引从 0 开始，按需求配置即可，窗口会自动适应）
#
=====
===

# --- 会话名称定义 ---
SESSION_NAME="main"
```

```

# --- 命令定义 ---
COMMANDS=(
# -- [0] -- #
' jtop '
# -- [1] -- #
' bash -c "sudo chmod 777 /dev/ttyACM0; roslaunch mavros px4.launch; exec
bash" '
# -- [2] -- #
' bash -c "roslaunch realsense2_camera rs_camera.launch --wait; exec bash" '
# -- [3] -- #
' bash -c "sleep 5; roslaunch vins vins.launch --wait; exec bash" '
# 如需更多命令，按上述形式附加在数组中即可
)

#
=====
===
# 执行区（如无必要，请勿改动）
#
=====
===

# 获取命令数量
readonly NUM_COMMANDS=${#COMMANDS[@]}

if [ $NUM_COMMANDS -eq 0 ]; then
    echo "命令数组为空。"
    exit 1
fi

# 自适应窗口分割
tmux new-session -d -s $SESSION_NAME
for i in $(seq 1 $(( $NUM_COMMANDS - 1 ))); do
    if [ $(( $i % 2 )) -ne 0 ]; then #
        tmux split-window -h -t $SESSION_NAME:0
    else
        tmux split-window -v -t $SESSION_NAME:0
    fi
    tmux select-layout -t $SESSION_NAME:0 tiled
done

# 命令发送
for i in $(seq 0 $(( $NUM_COMMANDS - 1 ))); do
    PANE_INDEX=$i
    tmux send-keys -t $SESSION_NAME:0.$PANE_INDEX "${COMMANDS[$i]}" C-m
done

# 附加到终端
tmux attach-session -t $SESSION_NAME

```

- 你可能注意到，上述命令中我在启动 MAVROS 前执行了 `sudo chmod 777 /dev/ttyACM0` 为串口赋权限。为了使这个指令无需输入密码自动执行，你可以：

1. 执行 `sudo visudo`
2. 在打开的文件中的最后一行添加（以 vim 的方法）：

```
iusl ALL=(ALL) NOPASSWD: ALL
```

这里 `iusl` 替换为你的用户名。

- 你可能还注意到，这里的 `roslaunch` 后有个参数—`wait`，强烈建议保留，`sleep` 可能不是特别准确，导致不同的 `launch` 几乎同时启动，注册多个 `master` 导致 `run_id` 冲突报错。
- 你可能根本注意不到，在 MAVROS 中，桥接 QGC 的地址也是可以用 `.local` 方式来指定的。例如说，你可以将 `px4.launch` 中的这两个参数进行如下更改：

```
<arg name="fcu_url" default="/dev/ttyACM0:20000000" />
<arg name="gcs_url" default="udp://@zzy-Legion-R9000P-ARX8.local:14550" />
```

其中 `ttyACM0` 是实际连接的串口名称（如果你用 USB 连接，那么走 USB-CDC-ACM 协议，波特率无所谓），`zzy-Legion-R9000P-ARX8` 是你想要运行 QGC 的主机名称（不建议用 UDP 桥接 QGC 时进行传感器校准哦）。

只需要在 `COMMANDS` 中增减你的命令（以换行分隔），该脚本就能自动为你分割窗口并执行。上述命令的执行效果为：

现在你可以试试关掉这个 SSH 连接窗口，再新开一个，然后执行：

```
tmux ls
```

可见你的 `main` 会话还在执行。然后重新显示它：

```
tmux at -t main
```

你的会话又回来了。这说明 `tmux` 的终端与 SSH 窗口无关，只要你不 `kill` 它，它是永远存在的，即使网络断开导致 SSH 窗口消失了也不影响。

你也可以试试在以上脚本中增添命令，窗口布局会怎样变化。

4. 配置 ROS 主从机（可选）

只适用于单无人机。对于 ROS1 的通讯而言，其依赖于 **master** 的机制使多机通讯很脆弱。

虽然有部分第三方解决方案，不过目前看来效果一般，不易实现。

一般来说，设置你的无人机为 ROS 主机，你的笔记本为 ROS 从机。

在无人机机载电脑上编辑 `.bashrc`，在最后添加下面两行：

```
export ROS_MASTER_URI=http://iusl-xavier-nx.local:11311
export ROS_HOSTNAME=iusl-xavier-nx.local
```

这里将 `iusl-xavier-nx` 替换为你机载计算机的主机名。

然后在你笔记本上编辑 `.bashrc`，在最后添加下面这段：

```
# =====
# ROS 多机环境切换函数
# =====

ROS_SESSION_FILE=~/.ros_session_mode
LOCAL_HOSTNAME="$(hostname).local"

function ros_connect() {
    if [ -z "$1" ]; then
        echo -e "\033[31m[错误] 请提供目标机器人的主机名。 \033[0m"
        echo "用法: ros_connect <robot_hostname>"
        return 1
    fi

    TARGET_HOST="$1"
    # 设置当前终端的环境变量
    export ROS_MASTER_URI=http://${TARGET_HOST}.local:11311
    export ROS_HOSTNAME=${LOCAL_HOSTNAME}
    echo "${TARGET_HOST}" > "${ROS_SESSION_FILE}"

    _update_ros_prompt
}

function ros_local() {
    unset ROS_MASTER_URI
    unset ROS_HOSTNAME

    rm -f "${ROS_SESSION_FILE}"
    _update_ros_prompt
}

if [ -z "$regular_PS1" ]; then
    regular_PS1=$PS1
fi

function _update_ros_prompt() {
```

```

if [[ -n "$ROS_MASTER_URI" && "$ROS_MASTER_URI" != *"localhost"* ]]; then
    ROS_MODE_PROMPT="(\\033[32mROS 从机模式\\033[0m) "
else
    ROS_MODE_PROMPT=""
fi
PS1="${ROS_MODE_PROMPT}${regular_PS1}"
}

PROMPT_COMMAND="_update_ros_prompt"

if [ -f "${ROS_SESSION_FILE}" ]; then
    SAVED_HOST=$(cat "${ROS_SESSION_FILE}")
    if [ -n "$SAVED_HOST" ]; then
        ros_connect "$SAVED_HOST" > /dev/null
    fi
fi

_update_ros_prompt
# =====

```

这是为了在需要的时候将本机设置为从机（因为从机不能单独运行 `roscore`，如果想在本机上运行其他 ROS 程序则又需要配置）。

想切换到 ROS 从机模式时，打开终端执行：

```
ros_connect iusl-xavier-nx
```

将 `iusl-xavier-nx` 替换为你想连接的机载计算机的 `hostname`。切换后终端提示符前会显示“ROS 从机模式”。

想切换回主机模式时，打开终端执行：

```
ros_local
```

5. 使用 VSCode Remote SSH 远程编辑代码（可选）

为了避免反复拷贝代码的麻烦，可以使用 VSCode 的 SSH 功能远程编辑代码。

参加 <https://developer.aliyun.com/article/1626959> 教程进行配置即可，唯一的区别是，在填写 `ssh` 命令时不用输 `ip` 地址，而是：

```
ssh iusl@iusl-xavier-nx.local
```

其中 `iusl` 是账户名，`iusl-xavier-nx.local` 是 `hostname`。