Automated Planning Lab Setup Guide

Windows + WSL + Ubuntu + VS Code + planutils + planners

Prof. Adrián Domínguez - Automated Planning - Universidad de Alcalá

This guide explains how to set up the working environment for the *Automated Planning* course.

By the end of this guide, you will be able to:

- Use Ubuntu on Windows via WSL
- Install and configure Python tooling
- Install Apptainer
- Install `planutils` and some planners
- Work remotely from VS Code
- Create and solve a simple PDDL planning problem

Table of Contents

---

1. Install WSL and Ubuntu

WSL (Windows Subsystem for Linux) allows you to run a real Linux environment directly on Windows. Most automated planning tools are developed for Linux and are easier to install and use in Ubuntu than in native Windows.

Steps

1. Open **Windows PowerShell**.
2. Install WSL:

```
wsl --install
```

3. Restart your computer if requested.
4. After reboot, you need to install an Ubuntu distribution in WSL:

```
wsl --install Ubuntu
```

5. After installation, it will ask you to create an username and password. Once created, it will automatically start an Ubuntu terminal.
6. Once the Ubuntu distribution is installed in WSL, to start a new Ubuntu terminal at any time just launch the WSL executable in Windows.

---

2. Update Ubuntu and Install Python

Python's `venv` module allows you to create isolated environments for Python projects. This is important to avoid conflicts between different projects and to ensure that the dependencies for the course are managed properly.

Steps

1. Update package lists:

```
sudo apt update
```

2. Install venv:

```
sudo apt install -y python3-venv
```

---

3. Install Apptainer

Many planners distributed through `planutils` run inside containers. Apptainer allows executing these containers securely without requiring Docker or root-level daemons.

Steps

1. Install prerequisites:

```
sudo apt install -y software-properties-common
sudo add-apt-repository -y ppa:apptainer/ppa
```

2. Install and verify Apptainer:

```
sudo apt update
sudo apt install -y apptainer
apptainer --version
```

4. Create Workspace and Virtual Environment

A dedicated workspace keeps all planning labs organized. The virtual environment ensures that Python dependencies used in the course remain isolated and reproducible.

Steps

1. Navigate to home directory and create the workspace folder:

```
cd ~
mkdir LabsPDDL
cd LabsPDDL
```

2. Create a virtual environment:

```
python3 -m venv venv
```

3. Activate the virtual environment:

```
source venv/bin/activate
```

Your terminal prompt should now start with `(venv)`.

---

5. Install planutils and some planners

- `planutils` provides a unified interface to install and run classical planners.
- FF is a widely used heuristic forward-search planner studied in the course.

Steps

1. Install `planutils`:

```
pip install planutils
```

2. Verify installation:

```
planutils --help
```

3. Install ff planner:

```
planutils install ff
```

4. Install lama-first planner:

```
planutils install lama-first
```

5. List installed planners:

```
planutils list
```

---

6. Install VS Code and WSL Extension

VS Code provides a modern development environment. The WSL extension allows editing files and running tools inside Ubuntu directly from Windows, without manual file transfers.

Steps

1. Download and install **Visual Studio Code (Windows)** from https://code.visualstudio.com
2. Open VS Code.
3. Open Extensions (`Ctrl+Shift+X`).
4. Install **WSL**.

---

7. Open the Project in VS Code (Remote)

Opening the project in a WSL remote session ensures that:

- The Linux environment is used
- The virtual environment and planners are accessible
- Commands behave consistently for all students

Steps

1. Navigate to your workspace directory:

```
cd ~/LabsPDDL
```

2. Open the current directory in VS Code:

```
code .
```

8. Install PDDL Extension

The PDDL extension provides syntax highlighting and basic validation, making it easier to write and debug domain and problem files.

Steps

1. Open Extensions (`Ctrl+Shift+X`)
2. Search for **PDDL**
3. Install a PDDL syntax/highlighting extension

---

9. Create a Test PDDL Domain and Problem

Creating a minimal domain and problem allows verifying that:

- The planner is correctly installed
- PDDL files are syntactically valid
- The full toolchain works end-to-end

Steps

1. Create a file named `domain.pddl` with the following content:

```
(define (domain simple)
  (:predicates (at ?x))

  (:action move
    :parameters (?x ?y)
    :precondition (at ?x)
    :effect (and
      (not (at ?x))
      (at ?y)
    )
  )
)
```

2. Create a file named `problem.pddl` with the following content:

```
(define (problem simple-problem)
  (:domain simple)
  (:objects A B)
  (:init (at A))
  (:goal (at B))
)
```

---

10. Run the FF Planner

This final step confirms that:

- The planner executes correctly
- The PDDL files are valid
- A solution plan can be generated

Steps

1. Activate the virtual environment if needed:

```
source ~/LabsPDDL/venv/bin/activate
```

2. Run the planner:

```
planutils run ff -o domain.pddl -f problem.pddl
```

Expected Output

```
step    0: move A B
```

Summary

You now have a fully functional automated planning environment:

- Linux-based execution via WSL
- Containerized planners via Apptainer
- Reproducible Python tooling
- A professional IDE setup
- A validated planning workflow

This setup will be used throughout the laboratory sessions of the course.