Section 1

CS237A Fall 2025
Last updated Sep 25, 2025

Overview

The goals for this section:

1. Learn to navigate in a Unix OS with terminal commands
2. Learn to use Git to create and track software developement
3. Learn to write Python and Shell executables

Working with Unix Terminal

Complete the following tasks using terminal commands only. Don't use the graphical interface and save those commands (in your mind) as you will need them later.

Task 1.1 — Cleanup
Remove directory `~/autonomy_ws` if it exists. This may be leftover from a previous section.

Task 1.2 – Create workspace
Pick a name for the directory that would host your robot autonomy code in the future. For the reset of this document, we use `<repo>` to denote the directory name you chose. Create a nested directory at `~/autonomy_ws/src/<repo>`.

Task 1.3 – Add a README
Create a `README.md` file in `<repo>` you just created, and add a title to it. For example,

```TypeScript
# We Build Autonomous Robots
```

Hint: you may use your favorite code editing software to edit the README file, For example, you can launch VSCode to edit any file with the following command,

```Shell
code <path>/<to>/<file>
```

Other code editing software works as well, e.g. `vim`, `nano`, `emacs`, etc.

Terminal Cheat Sheets

1. Special Directories

```Shell
~  # home directory
.  # current directory
.. # parent directory
/  # root directory
```

2. File System Commands

```
Shell
pwd             # print out the current working directory
ls              # list all files and sub-directories under pwd
ls -l <file>    # show permissions (read, write, executable) of <file>
cd <dir>        # change current directory to <dir>
mkdir <dir>     # create a single directory at <dir>
mkdir -p <dir>  # create a possibly nested directory at <dir>
mv <A> <B>      # rename file <A> to file <B>
cp <A> <B>      # copy file <A> to file <B>
rm <file>       # remove a single <file>
rm -r <dir>     # remove a directory <dir> and everything in it
rm -rf <dir>    # same as above but also work if <dir> does not exist
touch <file>    # create an empty <file>
chmod +x <file> # make <file> executable
chmod -w <file> # make <file> read-only
cat <file>      # print all the content of <file>
du -sh <dir>    # show total size of directory <dir>
```

Using Git

Git is a source control tool that allows people to share code with each other, and more importantly, keep track of a history of code changes for any type of software development. We will guide you through creating a Git repository and share with you some common workflows when working collaboratively with multiple people on the same repository.

Task 2.1 – Create a GitHub account.
Go to [Github](https://github.com) and create an account with your university email if you have not yet done so before.

Task 2.2 – GitHub authentication.
Use the following command and follow the prompts to authenticate the terminal with your GitHub account.

```
Shell
gh auth login
```

Task 2.3 – Setup a GitHub repository.
Create a private new repository with the name <repo> you picked earlier. Follow the guide on your newly created GitHub project page to initialize ~/autonomy ws/src/<repo> as a git repository and push the README to GitHub.

Task 2.4 – Add collaborators.
Go to project settings in GitHub and add your teammates as collaborators to the repository you just created.

Task 2.5 – Add group info by creating a PR.
Use the following steps as a guide:

1. Create a new branch.
2. Add a new file named team.txt and edit it to include all SUNetIDs of your team members (separated by newlines).
3. Add, commit, and push the new file to a new branch.
4. Create a pull request from the GitHub project page.
5. Review the changes and merge it back to the main branch.

Git Cheat Sheet

Shell

```shell
# local commands
git checkout <branch>      # switch to <branch>
git checkout -b <branch>   # create a new local branch from the current branch
git add .                  # add all files in the current directory
git add <file or dir>      # add a single file or directory
git add -u                 # add all tracked files
git commit -m "<msg>"      # commit all added changes to the local branch

# server commands
git fetch              # sync remote branch (does not change local files)
git pull               # sync remote branch and also update local files
git clone <url>        # clone git repo from <url> to the current directory
```

```
git push -u origin <branch>  # push to a newly created local branch and track
remote branch
git push                     # subsequent pushes after a local branch is
tracked
```

Executables

Python

All executables are just files with the `x` permission turned on. Here we focus on two scripting languages, Python and Shell, which you will use the most to build up your robot autonomy stack.

Task 3.1.1 – Create a Python executable.

1. Create a Python file at `<repo>/scripts/section1.py`.

2. Edit the file and use the following code as a start-up template.

```python
#!/usr/bin/env python3

# add import and helper functions here

if __name__ == "__main__":

    # code goes here
```

The first line, a.k.a the shebang, is always required for the OS to find the correct Python interpreter to execute this file. Shell scripts will require a different shebang (see Shell).

3. Change the permission of the file to make it an executable.
   Hint: use `chmod`.

4. Check that the permission of the file is indeed correct by running the following command

```Shell
ls -l <path to section1.py>
```

   and you should get a line starting with

```Shell
-rwxr-xr-x # ...
```

   where the `x` indicates that the execute permission is indeed turned on for the file.

Task 3.1.2 – Execute it! Add something simple to the file (e.g. `print("hello world")`), and then try to execute the file by writing the full path to `section1.py` and hit return.

Note: if you are currently in the `<repo>/scripts` directory in your terminal, you cannot just type `section1.py` and run the code. Instead, you need to type `./section1.py`.

Task 3.1.3 – Learn about numpy matmul operators. Import numpy as `np` in your Python script and add the following snippet to the main block to generate two random matrices.

```python
np.random.seed(42)
A = np.random.normal(size=(4, 4))
B = np.random.normal(size=(4, 2))
```

Write code to compute and print out the matrix product A · B using the Python matmul operator – `@`. The result should match the following

```python
array([[ 0.67459114, -1.96480447],

       [ 2.81615463, -1.19285965],

       [-0.72781678, -0.14561428],

       [-1.07385636,  3.96873247]])
```

Task 3.1.3 – Learn about numpy slicing and broadcasting. Add the following snippet to the main block to generate a batch of size-10 vectors.

```python
np.random.seed(42)
x = np.random.normal(size=(4, 10))
```

Write code to compute for a L2 squared distance matrix among the vectors. Let D denote such matrix, then it can be defined with

$$D_{ij} = \|x_i - x_j\|_2^2 = (x_i - x_j)^T (x_i - x_j)$$

Do NOT use a for loop!

The results should match the following

```python
array([[ 0.        , 26.26987784, 18.50845045, 24.5282298 ],
       [26.26987784,  0.        ,  7.26795838, 22.54199209],
       [18.50845045,  7.26795838,  0.        , 20.44680969],
       [24.5282298 , 22.54199209, 20.44680969,  0.        ]])
```

Hints:

1. `np.sum` and `np.square` could be useful.
2. Some intermediate variables should broadcast to a shape of `(4, 4, 10)`.
3. (Optional) A clever math trick can reduce the broadcast size to `(4, 4)`.

Cheat Sheet

```python
a = np.zeros((4, 6))  # a is a (4, 6) 2d array
b = np.zeros((4, 1))  # b is a (4, 1) 2d array
c = np.zeros((1, 6))  # c is a (1, 6) 2d array
d = np.zeros(6)       # d is a size-6 1d array

# slicing
a[0, 0]       # gets a single element
```

```
a[:, 0]        # gets a size-4 vector
a[0, :]        # gets a size-6 vector
a[:, 0:3]      # gets a (4, 3) 2d array
a[None, :, :] # gets a (1, 4, 3) 3d array
a[:, None, :] # gets a (4, 1, 3) 3d array

# broadcasting
b + c          # gets a (4, 6) array
c + d          # gets a (1, 6) array
```

Shell (If using lab computer)

See Wikipedia for some histories of the shell language. In short, this is the default language to interface with most UNIX terminal environments.

Task 3.2.1 – Create a cleanup script. Create a shell script at `<repo>/scripts/cleanup.sh` and change the permission to make it an executable. You also need to add the following shebang to the top of the script.

Shell
```shell
#!/usr/bin/sh
```

Task 3.2.2 – Write the cleanup script. This script will be used to quickly clean up your workspace at the end of each section. Edit cleanup.sh to achieve the following functionalities:

1. Logout your GitHub account — you can add the following line to your shell script

```shell
gh auth logout
```

2. Remove your workspace directory `~/autonomy_ws`.

Wrap Up

Task 4.1 – Upload changes to GitHub. Commit and push all changes to a new branch and create a PR and name it **Section 1**.

Task 4.2 – Cleanup (if on lab computer). Before doing this, make sure your changes are pushed to GitHub. Otherwise your work will be wiped without a backup!

Run `cleanup.sh` to clean up your workspace for the next section.