A Getting Started

The individual mandatory assignments (also referred to as "labs") in DM587/AI511 aim to introduce applications of Linear Algebra and to train your Python programming skills. There will be weekly programming tasks for which you have to submit a solution.

Submitting Assignments

Labs

Every lab has a corresponding specifications file with some code to get you started and to make your submission compatible with automated test drivers. The template code will be provided via IMADA Git Server at `https://gitlab.sdu.dk`. How to proceed in detail will be described below.

To submit a lab, modify the provided specifications file and use `git` to submit your solution (discussed in the next section). The submissions will be automatically graded. The first assignment, `asg-tryout`, will not be graded and is just used to introduce you to the procedure of how you should submit your solutions and how your solution is graded. It has the specifications file `asg-tryout/tryout.py`. To complete that assignment, provide your implementation in the file `asg-tryout/tryout.py` and submit it via `git`. After grading, you should visit the web page `https://dalila.imada.sdu.dk/` to see the outcome of your tests. Each assignment will have a formal deadline. The final grade of each assignment will be based on your files as they are at the exact time of the deadline of the assignment.

> ACHTUNG!
>
> Do **not** move or rename the lab folders or the enclosed specifications files; if you do, the test drivers will not be able to find your assignment. Do **not** edit the files `.drone.yml` and `grade.txt`, if present. These files are overwritten when you pull with git from the remote server and must stay unchanged.

Setup

> **Achtung!**
>
> We strongly recommend using a Unix-based operating system (Mac or Linux or Windows Subsystem for Linux) for the labs. Unix has a true bash terminal, works well with git and python, and is the preferred platform for computational and data scientists. It is possible to do this curriculum with Windows, but expect some road bumps along the way. We will ensure that all the exercises can be solved in the IMADA Virtual Computer Lab. You can use your own environment, but you should not expect that we are able to answer your environment specific questions.

Code has to be submitted using git.

Setup With Git

Git is a program that manages updates between an online code repository and the copies of the repository, called *clones*, stored locally on computers. Git is installed in the IMADA Virtual Computer Lab. The instructions given below in this document should be enough for the needs in this course. The tutorials linked below will provide much more information than needed in this course. Nevertheless, git is an industry-standard collaboration tool, and being able to use it efficiently is an asset.

If you decide to use your own computer, and git is not already installed on your computer, you can download it at `http://git-scm.com/downloads` (or use the installation procedure of your specific system). If you have never used git, you might want to read a few of the following resources.

- Official git tutorial: `https://git-scm.com/docs/gittutorial`
- Bitbucket git tutorials: `https://www.atlassian.com/git/tutorials`
- GitHub git cheat sheet: `https://education.github.com/github-git-cheat-sheet.pdf`
- GitLab git tutorial: `https://docs.gitlab.com/ce/gitlab-basics/start-using-git.html`
- Codecademy git lesson: `https://www.codecademy.com/learn/learn-git`
- Training video series by GitHub: `https://www.youtube.com/playlist?list=PLg7.../`

There are many websites for hosting online git repositories. SDU has its own server for hosting git repositories `https://gitlab.sdu.dk`. While not needed for submitting your code, you can login to the webpage using your university account name and the same password as you use for reading your mail or logging into blackboard. Choose as authentication source "SDU". Via the webpage you will always be able to see the state of your code that will be used for auto-grading.

1. *Clone your existing repository.*

   Usually, you have to create a repository. However, we already created a repository for each student of DM587. You will not have to create any repositories, but only clone it.

2. *Connect your folder to the new repository.* In a shell application (Terminal on Linux or Mac, or Git Bash (`https://gitforwindows.org/` on Windows), enter the following commands (we will use the student with the username "username" as example, of course you have to change this).

```
# Navigate to the folder where you want to store your files
$ cd /path/to/folder  # cd means 'change directory'.

# Make sure you are in the right place.
$ pwd                 # pwd means 'print working directory'.
/path/to/folder

# Clone the repository we provided

$ git clone https://gitlab.sdu.dk/dm587-e24/username-repo.git

Cloning into 'username-repo'...
Username for 'https://gitlab.sdu.dk': username
Password for 'https://username@gitlab.sdu.dk': ********
remote: Counting objects: 48, done.
remote: Compressing objects: 100% (44/44), done.
remote: Total 48 (delta 16), reused 0 (delta 0)
Unpacking objects: 100% (48/48), done.
$ ls
username-repo

$ cd username-repo
$ ls -rtl
drwx------ 2 username username 4096 Oct 31 19:48 asg-tryout

# Record your credentials (has to be done once only).
$ git config --local user.name "Firstname Surname"
$ git config --local user.email "username@student.sdu.dk"
```

You can also use SSH for authentification. In this case, you have to follow the instructions linked from the GitLab official documentation.

3. *Install Python package dependencies.* Some of the labs require third-party Python packages that you might not have installed on your system. If they are missing you will see an error message similar to

```
$ python test.py
Traceback (most recent call last):
  File "test.py", line 1, in <module>
    import matplotlib
ImportError: No module named matplotlib
```

You can easily install missing packages via

```
$ pip3 install --user matplotlib
Collecting matplotlib
  Downloading https://files.pythonhosted.org/packages/b2/58/5842588↩
      fa67b45ffb451c4c98eda283c0c42b8f2c5e503e4f6d9ff3c3a63/matplotlib↩
      -3.0.1-cp35-cp35m-manylinux1_x86_64.whl (12.9MB)
```

```
[...]
```

Note, that you must have git installed in order to i.) get the data files for each lab and ii.) to submit you solution. Git is installed in the Virtual Computer Lab — if you want to install it within your own environment, `http://git-scm.com/downloads` is a good starting point.

Using Git

Git manages the history of a file system through *commits*, or checkpoints. Use `git status` to see the files that have been changed since the last commit. These changes are then moved to the (local) *staging area* (a list of files for the next commit) with `git add <filename(s)>`. Record the changes in the staging area with the command `git commit -m "<A brief message describing the changes>"`.

All of these commands are done within a "clone" of the repository, which is stored somewhere on a computer. This repository must be manually synchronized with the remote repository server via two other git commands: `git pull`, to pull updates from the web to the computer; and `git push`, to push updates from the computer to the git server.

In a nutshell, for the Labs in DM587 you usually have to modify one file only. This file first has to be added to the staging area, then it has to be commited, and then it has to be pushed to the remote server. In order to get the grading, you have to pull the corresponding file from the server after we tested your solution and created the grading file.

Command	Explanation
`git status`	Display the staging area and untracked changes.
`git pull`	Pull changes from the online repository.
`git push`	Push changes to the online repository.
`git add <filename(s)>`	Add a file or files to the staging area.
`git commit -m "<message>"`	Save the changes in the staging area with a given message.

Table A.1: Most common git commands needed for DM587.

Command	Explanation
`git add -u`	Add all modified, tracked files to the staging area.
`git checkout -- <filename>`	Revert changes to an unstaged file since the last commit.
`git reset HEAD -- <filename>`	Remove a file from the staging area.
`git diff <filename>`	See the changes to an unstaged file since the last commit.
`git diff --cached <filename>`	See the changes to a staged file since the last commit.
`git config --local <option>`	Record your credentials (`user.name`, `user.email`, etc.).

Table A.2: Some more git commands.

Note

When pulling updates with `git pull origin master`, your terminal may sometimes display the following message.

```
Merge branch 'master' of https://gitlab.sdu.dk/<name>/<repo> into master

# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
#
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.
~
~
```

This means that someone else (the grading system) has pushed a commit (e.g., the file containing your grades) that you do not yet have, while you have also made one or more commits locally that they (the grading system) do not have. This screen, displayed in *vim* (`https://en.wikipedia.org/wiki/Vim_(text_editor)`), is asking you to enter a message (or use the default message) to create a *merge commit* that will reconcile both changes. To close this screen and create the merge commit, type `:wq` and press `enter`.

Example Work Session

Cloning and giving details on your name and email has only to be done once. The below work session assumes this has been done already.

Short version:

```
$ cd ~/Desktop/Student-Materials/
$ git pull                                          # Pull updates.

# Make changes to a file (in this example tryout.py)

# Record the changes in git.
$ git add tryout.py                               # Track changes.
$ git commit -m "Made some changes."                # Commit changes.
$ git push                                          # Push updates.
```

Long version:

```
# Navigate to the clone of the repository.
$ cd ~/Desktop/Student-Materials/

# Pull any updates from the online repository (such as preliminary feedback and↩
    grading), if they exist.

$ git pull

remote: Counting objects: 4, done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 2), reused 0 (delta 0)
Unpacking objects: 100% (4/4), done.
```

```
From https://gitlab.sdu.dk/dm587-e24/username-repo
   6dde06d..e24cee5  master     -> origin/master
Updating 6dde06d..e24cee5
Fast-forward
 asg-tryout/grade.txt | 33 +++++----------------------------
 1 file changed, 5 insertions(+), 28 deletions(-)

# It seems someone graded your solution, and you would find the result in the ↩
    file asg-tryout/grade.txt

# Work on the labs. For example, modify asg-tryout/tryout.py

$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   asg-tryout/tryout.py

no changes added to commit (use "git add" and/or "git commit -a")

$ git add asg-tryout/tryout.py
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   asg-tryout/tryout.py

# Commit the changes to the repository with an informative message.
$ git commit -m "Made some changes"
[master 72a5ab3] Made some changes
 1 file changed, 1 insertion(+)
[master fed9b34] Made some changes
 1 file changed, 10 insertion(+) 1 deletion(-)

# Push the changes to the online repository.
$ git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 373 bytes | 373.00 KiB/s, done.
Total 4 (delta 2), reused 0 (delta 0)
```

```
To https://gitlab.sdu.dk/dm587-e24/username-repo.git
   e24cee5..72a5ab3  master -> master


# The changes have been saved and the online repository updated.
$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```