Lab 3 - Pre-Lab for Introduction to MOOS

2.680 Unmanned Marine Vehicle Autonomy, Sensing and Communications

February 12th, 2024

Michael Benjamin, mikerb@mit.edu
Tyler Paine, tpaine@mit.edu
Oscar Visquez origin
Department of Mechanical Engineering
MIT, Cambridge MA 02139

1 Overview and Objectives

This pre-lab will help new users get ready to use version control, before introducing MOOS in the upcoming lab proper. The goals of this lab are to (a) create a user-owned copy of the *moos-ivp-extend* repository and (b) introduce *git* as a tool to track the user's software development progress (also known as version control). If you would like, we can also configure Continuous Integration and Continuous Delivery (CI/CD) pipeline, but this is optional at this time.

1.1 Preliminaries

This lab assumes you have a working MOOS-IvP tree checked out and built on your computer. To verify this make sure that the following executables are built and findable in your shell path:

```
$ which MOOSDB
/Users/you/moos-ivp/bin/MOOSDB
$ which pHelmIvP
/Users/you/moos-ivp/bin/pHelmIvP
```

If unsuccessful with the above, return to the steps in Lab 1:

https://oceanai.mit.edu/ivpman/labs/machine_setup

You will also need to have `git` installed. To verify this make sure that the following executable is findable in your shell path:

```
$ which git
/usr/bin/git
```

1.2 Cloning the moos-ivp-extend repository

We plan on using `GitLab.com` for the class. After signing up for a free account, you can import a copy of the course original repository into your account by following these steps:

- Login to `GitLab.com`
- Navigate to Menu - Projects - Create New Project
- Click on Import Project
- Select Repo by URL, with the following information:
  - Git repository URL: `https://gitlab.com/moos-ivp/moos-ivp-extend.git`
  - Project name: "MOOS IvP [your kerberos]" and the project url should read something like `git@gitlab.com:<your GitLab ID>/moos-ivp-<your kerberos>.git`
  - Visibility: choose private
  - Create project

Note that, if GitLab says it can't find a repo in the given URL, you must make sure you have the .git suffix.

The steps above will create an independent copy of the source repository, preserving the history of the original project up to that point. Unlike a fork (another method for creating a copy of a project's repository), it will detach from the original repository going forward. Forking a repository would enable you to make changes and then submit upstream merge requests (pull requests, PRs, in GitHub), for the consideration of the original project team.

1.3 Granting the teaching staff access to your repository

After you have created your own copy of the `moos-ivp-extend` repository, you will have to grant the teaching staff access to your codebase. Later in the term, we will have lab assignments submitted by committing your changes to the repo, followed by a notification to us via email, with the appropriate commit ID.

To add a teaching staff user to your project, follow these steps:

1. Go to your project and select Project information > Members.
2. On the Invite member tab, under GitLab member or Email address, type the username or email address.
3. Select a role with source code access (Developer or Maintainer).
4. Select Invite.

Once you've completed this step, the teaching staff should receive an email from GitLab with a notification about the invite. In the interest of ensuring all was done correctly, however, we ask that students also email the teaching staff with the URL to their repository. This will let us catch issues with the configuration early on, to help you get it all sorted out.

1.4 Configure SSH access on your account

The previous assignment above had you create a copy of the original `moos-ivp-extend` repository into your own account by using the web interface. However, to start developing your own code, you'll need to clone the repository to your workstation. There are two main ways of doing this: using the HTTPS approach or the SSH approach. The preferred method is the latter, for which you will first need to establish a set of public-private keys for your workstation.

```
$ ssh-keygen -t ed25519 -C "your_email@example.com"
```

This command will generate a key pair for you. When prompted for the file where you want to save the key, press enter to accept the default. You can also secure the use of your key by setting a password.

```
> Generating public/private algorithm key pair.
> Enter a file in which to save the key (/Users/you/.ssh/id_algorithm): [Press enter]
> Enter passphrase (empty for no passphrase): [Type a passphrase]
> Enter same passphrase again: [Type passphrase again]
```

On MacOS, you will need to modify your $\tilde{/}$.ssh/config file to automatically load keys into the ssh-agent and store passphrases in your keychain.

```
$ emacs ~/.ssh/config
```

Note that the file may be empty, and might not exist yet. In this case, emacs will create the file for you. In the $\tilde{/}$.ssh/config file, add the following:

```
Host *
  AddKeysToAgent yes
  UseKeychain yes
  IdentityFile ~/.ssh/id_ed25519
```

Once you've created the key and configured your preferences to use the keychain, you'll need to add the key to your ssh-agent. First, start the agent using the following command:

```
$ eval "$(ssh-agent -s)"
> Agent pid 59566
```

The key can be added to your agent using the following command:

```
$ ssh-add -K ~/.ssh/id_ed25519  # on macOS (specifies the keychain option)
$ ssh-add ~/.ssh/id_ed25519     # on Linux
```

The next step is to add the public key to your GitHub or GitLab account, so that the server recognizes your workstation. First, copy the public key:

```
$ cat ~/.ssh/id_ed25519.pub
```

Note: on macOS, you can copy to the system clipboard from terminal using the `pbcopy` utility:

```
$ cat ~/.ssh/id_ed25519.pub | pbcopy
```

Please make sure you include the .pub suffix! Your private key will have the same base name but no suffix, and should remain secret for security reasons. Next, follow these instructions to add the key to your GitLab account:

1. Sign in to GitLab.com
2. Click on your profile photo (upper right corner), then click Preferences
3. On the sidebar, click on "SSH keys"

4. In the Key field, paste the contents of your public key file
5. In the Title field, add a brief descriptionof which computer this is for (such as "2.680 laptop")
6. Click "Add key"

1.5 Pulling code from the server

Once you've added your SSH key to your account, you can clone your repository using one of the following commands:

```
$ cd ~
$ git clone git@gitlab.com:<user>/moos-ivp-<kerberos>
```

This will pull a copy of your repository to your workstation. As this is where you will be developing your new software, this is called a working copy.

Git is designed to facilitate development across multi-member teams. Even for single-member development, you may work from multiple computers (such as a robot's computer and your own laptop). As such, the second command you'll need is one to pull updates that have been pushed to the server either by other team members, or by yourself but from a different computer than you are using at the moment – this includes any changes you might have saved while editing files directly on the website. You can do this with:

```
$ git pull
```

1.6 Configuring your Git environment

When committing changes to a repository, Git creates an attribution record so the author can be identified. You can tell Git what name and email address you want associated with your commits by setting the global defaults:

```
$ git config --global user.email "your_email@example.com"
$ git config --global user.name  "Your Name"
```

1.7 Saving your changes

When you've changed files in your working copy and you want to save your progress, you will need to execute the following sequence of commands:

```
$ git add <file(s) or path(s)>
$ git commit -m "a comment about what changed"
$ git push
```

The first command, `git add`, will create a temporary, local checkpoint of your changes. If you make additional modifications to the file before calling `git commit`, you can revert to the checkpoint created by `git add`. However, if you then call `git add` once again, the checkpoint will be replaced to include the additional changes.

The second command, `git commit`, essentially confirms that you want this checkpoint to become a permanent point in the version control history. The comment is intended as a brief reminder of what changes were made, in case you need to look back through the version history later on.

The checkpoint created by `git commit` will be only available on your current workstation until you share it with the server, which you can do with the third command in the series: `git push`.

To get started with Git, update the README.md file in your local copy (located in the top directory) to indicate that this is your personal repository. An update suggestion is to add a block at the top of the file, listing your name and kerberos, along with an indication that you are taking the class during Spring 2022.

Once you've modified the file locally using your preferred text editor, commit and push your changes using the following sequence:

```
$ git status            # (to see what files have changed in the repo)
$ git add README.md
$ git commit -m "updating README.md to reflect ownership by <kerberos> "
$ git push
```

You should be able to verify the changes to the README.md file by going to your repository's URL using a web browser.

2 Bonus: Setting up CI/CD

2.1 Showing the CI pipeline badge

First we need to create a registration token which will allow the code in your repository to build on a remote server.

- On the sidebar menu, navigate to Settings > CI/CD
- Click Expand on the "Runners" section
- First switch the toggle to "off" to disable instance runners for this project. The toggle is in the right column under "Instance runners".
- Next, in the left column under "Project runners", look to the right of the blue button to create "New project runner" and click the small button with three vertical dots. Copy the registration token and email it and your github url to Tyler at tpaine@mit.edu so he can add it to the server.
- When your token has been registered, you should see a new runner appear at the bottom of the left column. (You may need to refresh the page.) When the runner appears you can finsh the setup as described below.

By default, the project will launch a simple pipeline to test building your code each new commit pushed to the server. Additional rules can be added to limit when and how builds happen, or to define additional steps the server should take. This should start running when you update your README.md file. On GitLab.com, you can also launch the pipeline manually:

- On the sidebar menu, navigate to CI/CD `->` Pipelines
- Click on Run Pipeline
- Select the branch you want to work from; the default is main
- Click on Run Pipeline

When the pipeline fails, such as when a new source file has not been committed and is thus not available on the server during the test build, the system will produce an error and email you. GitLab will also display a marker next to the latest commit hash, shown at the top of the content list, whenever a pipeline is executed. An optional pipeline badge can also be displayed at the top of the repository information.

On GitLab, the optional status badge can be added with the following steps:

- On the sidebar menu, navigate to your project site (i.e. https://gitlab.com/<username>/moos-ivp-<username>)
- Go to Settings > General

- Go to Settings `->` General
- Expand the Badges section
- Enter the following information:
  - Name: Pipeline Status
  - Link: `https://gitlab.com/project_path/-/commits/default_branch`
  - Badge image URL: `https://gitlab.com/project_path/badges/default_branch/pipeline.svg`

You can see the badge in the original GitLab repo (`https://gitlab.com/moos-ivp/moos-ivp-extend`), right under the number of commits to the project.