School of Computing Science

CSC2101 – Professional Software Development & Team Project 1 (PSD & TP1)

Labs 1:
Using Github and Setup the Github Account

Asst. Prof Cao Qi
Qi.Cao@Glasgow.ac.uk

The modules of CSC2101 - PSD & TP1 and CSC2102 - PSD & TP2 will be using Github platform for your learning and TP projects development. This labs manual will guide you to setup the Github accounts, create a repository, use Git commands to remote pull/push between the local contents and the Github repository.

If you cannot complete this labs manual, you can continue working after class and in Week 2 labs session.

(1). Access and setup the Github accounts

The GitHub organisation for the PSD & TP1 and PSD & TP2 is at https://github.com/orgs/UofG-CS-2023/teams. Github is a code repository to be jointly managed by your TP group members. You can share your Github codes to your team members and your TP customers.

You should have received an email invitation from GitHub via your Glasgow email address. Please follow the instructions in the email to get access your TP group account on GitHub, for your group's repository by clicking on the name of your group, for example TP-2023-group-XX (from TP-2023-group-01 to TP-2023-group-24).

On the project board of your Github repository, you can manage and update the descriptions of your project board. You can perform the following actions on your project board:

- Sort tasks

- Plan your project
- Track progress
- Automate your workflow
- Share status

(2). Install Git on your laptops

Download the github software and install on your laptops.

- For Windows OS, use this link: https://git-scm.com/download/win
- For Mac OS, use this link: https://git-scm.com/download/mac
- On Linux Ubuntu OS, type `sudo apt-get install git-all` into a shell.

(3). Git Operations

Git is a distributed change management tool, meaning everyone has their own copy of the history of a project in a local repository. Changes can be migrated between any repositories that share some common history. In practice, Git is used with centralised hubs, github, gitlab, bitbucket, etc. Git has many user interfaces (GUI, command line).

Web link of user manual: About Git - GitHub Docs

Basic Git commands

To use Git, developers use specific commands to copy, create, change, and combine code. These commands can be executed directly from the command line or by using an application like GitHub Desktop. Here are some common commands for using Git:

- **git init** initializes a brand new Git repository and begins tracking an existing directory. It adds a hidden subfolder within the existing directory that houses the internal data structure required for version control.
- **git clone** creates a local copy of a project that already exists remotely. The clone includes all the project's files, history, and branches.
- **git add** stages a change. Git tracks changes to a developer's codebase, but it's necessary to stage and take a snapshot of the changes to include them in the project's history. This command performs staging, the first part of that two-step process. Any changes that are staged will become a part of the next snapshot and a part of the project's history. Staging and committing separately gives developers complete control over the history of their project without changing how they code and work.
- **git commit** saves the snapshot to the project history and completes the change-tracking process. In short, a commit functions like taking a photo. Anything that's been staged with git add will become a part of the snapshot with git commit.

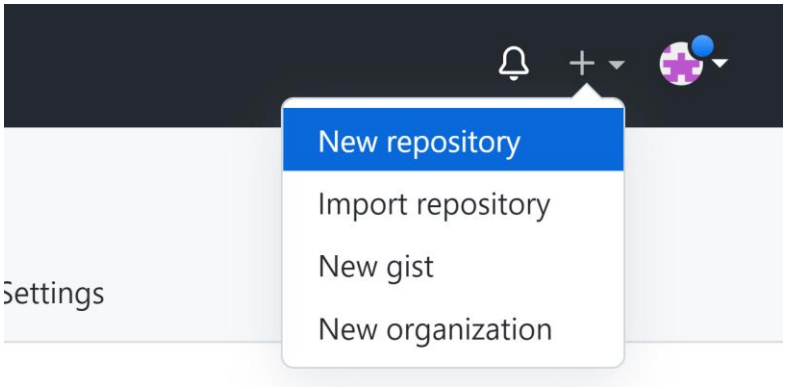
- **git status** shows the status of changes as untracked, modified, or staged.
- **git branch** shows the branches being worked on locally.
- **git merge** merges lines of development together. This command is typically used to combine changes made on two distinct branches. For example, a developer would merge when they want to combine changes from a feature branch into the main branch for deployment.
- **git pull** updates the local line of development with updates from its remote counterpart. Developers use this command if a teammate has made commits to a branch on a remote, and they would like to reflect those changes in their local environment.
- **git push** updates the remote repository with any commits made locally to a branch.

For more information, see the full reference guide to Git commands.

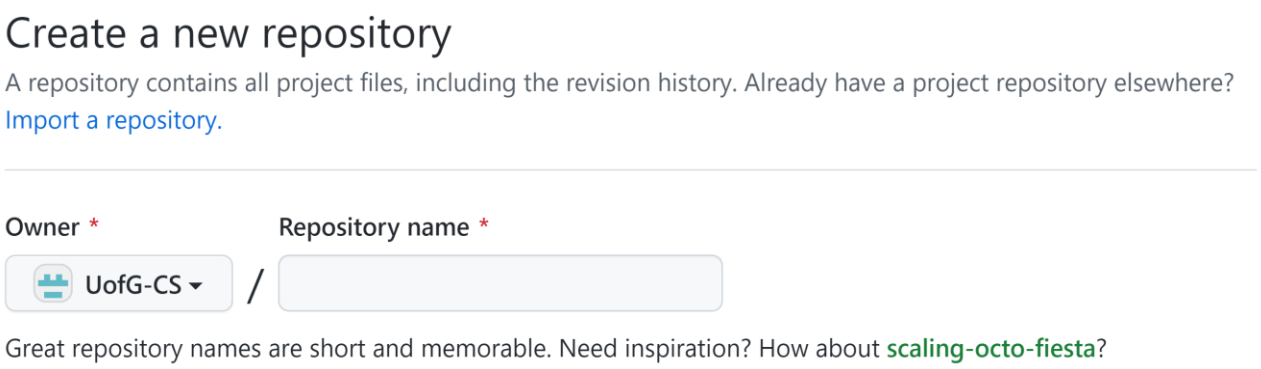
(4). Create a repository for your Github Group on Github.com

Web link of user manual: Repositories - GitHub Docs

On your Github account page, at the top-right corner, click the menu of "New repository".



On the page, you can choose to import an existing repository, or create a new repository.



…or create a new repository on the command line

```
echo "# project-TP-groupXX" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M master
git remote add origin https://github.com/Your-Account/project-TP-groupXX.git
```

…or push an existing repository from the command line

```
git remote add origin https://github.com/Your-Account/project-TP-groupXX.git
git branch -M master
git push -u origin master
```

Then add the repository into your TP group Github account.

(5). Configuring your Git username and email using Git Bash

Git options can either be set for a repository, or globally. For example:

```
$ git config --global user.name “your-github-user-name”
$ git config --global user.email “your-github-email-address” (your school email)
```

Setting your Git username for *every* repository on your computer

1. Open Git Bash.
2. Set a Git username:

   $ git config --global user.name "*Mona Lisa*"
3. Confirm that you have set the Git username correctly:
4. $ git config --global user.name

   > Mona Lisa

Setting your Git username for a single repository

1. Open Git Bash.
2. Change the current working directory to the local repository where you want to configure the name that is associated with your Git commits.
3. Set a Git username:

   $ git config user.name "*Mona Lisa*"
4. Confirm that you have set the Git username correctly:
5. $ git config user.name

> Mona Lisa

Setting your commit email address in Git

You can use the `git config` command to change the email address you associate with your Git commits. The new email address you set will be visible in any future commits you push to GitHub.com from the command line. Any commits you made prior to changing your commit email address are still associated with your previous email address.

Setting your email address for every repository on your computer

1. Open Git Bash.
2. Set an email address in Git.

   $ git config --global user.email "*your-github-email-address*"
3. Confirm that you have set the email address correctly in Git:
4. $ git config --global user.email

   your-github-email-address
5. Add the email address to your account on GitHub, so that your commits are attributed to you and appear in your contributions graph. For more information, see "Adding an email address to your GitHub account."

Setting your email address for a single repository

GitHub uses the email address set in your local Git configuration to associate commits pushed from the command line with your account on GitHub.com.

You can change the email address associated with commits you make in a single repository. This will override your global Git configuration settings in this one repository, but will not affect any other repositories.

1. Open Git Bash.
2. Change the current working directory to the local repository where you want to configure the email address associated with your Git commits.
3. Set an email address in Git.

   $ git config user.email "*your-github-email-address*"
4. Confirm that you have set the email address correctly in Git:
5. $ git config user.email

   your-github-email-address

(6). Adding a remote repository

To add a new remote, use the git remote add command on the terminal, in the directory your repository is stored at.
The git remote add command takes two arguments:

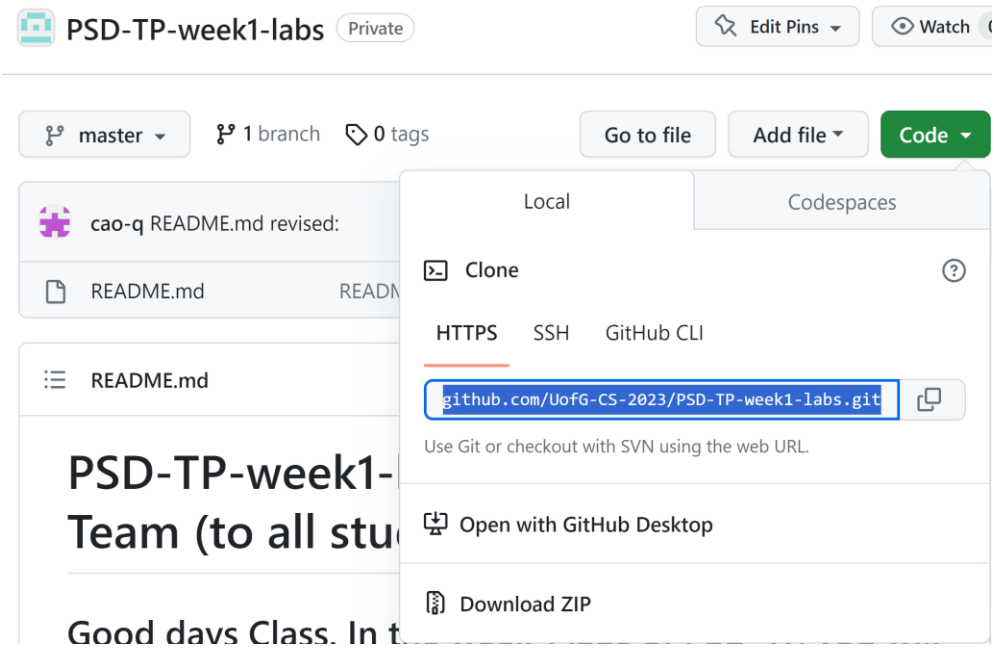
- A remote name, for example, origin, mycode, yourcode, etc.

- A remote URL, for example, https://github.com/user/repo.git

For example:

$ git remote add origin https://github.com/*Your-Account*/Your-*repository*.git
# Set a new remote

Or add the repository of Week 1 labs into your local repository:
$ git remote add origin https://github.com/UofG-CS-2023/PSD-TP-week1-labs.git



$ git remote -v
# Verify new remote
> origin https://github.com/*Your-Account*/Your-*repository*.git (fetch)
> origin https://github.com/*Your-Account*/Your-*repository*.git (push)
For more information on which URL to use, see "About remote repositories."

Troubleshooting: Remote origin already exists

This error means you've tried to add a remote with a name that already exists in your local repository.

$ git remote add origin https://github.com/octocat/Spoon-Knife.git
> fatal: remote origin already exists.

To fix this, you can:

- Use a different name for the new remote.

- Rename the existing remote repository before you add the new remote. For more information, see "Renaming a remote repository" below.
- Delete the existing remote repository before you add the new remote. For more information, see "Removing a remote repository" below.

Inspecting a Remote

If you want to see more information about a particular remote, you can use the Git command:

```
git remote show <remote>
```

If you run this command with a particular shortname, such as origin, you get like this:

$ git remote show origin
** remote origin*

(7). Pull and Push with a remote repository

PULL Request

If you make a change in a repository, GIT PULL can allow others to view the changes. It is used to acknowledge the change that you've made to the repository that you're working on. Or also called a target repository.

The simple command to PULL from a branch is:

```
git pull 'remote_name' 'branch_name'
```

The git pull command is a combination of git fetch which fetches the recent commits in the local repository and git merge, which will merge the branch from a remote to a local branch also 'remote_name' is the repository name and 'branch_name' is the name of the specific branch.

PULL Request through Command Line at two different ways

You can submit a PULL Request you'll see the process for making a PULL Request through a particular example given below.

(a). Fork the Repository

The "Fork" is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. Follow the steps according to the web link: Fork a repo - GitHub Docs

(b). Make a new branch

You can create a new branch by using the command:

```
git checkout -b 'branch_name'
```

In the above code, '-b' flag is used to create a new branch, and 'branch_name' is used to give the branch a specific name, and with checkout, the branch is switched to the newly created branch.

Pushing to Your Remotes

When you have your project at a point that you want to share, you have to push it upstream. The command for this is simple: git push <remote> <branch>. If you want to push your master branch to your origin server (again, cloning generally sets up both of those names for you automatically), then you can run this to push any commits or local contents to GitHub:

$ git push origin master

Or $ git push -u origin master

('-u' flag is upstream, which is equivalent to '-set-upstream.' and the master is the branch, name.upstream is the repository that we have cloned the project.)

This command works only if you cloned from a server to which you have write access and if nobody has pushed in the meantime. If you and someone else clone at the same time and they push upstream and then you push upstream, your push will rightly be rejected. You'll have to fetch their work first and incorporate it into yours before you'll be allowed to push. See Git Branching for more detailed information on how to push to remote servers.

(Web link for more user manual: GIT Push and Pull Tutorial | DataCamp)

(8). Syntax of *markdown* documents

Nearly all Markdown applications support the basic syntax outlined in the original Markdown design document. Please see the detailed user manual here: https://www.markdownguide.org/basic-syntax/

Headings

To create a heading, add number signs (#) in front of a word or phrase. The number of number signs you use should correspond to the heading level. For example, to create a heading level three (<h3>), use three number signs (e.g., ### My Header).

Markdown	HTML	Rendered Output
# Heading level 1	Heading level 1	**Heading level 1**
## Heading level 2	Heading level 2	Heading level 2
### Heading level 3	Heading level 3	**Heading level 3**
#### Heading level 4	Heading level 4	***Heading level 4***
##### Heading level 5	Heading level 5	**Heading level 5**
###### Heading level 6	Heading level 6	**Heading level 6**

Paragraphs

To create paragraphs, use a blank line to separate one or more lines of text.

Markdown	HTML	Rendered Output
I really like using Markdown. I think I'll use it to format all of my documents from now on.	I really like using Markdown. I think I'll use it to format all of my documents from now on.	I really like using Markdown. I think I'll use it to format all of my documents from now on.

Ordered Lists

To create an ordered list, add line items with numbers followed by periods. The numbers don’t have to be in numerical order, but the list should start with the number one.

Markdown	HTML	Rendered Output
1. First item 2. Second item 3. Third item 4. Fourth item	 - First item - Second item - Third item - Fourth item 	1. First item 2. Second item 3. Third item 4. Fourth item

Unordered Lists

To create an unordered list, add dashes (-), asterisks (*), or plus signs (+) in front of line items. Indent one or more items to create a nested list.

Markdown	HTML	Rendered Output
- First item - Second item - Third item - Fourth item	 - First item - Second item - Third item - Fourth item 	• First item • Second item • Third item • Fourth item
* First item * Second item * Third item * Fourth item	 - First item - Second item - Third item - Fourth item 	• First item • Second item • Third item • Fourth item

Bold

To bold text, add two asterisks or underscores before and after a word or phrase. To bold the middle of a word for emphasis, add two asterisks without spaces around the letters.

Markdown	HTML	Rendered Output
I just love **bold text**.	I just love bold text.	I just love **bold text**.
I just love __bold text__.	I just love bold text.	I just love **bold text**.
Love**is**bold	Loveisbold	Love**is**bold

Italic
To italicize text, add one asterisk or underscore before and after a word or phrase. To italicize the middle of a word for emphasis, add one asterisk without spaces around the letters.

Markdown	HTML	Rendered Output
Italicized text is the *cat's meow*.	Italicized text is the cat's meow.	Italicized text is the *cat's meow*.
Italicized text is the _cat's meow_.	Italicized text is the cat's meow.	Italicized text is the *cat's meow*.
A*cat*meow	Acatmeow	A*cat*meow

Blockquotes

To create a blockquote, add a > in front of a paragraph.

```
> Dorothy followed her through many of the beautiful rooms in her castle.
```

The rendered output looks like this:

> Dorothy followed her through many of the beautiful rooms in her castle.

Blockquotes with Multiple Paragraphs

Blockquotes can contain multiple paragraphs. Add a > on the blank lines between the paragraphs.

```
> Dorothy followed her through many of the beautiful rooms in her castle.
>
> The Witch bade her clean the pots and kettles and sweep the floor and keep the fire fed with wood.
```

The rendered output looks like this:

> Dorothy followed her through many of the beautiful rooms in her castle.
>
> The Witch bade her clean the pots and kettles and sweep the floor and keep the fire fed with wood.

Nested Blockquotes

Blockquotes can be nested. Add a >> in front of the paragraph you want to nest.

```
> Dorothy followed her through many of the beautiful rooms in her castle.

>

>> The Witch bade her clean the pots and kettles and sweep the floor and keep the fire fed with wood.
```

The rendered output looks like this:

```
Dorothy followed her through many of the beautiful rooms in her castle.
The Witch bade her clean the pots and kettles and sweep the floor and keep the fire fed with wood.
```

(9). Creating README.md

Create a README.md in your GitHub repository with the details of your own information, and your expectation on the learning of PSD and TP.
Use these Git commands to commit the README.md file to your Github repository.

```
$ git init
$ git status
$ git add README.md
$ git commit -m "Added README.md"
```

(10). Edit the file of .gitignore

A file that lists any files that should be ignored by Git (i.e., files not subject to version control).

- Create this .gitignore file using **touch** or **nano** command:

  nano .gitignore

- This file can be placed anywhere in the project directory structure. Usually it is placed in the root directory.
- Then edit the .gitignore file, by adding the files that you don't want to be the version controlled, e.g.,

```
# Python Files
*.pyc
```

- Then save this file.
- Use the Git command to configure the gitignore file.

  git config --global core.excludesFile .gitignore

(11). Merge Conflicts

Sometimes somebody will push changes to a repository that you are working on locally. If you then pull these changes, they may create a merge conflict that needs to be resolved.

To recreate this scenario, take the following steps with one of your main repository. Try the following with two team members. In this demonstration, Ed and Jac are pseudonames.

a) Ed clones **https://github.com/UofG-CS-2023/PSD-TP-week1-labs.git** on GitHub.
b) Jac clones **https://github.com/UofG-CS-2023/PSD-TP-week1-labs.git** on GitHub.
c) Ed adds the following the beginning of the file **README.md.**

```
# TP common group repository
```

d) Ed commits and pushes the changes to GitHub.
e) Jac pulls the changes from GitHub.
f) Jac edits the beginning of the file **README.md.**

```
# this file is revised by Jac at 9:10 am on 1 Sep 2023
```

g) Jac commits and pushes the changes to GitHub.
h) Ed now makes further changes to the **README.md** file.

```
# this file is revised by Ed at 9:20 am on 1 Sep 2023
```

i) Ed stages and commits the change.
j) Ed tries to push the change, but gets an error, telling him he needs to pull.

```
git pull origin master
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
```

Ed does this and a file like this:

```
<<<<<<< HEAD
# this file is revised by Ed at 9:20 am on 1 Sep 2023
=======
# this file is revised by Jac at 9:10 am on 1 Sep 2023
>>>>>>> a6969ec0d266034186a0a277f7b1f561825b5cd6
…
```

k) Jac will now need to edit this (either manually or with a diff tool) and then stage, commit and push the resulting change.

(12). Creating your first milestone

Please create the first milestone on the project as “Initial Setup” under your Github repository:

- Go to **Issues.**
- Click on **Milestones.**
- Click on **New milestone.**
- Enter information and click “**Create milestone**”.

(13). Creating the first issue

- Go to **Issues.**
- Click on **New Issue.**
- Enter the issue as “Basic Setup” and click “**Submit new issue**”.
- The issue then should be **closed** by another team member.