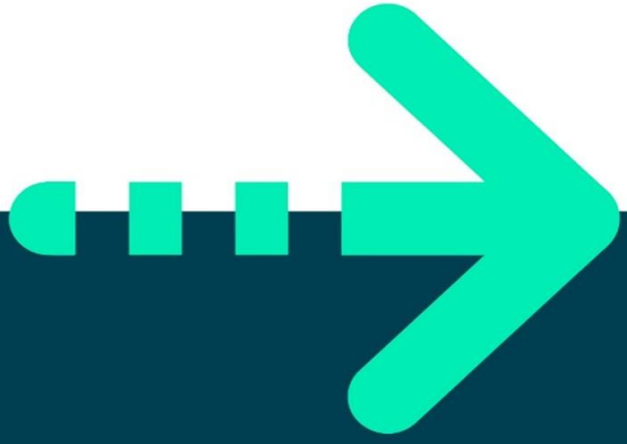
Introduction to Git

Exercise Guide



Exercise 01 - Setting up Git and the lab environment

Objective

In this exercise we will make sure Git is ready to use and create a folder to use in the labs.

Step 0 - Install Git

1. If running Linux, install via your distro's package manager.
2. If running Windows, navigate to https://git-scm.com/download/win and download the standalone installer. Once it has downloaded, run the installer and accept all defaults.

Step 1 - Configure Git

1. Configure your Git installation with your name and email address. Open Git Bash or a Linux terminal and type:

```
git config --global user.email "you@email.com"
git config --global user.name "Your Name"
```

Step 2 - Create a folder to use as a Git repository

1. Create a folder on your computer to store our local Git repo, e.g.:

```
cd /
md labs
cd labs
md gitrepo
cd gitrepo
```

Exercise 02 - Basic Git usage

Overview

In this exercise we will be creating a Git repo and experimenting with basic Git usage. This includes committing changes, viewing status, looking at the log, and reverting changes.

Step 1 - Initialise the Git repo

1. At a command prompt, change to the folder you created in the first exercise if you aren't still located there, e.g.:

   ```
   cd /labs/gitrepo
   ```

2. Initialise the folder as a Git repo

   ```
   git init
   ```

3. Check the folder is a Git repo

   ```
   git status
   ```

Step 2 - Add files and commit changes

1. Create a file in the folder using your text editor of choice e.g., notepad if on windows.

   ```
   notepad README.md
   ```

2. Add some text to this file, a simple hello world will do, then save the file.

3. Get the status of the repo and observe the new files are not currently being tracked.

   ```
   git status
   ```

4. Try to commit the changes and note that nothing is actually committed.

   ```
   git commit -m "First commit"
   git status
   ```

5. Add the file to the staging area and commit.

   ```
   git add .
   git commit -m "First Commit"
   git status
   ```

   or use the following to open the default editor if you wish to provide a longer commit message:

```
git add .
git commit
git status
```

6. View the log for our repo

   ```
   git log
   ```

   or for a shorter response try:

   ```
   git log --oneline
   ```

Step 3 - Add additional files and track updates

Use instructions from the previous step for the exact syntax.

1. Create another file in the folder called newfile.txt
2. Check the status of the repo.
3. Add and commit the changes. Check the status and the log output.
4. Make a change to one or more of the files and stage, then commit your changes. Again, check the status and the log output.

Exercise 03 - Change Tracking

Overview

In this exercise we will see how Git tracks changes.

Step 1

1. Create a new Git repo

   ```
   cd /labs
   md gitdiff
   cd gitdiff
   git init
   ```

2. Create a text file in the repo called newfile.txt, and add the following line to it:

   ```
   # Version 1
   ```

3. Add and commit the file.

4. Edit the file and add an additional line to the contents.

   ```
   # Second line
   ```

5. Execute a status command and note the output.

6. Issue a diff command.

   ```
   git diff
   ```

7. Commit the change.

8. Make another two or three changes and commit each of them. Use the diff command each time to view the changes.

Step 2 - Unstaging and Resetting

1. Make a change to one of the existing files and look at the Git status.

2. Add the file to the staging area and view the status.

   ```
   git add .
   git status
   ```

3. Remove the file from the staging area and view the status.

   ```
   git reset <filename>
   git status
   ```

4. Undo pending changes by checking out the previous version of the file you just edited.

```
git checkout <filename>
git status
```

Step 3 - Reverting to previous versions

1. Display the log of the repo and note the various commit ids.

   ```
   git log --oneline
   ```

2. Revert to a previous version of the repo.

   ```
   git checkout <pick a commit-id from previous output>
   ```

3. Inspect the folder/file contents to see they have been reverted to how the folder was when the commit was made.

4. Return to the current version of the repo.

   ```
   git switch -
   ```

Exercise 04 - Tagging, Branching, and Merging

Overview

This exercise is designed to show how Git branches work, and how we can use them to streamline our workflow.

Step 1 - Create a repo and add some files

1. Go to your labs folder and create a new Git repo e.g.:

```
cd /labs
md gitbranch
cd gitbranch
git init
```

2. Add and commit a README.md file with some text content.
3. Make some changes, add and commit, repeat two or three times. Check the Git log to see how your changes are tracked.

Step 2 - Tag a commit

1. Use the Git log command to view the repo history. Note the commit id at the start of each commit.
2. Use the tag command to assign a label to the current version and then look at the log again. Note how the tag appears.

```
git tag v1
git log --oneline
```

3. Use the Git tag command to see the list of tags.

```
git tag
```

4. Tag an earlier commit.

```
git tag <commit id>

git tag
```

Step 3 - Branching

1. Display the current log history usinf the oneline option.
2. Display the list of branches (there should only be one, probably **master** or **main** - remember which for later).

```
git branch
```

3. Create and checkout a new branch.

```
git checkout -b newbranch
```

4. Edit one of the existing files (or add a new file), stage and commit the changes. View the Git log and status.

5. Return to the original branch and check the file contents/Git log.

```
git checkout <original branch name>
git log --oneline
```

6. Return to the new branch.

```
git checkout newbranch
```

Step 4 - Merging

1. Return to the original branch and merge changes.

```
git checkout <original branch name>
git merge newbranch
```

2. Inspect the file contents to see the changes have been merged.

Step 5 - Resolve a merge conflict

1. Create and checkout a new branch, add a comment at the start of one of the files. Commit this change.
2. Checkout the main branch.
3. Create and checkout a new branch, add a comment to the start of the same file as in Step 1 (making sure the comment is different). Commit the change.
4. Checkout the main branch.
5. Merge the branch from Step 1.
6. Merge the branch from Step 3. You should see a message pointing out a conflict.
7. Open the conflicting file in a text editor and resolve the conflict.

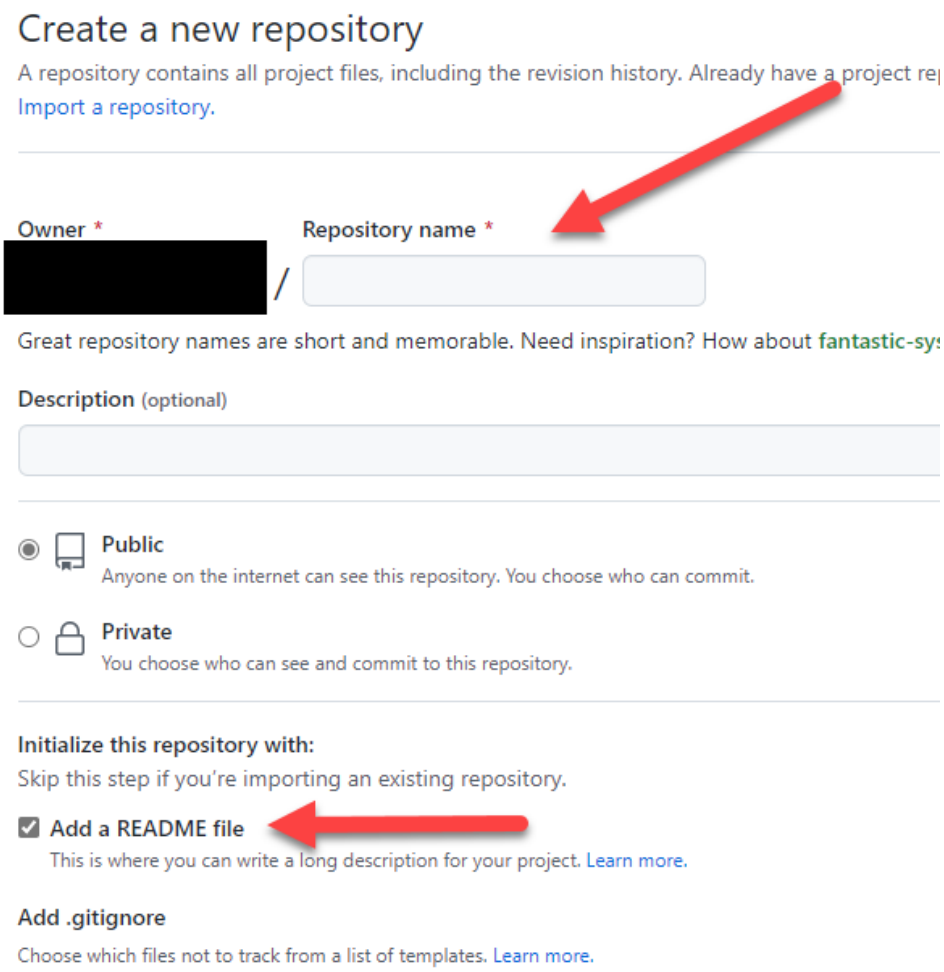
Exercise 05 - Working with GitHub

Overview

The purpose of this exercise is to see how we can work with a central Git server. We will be cloning, pushing, and pulling from GitHub.

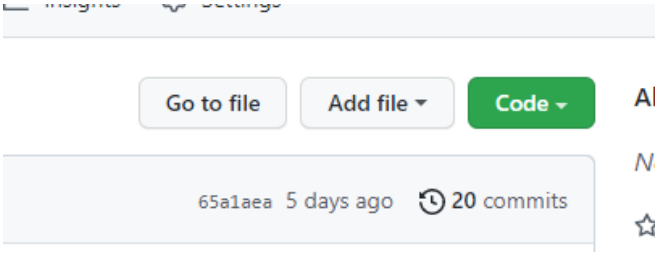
Step 1 - Setup GitHub

1. Navigate to GitHub and either log in if you have an existing account, or sign up for a new account.
2. In GitHub, create a new repository and give it a name, and tick the **Add a README file** option as well.



Step 2 - Clone the repo

1. In GitHub, click the **Code** button, and copy the repo's URL.



2. At a command prompt, clone the repo to your labs folder.

```
cd /labs
git clone <github repo url>
```

3. Display the status/log to see the local repo is a copy of the server repo.
4. Make some local changes (add a file or two, edit the README file), commit the changes. Repeat two or three times.
5. Push the changes to the server.

```
git push
```

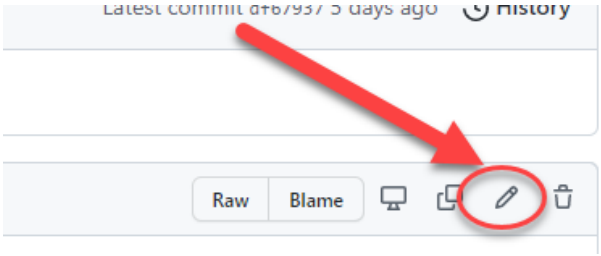
6. Return to GitHub in the browser and refresh the page. You should see your changes appear.

Step 3 - If time permits

1. Create a local branch (or two!), commit some changes to the local branch(es).
2. Push all branches to GitHub.

```
git push --all
```

3. Try editing a file directly in GitHub - click on any file (such as the README file) and click the **Edit** icon.



4. Make some changes via the browser and then commit them.
5. From your console, pull these changes to your local repo.

```
git pull
```

6. Try getting the local status/log to observe what has happened.

Exercise 06 - Collaborating with GitHub

Overview

The purpose of this exercise is to see how multiple people can work with a central Git server. As with Exercise 05, we will be cloning, pushing, and pulling from GitHub, but this will be done in groups of two or more.

Your tutor will arrange a breakout session so that you will be in a group with at least one other person. To make the instructions easier to follow, let's suppose your name is Sam and you are in a team with Chris and Jude. Obviously use your real name, and those of your teammates.

Step 1 - Setup GitHub

1. In GitHub create a new repository and give it the name Collab. (Chris and Jude do the same in their GitHub accounts.)
2. Click on **Add file**, **Create new file**. Name the file **shared.txt**
3. Edit shared.txt to add "First line by Sam." (Chris and Jude will add "First line by Chris." and "First line by Jude." respectively in their shared.txt file.) Commit the changes.

Step 2 – Clone other repos

1. In Git Bash, create a folder called Sam or Mine. Clone your own Collab repo.
2. Create a folder called **Chris**, and clone Chris's Collab repo to it. (Chris will clone Jude's Collab repo, and Jude will clone your Collab repo.)
3. Edit shared.txt. It should already include the text "First line by Chris." Add "**Second line by Sam**." Commit the changes and push.
4. Create a folder called **Jude**, clone Jude's Collab repo to it, and edit shared.txt to add "**Third line by Sam**." Check to see that the previous lines are as expected.

Step 3 – Pull repos

1. In Git Bash, return to your own Collab. Check that shared.txt contains the line "First line by Sam." and nothing else.
2. Use the pull command rather than the clone command.
3. Edit shared.txt. The file should now have all of the lines added.
4. Enter:

   ```
   git log --oneline
   ```

   to confirm that all of the commits are accessible to you.