TASK

Git Basics

Visit our website

Introduction

Welcome to The Git Basics Task!

In this task, we dive into using Git and discuss the basic commands you will need in order to use this tool. We will explore how to set up a repository for a new or existing project, use common Git commands, commit a modified file, view your project's history and branching.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact your mentor to get support on any aspect of your course.

The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with your mentor. You can also schedule a call or get support via email.

Your mentor is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

GETTING A GIT REPOSITORY

There are two ways to get a Git project. You can either initialise a new repository or clone an existing repository.

Initialising a Repository

To create a new repository, you have to initialise it using the `init` command. To do this, firstly open your terminal (or Command Prompt if you are using Windows) and go to your project's directory. To change your current directory, you use the `cd` (change directory) command followed by the pathname of the directory you wish to access.

After you have navigated to your project's directory, enter the following command:

```
git init
```

This creates a new, hidden subdirectory called .git in your project directory. This is where Git stores all of your necessary repository files, such as its database and configuration information, so that you can track your project.

Cloning a Repository

If you would like to get a copy of an existing Git repository, such as a project you would like to contribute to, you need to use the Git `clone` command. Running a Git clone command pulls a complete copy of the remote repository to your local system. In order to use this command, simply enter, `git clone [repository_url]` into the terminal or command prompt. For example, if you would like to clone the Wikimedia Commons Android App repository, you would enter the following:

```
git clone https://github.com/commons-app/apps-android-commons.git
```

This creates a new directory called "apps-android-commons", initialises a .git directory within it and pulls all the data from the remote repository. If you go to this new directory, you will find all of the project files, ready to be worked on.

ADDING A NEW FILE TO THE REPOSITORY

Now that your repository has been cloned or initialised, you can add new files to your project using the `git add` command.

Assume that you have set up a project at /Users/user/your_repository and that you have created a new file called newFile.js. To add newFile.js to the repository staging area, you would need to enter the following into your terminal or command prompt:

```
cd /Users/user/your_repository
git add newFile.js
```

CHECKING THE STATUS OF YOUR FILES

Files can either exist in a tracked state or in an untracked state in your working directory. Tracked files are files that were in the last snapshot, while untracked files are any files in your working directory that were not in your last snapshot and are not currently in the staging area. We use the `git status` command to determine which files are in which state. Using the `git add` command begins tracking a new file. If you run the `git status` command after you have added newFile.js, you should see the following code, showing that newFile.js is now tracked:

```
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)

    new file:   newFile.js
```

You can tell that newFile.java is staged because it is under the “Changes to be committed” heading.

COMMITTING YOUR CHANGES

You should now be ready to commit your staged snapshot to the project history using the `commit` command. If you have edited any files and have not run `git add` on them, they will not go into the commit. To commit your changes, simply enter the following:

```
git commit -m “added new file newFile.js”
```

The message after the -m flag inside the quotation marks is known as a commit message. Every commit needs a meaningful commit message. This makes it easier for other people who might be working on the project (or even for yourself later on) to understand what modifications you have made. Your commit message should be short and descriptive and you should write one for every commit you make.

VIEWING THE CHANGE HISTORY

Git saves every commit that is ever made in the course of your project. To see your repository or change history over time, you need to use the **`git log`** command. Running the **`git log`** command shows you a list of changes in reverse chronological order, meaning that the most recent commit will be shown first. The **`git log`** command displays the commit hash (which is a long string of letters and numbers that serves as an unique ID for that particular commit), the author’s name and email, the date written and the commit message.

Below is an example of what you might see if you run git log:

```
git log
commit a9ca2c9f4e1e0061075aa47cbb97201a43b0f66f
Author: Hyperion Student <hyperionstudent@gmail.com>
Date: Mon Sep 8 6:49:17 2017 +0200

Initial commit.
```

There are a large number and variety of options to the **`git log`** command that enable you to customise or filter what you would like to see. One extremely useful option is **`--pretty`** which changes the format of the log output. The **`oneline`** option is one of the number of prebuilt options available for you to use in conjunction to **`--pretty`**. This option simply displays the commit hash and commit message on a single line. This is particularly useful if you have many commits.
Below is an example of what you might see if you run **`git log --pretty=oneline`**:

```
git log --pretty=oneline
A9ca2c9f4e1e0061075aa47cbb97201a43b0f66f  Initial commit.
```

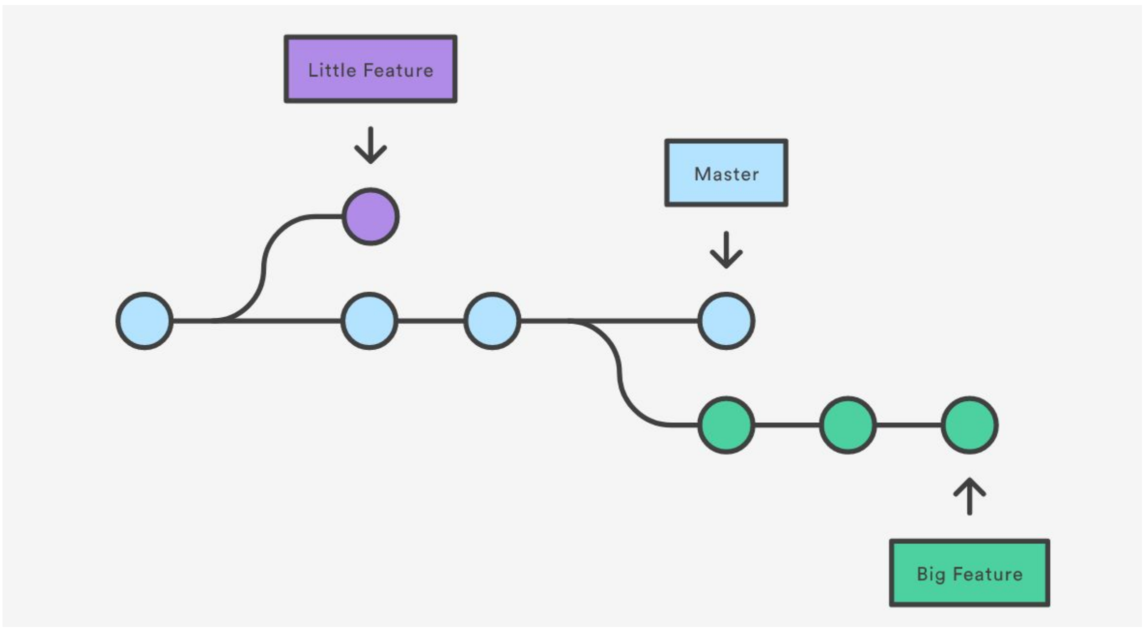
For the full set of options, you can run `git help log` from your terminal or command prompt or take a look at the reference documentation.

BRANCHES

It is extremely common for a number of developers to share and work on the same source code. Since different developers will have to be able to work on different parts of the code at the same time, it is important to be able to maintain different versions of the same codebase. This is where branching comes in.

One of the fundamental aspects of working with Git is branching. A branch represents an independent line of development. It allows each developer to branch out from the original codebase and isolate their work from others. By branching, you diverge from the main line of development and continue to work without messing up or disrupting the main line.

Branches are essential when working on new features or bug fixes. You create a new branch whenever you add a new feature or fix a bug to encapsulate your changes. This ensures that unstable code is not committed to the main codebase and also enables you to clean up your feature's history before merging it into the main branch.



Example of Branching (atlassian.com)

The image above visually represents the concept of branching. It shows a repository with two branches; one for a small feature and one for a larger feature. As you can see, each branch is an isolated line of development which can be worked on in parallel and keeps the main branch, known as the master branch, free from dubious code.

Git creates a master branch automatically when you make your first commit in a repository. Until you decide to create a new branch and switch over to it, all following commits will go under the master branch. You are therefore always working on a branch.

The HEAD is used by Git to represent the current position of a branch. By default, the HEAD will point to the master branch for a new repository. Changing where the HEAD is pointing will update your current branch. You can check where the HEAD is at the moment you use the **`git status`** command which will tell you in the first line of output.

Creating a Branch

In order to create a new branch, you use the **`git branch`** command, followed by the name of your branch. For example:

```
git branch my-first-branch
```

Switching Branches

Using the **`git branch`** command does not switch you to the new branch, it only creates the new branch. In order to switch to the new branch you created, you need to use the **`git checkout`** command.

```
git checkout my-first-branch
```

Using this command moves the HEAD to the my-first-branch branch.

Alternatively, you can run the **`git checkout`** command with a -b switch to create a branch and switch to it at the same time. For example:

```
git checkout -b my-second-branch
```

This is short for:

```
git branch my-second-branch
git checkout my-second-branch
```

Saving Changes Temporarily

When you make a commit, you save your changes permanently in the repository. However, you might find that you would like to save your local changes temporarily. For example, imagine you are working on a new feature when you are suddenly required to make an important bug fix right away. Obviously the changes you made so far for your feature don't belong to the bug fix you are going to make. Fortunately, with Git, you don't have to deploy your bug fix with the new feature changes you have made. All you have to do is switch back to the master branch.

Before you switch to the master branch, however, you should first make sure that your working directory or staging area has no uncommitted changes in it otherwise Git will not let you switch branches. It is therefore better to have a clean working slate when switching branches. To work around this issue we use the **`git stash`** command.

git stash

The stash command takes all the changes in your working copy and saves them on a clipboard. This leaves you with a clean working copy. Later, when you want to work on your feature again, you can restore your changes from the clipboard in your working copy.
To restore your saved Stash you can either:

- Get the newest Stash and clear it from your Stash clipboard by using **`git stash pop`**.
- Use **`git stash apply <stashname>`**, which will give you the specified stash but it will remain saved.

Merging

When you are done working on your new feature or bug fix in an isolated branch, it is important to merge it back into the master branch. The **`git merge`** command lets you take an independent line of development created by git branch and integrate it into a single branch.

To perform a merge you just need to:

- Check out the branch that you would like to use to receive the changes.
- Run the git merge command with the name of the branch you would like to merge.

  ```
  git checkout master
  ```

```
git merge my-first-branch
```

The above example merges the branch, my-first-branch into the master branch.

A note from our coding mentor
Nkosi

Check out [this brief overview]() of the advantages of using git. As a web developer it is important to learn the lingo so this post is great because tt also provides definitions for a number of key terms related to version control systems.

Instructions

Before you get started we strongly suggest you start using Notepad++ or IDLE to open all text files (.txt) and python files (.py). Do not use the normal Windows notepad as it will be much harder to read.

First read example.py, open it using Notepad++ (Right click the file and select 'Edit with Notepad++') or IDLE.

- example.py should help you understand some simple Python. Every task will have example code to help you get started. Make sure you read all of example.py and try your best to understand.
- You may run example.py to see the output. Feel free to write and run your own example code before doing the Task to become more comfortable with Python.
- You are not required to read the entirety of Additional Reading.pdf, it is purely for extra reference.

Compulsory Task 1

Follow these steps:

- Create an empty folder called task1_project.
- Open your terminal or command prompt and then change directory (**`cd`**) to your newly created folder.
- Enter the **`git init`** command to Initialise your new repository.
- Enter the **`git status`** command and make note of what you see. You should have a clean working directory.
- Create a new file in the task1_project folder called helloWorld.js and write a program that prints out the message "Hello World!"
- Run the **`git status`** command again. You should now see that your helloWorld.js file is untracked.
- Enter the **`git add`** command followed by helloWorld.js to start tracking your new file.
- Once again, run the **`git status`** command. You should now see that your helloWorld.js file is tracked and staged to be committed
- Now that it is tracked, let us change the file helloWorld.js. Change the message printed out by the program to "Git is Awesome!"

- Run **`git status`** again. You should see that helloWorld.js appears under a section called “Changes not staged for commit”. This means that the file is tracked but has been modified and not yet staged.
- To stage your file, simply run **`git add`** again.
- If you run **`git status`** again you should see that it is once again staged for your next commit.
- You can now commit your changes by running the **`git commit -m`** command. Remember to enter a suitable commit message after the -m switch.
- Running the **`git status`** command should show a clean working directory once again.
- Now run the **`git log`** command. You should see your commit listed. Take a screenshot of the output and send it to your mentor.

Compulsory Task 2

Follow these steps:

- Open your terminal or command prompt and change directory (**`cd`**) to the folder task1_project created above.
- Create a new branch called issue-1 using the **`git branch`** command.
- Switch to your new issue-1 branch by using the **`git checkout`** command.
- Once you are on the the issue-1 branch, change the helloWorld.js file. Modify your program to accept input from the user and then print out the inputted data.
- Add and commit your changes.
- Checkout the master branch and use the **`git merge`** command to merge branches.
- Take a screenshot of the output after running the **`git merge`** command and send it to your mentor.

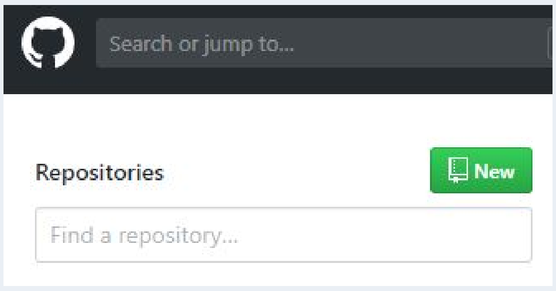
Instructions

Feel free to refer to the git cheatsheet in the 'Additional reading' folder for this task as needed for this or any future tasks in which you use git.

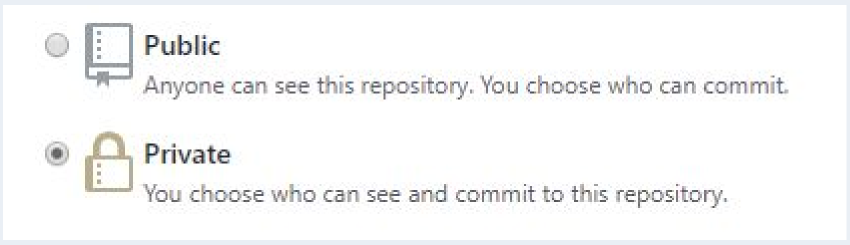
Compulsory Task 3

Follow these steps:

- Login to GitHub using the account you created in the previous task.
- Create a new repository by selecting the 'New' button as shown in the image below.



- Name the repository 'gitTask' and make sure that it is private.

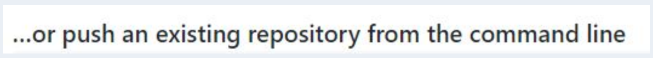


○ Public
Anyone can see this repository. You choose who can commit.

● Private
You choose who can see and commit to this repository.

- You now have two git repositories. The one you have been using in the previous compulsory tasks on your local machine and this remote repository that you have just made on gitHub. Push the repository on your local machine to the remote repository on GitHub by following these steps:
  - Open your terminal or command prompt and change directory (`cd`) to the folder task1_project created above.
  - Add your remote repository using the following command:
    `git remote add [shortname] [url]`
    E.g. `git remote add task4 https://github.com/Hyperion/gitTask.git`.

    Now you can use the short name (e.g. task4) on the command line in lieu of the whole URL. The URL will be indicated under the heading shown below once you have created your repository on GitHub.


...or push an existing repository from the command line

- Push your local repository to your remote repository using the following command:
  `git push [remote-name] [branch-name]`
  E.g. `git push -u task4 master`

- Invite your mentor to be a collaborator to the repository you have created. Help here.
- Once your mentor has marked this task as complete (and not before!) you can delete the repository that you have created here since it doesn't store any meaningful application code. Help here.

Once you have completed the task in line with the instructions above, click the button below to request your mentor to review your work and provide feedback. If you have any questions while attempting the task, leave your mentor a note on the comments.txt file in your Student Dropbox folder.

Completed the task(s)?

Ask your mentor review your work!

Review work

Rate us

Share your thoughts

Hyperion strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think the content of this task, or this course as a whole, can be improved or think we've done a good job?

Click here to share your thoughts anonymously.