**Lab 6 Makefile, Terraform, Github, Gist**

**ปฏิบัติการนี้มี 4 ตอน ดังนี้**

**ตอนที่ 1 แนะนำ Makefile**

* 1. **C**
  2. **Python**

**ตอนที่ 2 แนะนำ IaC สำรวจ ไฟล์ .tf**

**ตอนที่ 3 Terraform Fundamentals**

**Link:** [**https://www.cloudskillsboost.google/focuses/1208?parent=catalog**](https://www.cloudskillsboost.google/focuses/1208?parent=catalog)

**Video:** [**https://youtu.be/-bl-XylU3R0**](https://youtu.be/-bl-XylU3R0)

**ตอนที่ 4 ใช้ GitHub สร้าง repository และ gist**

## **ตอนที่ 1 แนะนำ Makefile**

**Makefile** คือไฟล์กำหนดค่าที่ยูทิลิตี make ใช้ มักใช้เพื่อระบุตำแหน่งของไฟล์ต้นฉบับที่จะใช้ รวบรวม หรือเชื่อมโยงกันในแอปพลิเคชันหนึ่ง ๆ มีลักษณะเป็น script คล้ายกับการทำแผนที่ขั้นตอนที่เกี่ยวข้องในแอปพลิเคชัน และทำให้การดำเนินการตามขั้นตอนต่าง ๆ มี consistency

These are issues Makefiles were invented to solve:

Implementing the same basic functionality over and over again:

* Parsing input parameters and environment variables.
* Manually managing dependencies between build steps.
* Error handling (...maybe).

Making the same basic mistakes:

* Incorrectly handling [input parameters](https://www.pixelbeat.org/programming/shell_script_mistakes.html) and [environment variables](https://en.wikipedia.org/wiki/Shellshock_(software_bug)).
* Missing dependencies between build steps.
* [Forgetting to handle errors](https://www.davidpashley.com/articles/writing-robust-shell-scripts/) and — even worse — carrying on with the program execution.

By plainly writing down the dependencies between the steps, make helps us to execute them correctly.

* **The first target in a Makefile will be executed by default when we call make**.
* The order of the targets does not matter.
* Shell commands must be indented with a tab.
* Add an @ sign to suppress output of the command that is executed.

Each build step in a Makefile has the following structure:

target: [dependencies]

<shell command to execute>

<shell command to execute>

...

**ตัวอย่างที่ 1** ใช้ไฟล์ ***dressing*** เป็น Makefile โดยใช้คำสั่ง cp dressing makefile

dress: trousers shoes jacket

@echo "All done. Let's go outside!"

jacket: pullover

@echo "Putting on jacket."

pullover: shirt

@echo "Putting on pullover."

shirt:

@echo "Putting on shirt."

trousers: underpants

@echo "Putting on trousers."

underpants:

@echo "Putting on underpants."

shoes: socks

@echo "Putting on shoes."

socks: pullover

@echo "Putting on socks."

$ make dress

Putting on underpants.

Putting on trousers.

Putting on shirt.

Putting on pullover.

Putting on socks.

Putting on shoes.

Putting on jacket.

All done. Let's go outside!

$ make jacket

Putting on shirt.

Putting on pullover.

Putting on jacket.

Note There must be a tab ( ) at the beginning of any command in Makefile.

.

**ตัวอย่างที่ 2** ใช้ Makefile ในการจัดการโปรแกรมภาษา C

***ไฟล์ hellofunc.h***

/\* example include file \*/

void myPrintHelloMake(void);

***ไฟล์ hellomake.c***

#include <stdio.h>

#include "hellofunc.h"

int main() {

// call a function in another file

myPrintHelloMake();

return(0);

}

***ไฟล์ hellofunc.c***

#include <stdio.h>

void myPrintHelloMake(void) {

printf("Hello makefiles!\n");

return;

}

คำสั่งในการสร้าง executable file ปกติ

gcc -o hellomake hellomake.c hellofunc.c -I.

***ไฟล์ makefile1***

hellomake: hellomake.c hellofunc.c

gcc -o hellomake hellomake.c hellofunc.c -I.

The list of files (hellomake.c hellofunc.c) on which the command depends on the first line after the :

ใช้ไฟล์ makefile1 เป็น Makefile โดยใช้คำสั่ง cp makefile1 makefile ลองใช้คำสั่ง make และ make hellomake และดูผลลัพธ์ร่วมกับคำสั่ง ls

***ไฟล์ makefile2***

CC=gcc

CFLAGS=-I.

hellomake: hellomake.o hellofunc.o

$(CC) -o hellomake hellomake.o hellofunc.o

clean:

rm \*.o

In particular, the macro CC is the C compiler to use, and CFLAGS is the list of flags to pass to the compilation command. ใช้ไฟล์ makefile1 เป็น Makefile โดยใช้คำสั่ง cp makefile2 makefile ลองใช้คำสั่ง make และ make hellomake และ make clean และดูผลลัพธ์ร่วมกับคำสั่ง ls

***ไฟล์ makefile3***

CC=gcc

CFLAGS=-I.

hellofunc.o: hellofunc.c

$(CC) -c hellofunc.c $(CFLAG)

hellomake.o: hellomake.c hellofunc.h

$(CC) -c hellomake.c $(CFLAG)

makeall: hellomake.o hellofunc.o

$(CC) -o hellomake hellomake.o hellofunc.o

clean:

rm \*.o

ใช้ไฟล์ makefile3 เป็น Makefile โดยใช้คำสั่ง cp makefile3 makefile ลองใช้คำสั่งต่อไปนี้และดูผลลัพธ์ร่วมกับคำสั่ง ls

make hellofunc.o

make hellomake.o

make makeall

make clean

***ไฟล์ makefile4***

CC=gcc

CFLAGS=-I.

DEPS = hellomake.h

OBJ = hellomake.o hellofunc.o

%.o: %.c $(DEPS)

$(CC) -c -o $@ $< $(CFLAGS)

makeall: $(OBJ)

$(CC) -o $@ $^ $(CFLAGS)

clean:

rm \*.o

cleanall:

rm \*.o; rm hellomake

The special macros $@ and $^ are the left and right sides of the :, respectively.

The macro DEPS, which is the set of .h files on which the .c files depend.

All of the object files are listed as part of the macro OBJ.

ใช้ไฟล์ makefile4 เป็น Makefile โดยใช้คำสั่ง cp makefile4 makefile ลองใช้คำสั่งต่อไปนี้และดูผลลัพธ์ร่วมกับคำสั่ง ls

make hellofunc.o

make hellomake.o

make makeall

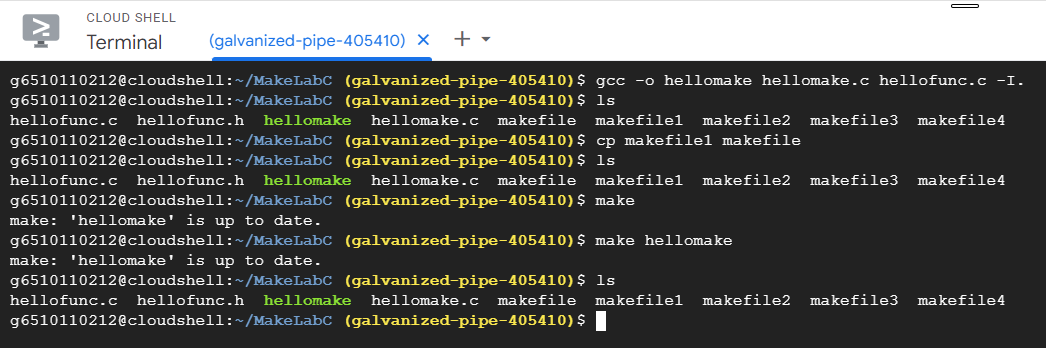
make clean

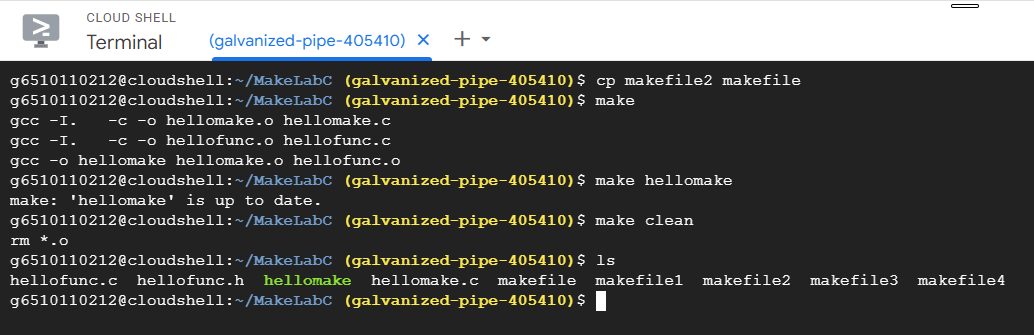
make cleanall

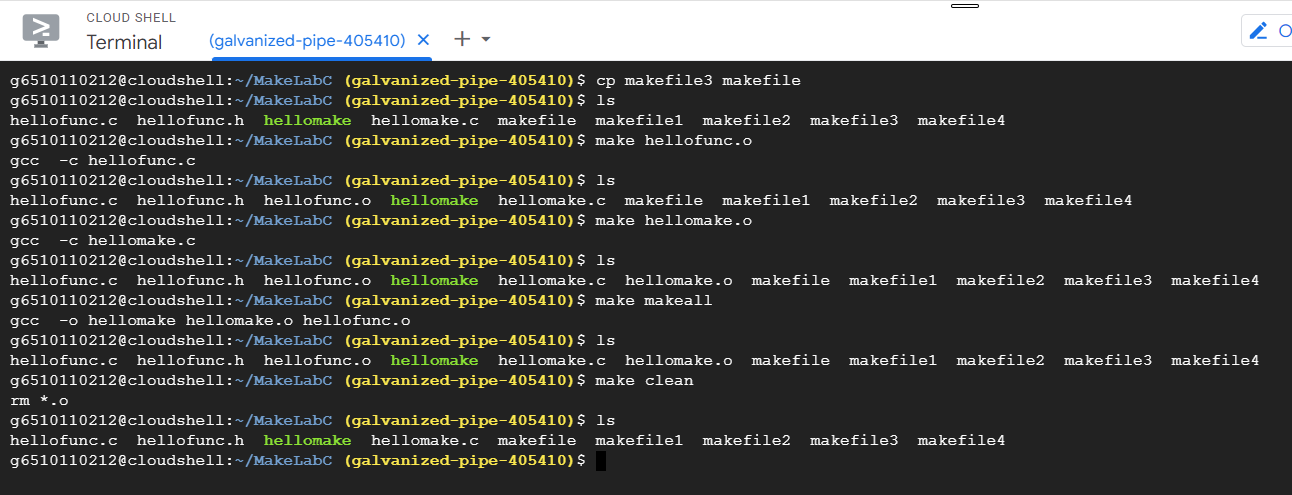
**Checkpoint 1.1**

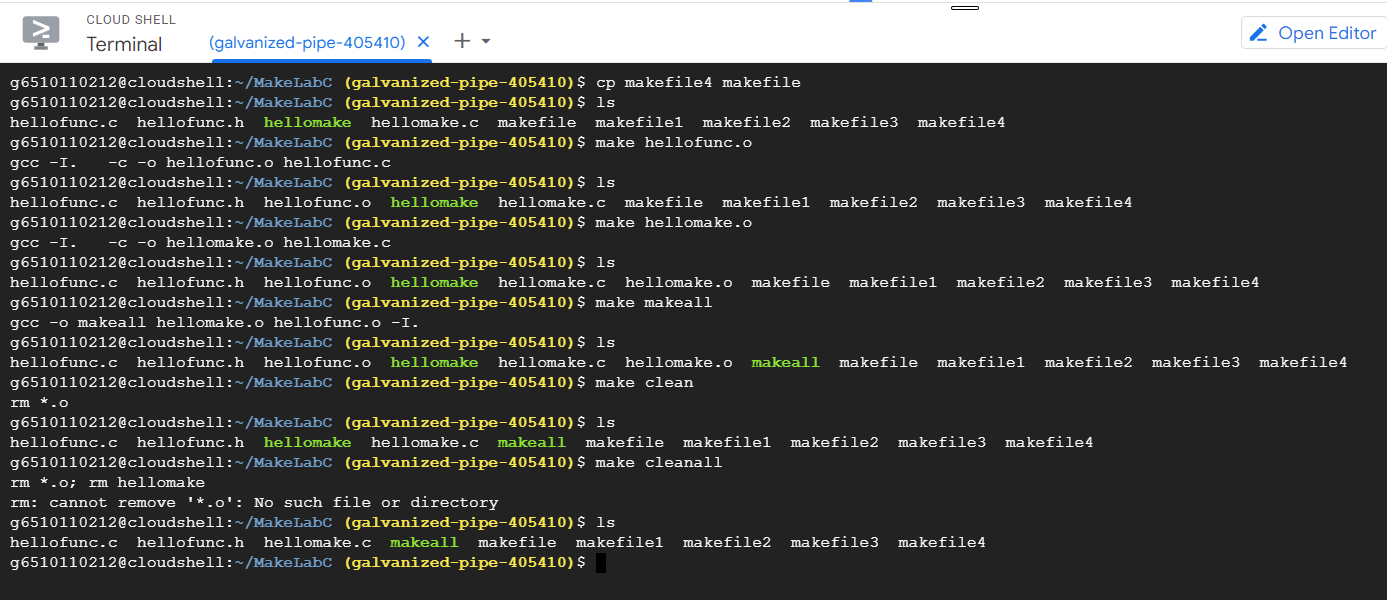
**ปรับใช้งาน** Makefile จากโปรแกรมภาษา C และไฟล์ makefile1-4 ที่กำหนดให้ บน **Google Cloud Shell**

ส่งตัวอย่างการใช้งาน และผลลัพธ์ มาประมาณ 1-2 หน้า









## **Example Makefile for Python**

**What is a Makefile and Why Do You Need it?**

A common reaction to hearing about a **Makefile** from an absolute beginner to Python is, "why do I need this?". Generally, it is healthy to have skepticism about things that appear to add work. In the case of a **Makefile**, though, the reason to use them in a project is that they are less work because they keep track of complicated build steps that are very difficult to remember and type out correctly.

pytest: It supports unittest test cases execution. It has benefits like supporting built in assert statement, filtering of test cases, returning from last failing test etc.

A great example is a lint step with the pylint tool. Lint, or a linter, is a tool that analyzes source code to flag programming errors, bugs, stylistic errors, and suspicious constructs. Pylint focuses on code analysis and style checking. It offers extensive customization options and supports various coding standards. Pylint is known for its comprehensive reports and ability to detect a wide range of code issues.

With a **Makefile**, you only need to run: **make lint**, and the same command can run inside a Continuous Integration server. The alternative approach is to *type out the full directive each time you need it*, such as the following.

**pylint --disable=R,C \*.py**

This sequence is very prone to errors and quite tedious to repeatedly typing over your project's life. Instead, it is much simpler to type the following:

**make lint**

Black is the uncompromising Python code formatter. Black comes into play not only does it report format errors but also fixes them.

When you embrace the **Makefile** approach, it simplifies your workflow and makes it easier to integrate your project into a continuous integration system. There is less code to type, and this is always a good thing for automation. Further, **Makefile** commands are recognized by shell auto-completion, making it easy to "tab-complete" the steps.

**หมายเหตุ ติดตั้ง black, pylint, pytest โดยใช้**

pip install black

pip install pylint

pip install pytest

## ***Example Makefile for Python***

install:

pip install –upgrade pip &&\

pip install -r requirements.txt

test:

python -m pytest -vv xxx.py

format:

black \*.py

lint:

pylint --disable=R,C yyy.py

all: install lint test

***ไฟล์ testblack.py***

def is\_unique(

s

):

s = list(s

)

s.sort()

for i in range(len(s) - 1):

if s[i] == s[i + 1]:

return 0

else:

return 1

if \_\_name\_\_ == "\_\_main\_\_":

print(

is\_unique(input())

)

***ไฟล์ testlint.py***

import numpy as np

import time

import pandas as pd

Captain='Picard'

def InitiateWarpSpeed(order):

if order=="engage":

print("initiating warp speed")

else:

print("you are not the captain of this vessel")

InitiateWarpSpeed("engage")

***ไฟล์ testlint.py***

# content of test\_sample.py

def func(x):

return x + 1

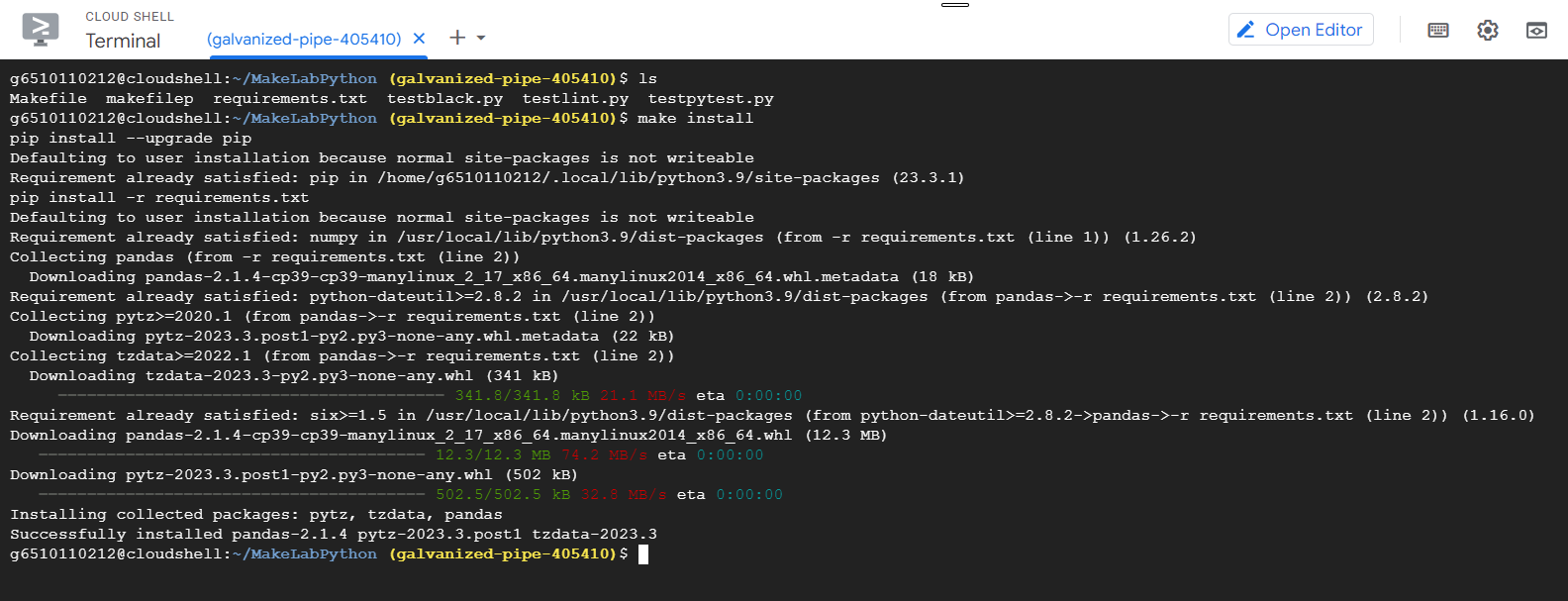
def test\_answer():

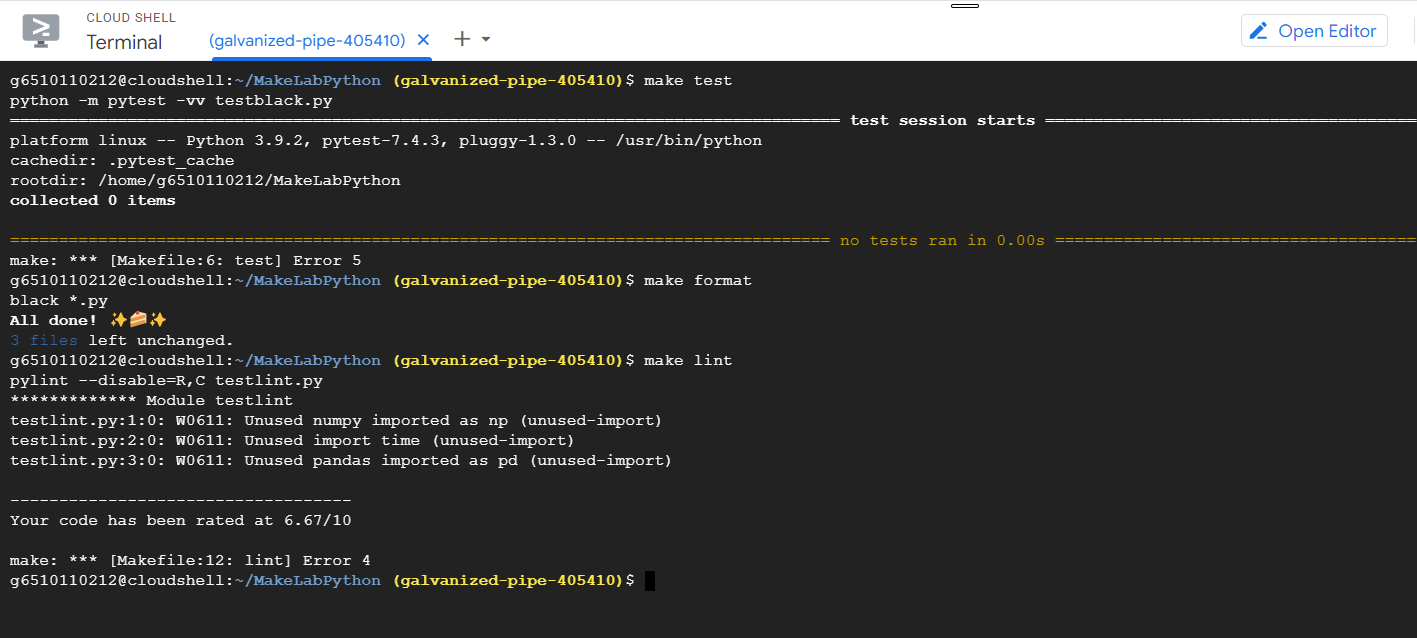
assert func(3) == 5

**Checkpoint 1.2**

**ปรับใช้งาน** Makefile สำหรับโปรแกรมภาษา Python (.py) ที่กำหนดให้ บน **Google Cloud Shell**

ส่งตัวอย่างการใช้งาน และผลลัพธ์ มาประมาณ 1-2 หน้า

****

****

**Information**

***Makefile for Python***

[Creating a Python Makefile - Earthly Blog](https://earthly.dev/blog/python-makefile/)

[python-project-template/Makefile at main · rochacbruno/python-project-template (github.com)](https://github.com/rochacbruno/python-project-template/blob/main/Makefile)

[Automate your Python project with Makefile (antonz.org)](https://antonz.org/makefile-automation/)

***PyTest***

[pytest: helps you write better programs — pytest documentation](https://docs.pytest.org/en/7.2.x/)

[Pytest - Quick Guide (tutorialspoint.com)](https://www.tutorialspoint.com/pytest/pytest_quick_guide.htm)

***PyLint***

[Pylint - code analysis for Python | www.pylint.org](https://pylint.org/)

[What is linting and how can it save you time? (freecodecamp.org)](https://www.freecodecamp.org/news/what-is-linting-and-how-can-it-save-you-time/)

[Using Pylint to write clean Python code | by Vishal Sharma | Towards Data Science](https://towardsdatascience.com/using-pylint-to-write-clean-python-code-660eff40ed8)

**Other Information**

***Makefile for C/C++***

[Makefile Tutorial By Example](https://makefiletutorial.com/)

[Top (GNU make)](https://www.gnu.org/software/make/manual/html_node/index.html#toc-An-Introduction-to-Makefiles)

[Makefile Cheatsheet | All tragedy erased, I see only wonders. (simphoni.uk)](https://diary.simphoni.uk/2022/02/04/makefile-cheatsheet/)

[Makefile Tutorial By Example](https://makefiletutorial.com/)

[Makefile Cheat Sheet](https://bytes.usc.edu/cs104/wiki/makefile/)

[Makefile cheat sheet · GitHub](https://gist.github.com/evertrol/4b6fd05f3b6be2b331c60638b1af7101)

[Makefile Cheatsheet - Daniel Imfeld](https://imfeld.dev/notes/makefile_cheatsheet)

[Makefile - Quick Guide (tutorialspoint.com)](https://www.tutorialspoint.com/makefile/makefile_quick_guide.htm)

[Makefile Tutorial By Example](https://makefiletutorial.com/)

[A Simple Makefile Tutorial (colby.edu)](https://cs.colby.edu/maxwell/courses/tutorials/maketutor/)

[Creating A Basic Make File for Compiling C Code - CodeProject](https://www.codeproject.com/articles/794764/creating-a-basic-make-file-for-compiling-c-code)

[Makefile example (sco.com)](http://osr507doc.sco.com/en/tools/make_example.html)

[Afraid of Makefiles? Don't be! | Matthias Endler](https://endler.dev/2017/makefiles/)

Examples of other works

[I Heart Make - Why, Scott, WHY?!? (ddrscott.github.io)](https://ddrscott.github.io/blog/2021/i-heart-make/)

Videos

[Makefile Symbols? Top Answer Update - Ar.taphoamini.com](https://ar.taphoamini.com/makefile-symbols-top-answer-update/)

<https://www.youtube.com/watch?v=zeEMISsjO38&embeds_referring_euri=https%3A%2F%2Fmakefiletutorial.com%2F&feature=emb_imp_woyt>

**ตอนที่ 2 แนะนำ IaC สำรวจ ไฟล์ .tf**

**Infrastructure as Code (IaC)**: This is code that's checked into a repo that deploys and Infrastructure.

Terraform:  a popular version of Infrastructure as Code.

* available to run on multiple Cloud platforms, including GCP, AWS, and Azure.

Environment Drift: a form of putting one environment into a state where you don't exactly know what's happening

That's what Infrastructure as Code solves, is stopping environments from drifting in changing because every time you do a deploy, it rectifies that environment.

Infrastructure as Code (IaC) is the management of infrastructure (networks, virtual machines, load balancers, and connection topology) in a descriptive model, using the same versioning as DevOps team uses for source code.

Like the principle that the same source code generates the same binary, an IaC model generates the same environment every time it is applied. IaC is a key DevOps practice and is used in conjunction withco ntinuous delivery.

**IaC Benefits**

* a best practice for doing cloud based deployments.
* the modern way of doing deployment,
* solves a lot of problems in terms of repeatability.

In terms of business continuity, if you don't have infrastructure as code, you can have some big problems in let's say a key employee leaves.

**Example of Terraform file (.tf)**

resource "google\_compute\_instance" "default" {

project = "starlit-casing-361707"

name = "terraform"

machine\_type = "n1-standard-1"

zone = "us-central-a"

boot\_disk {

initialize\_params {

image = "debian-cloud/debian-9"

}

}

network\_interface {

network = "default"

access\_config {

}

}

}

**Checkpoint 2** ส่ง ไฟล์ .tf และ .tfstate ที่พบใน GCP Cloud Shell ของนักศึกษา และอธิบายพอสังเขป

**ตอนที่ 3** Terraform Fundamentals

35 minutes1 Credit

**GSP156**

**Overview**

Terraform enables you to safely and predictably create, change, and improve infrastructure. It is an open source tool that codifies APIs into declarative configuration files that can be shared among co-workers, treated as code, edited, reviewed, and versioned.

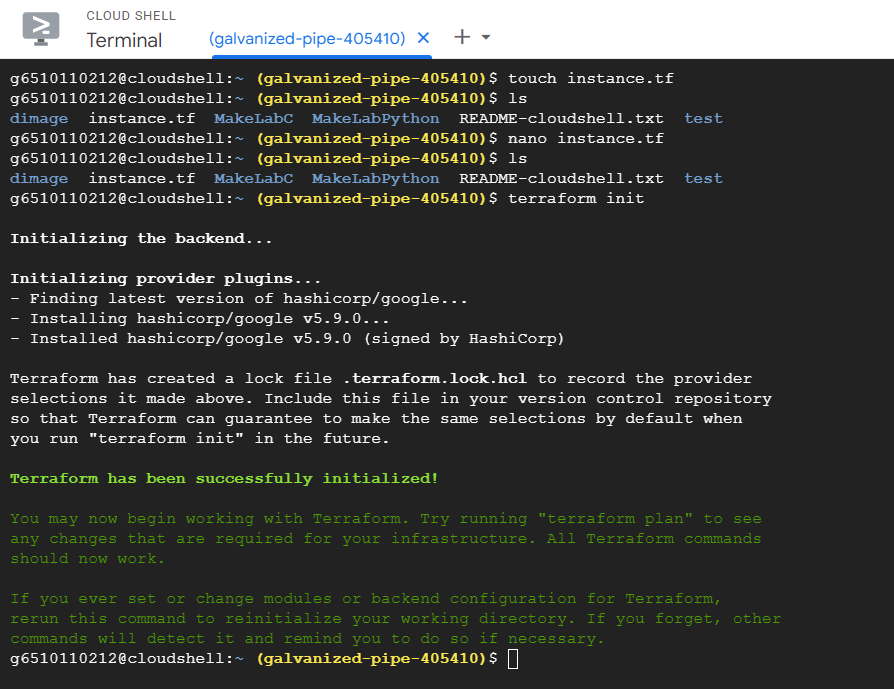
What you'll learn

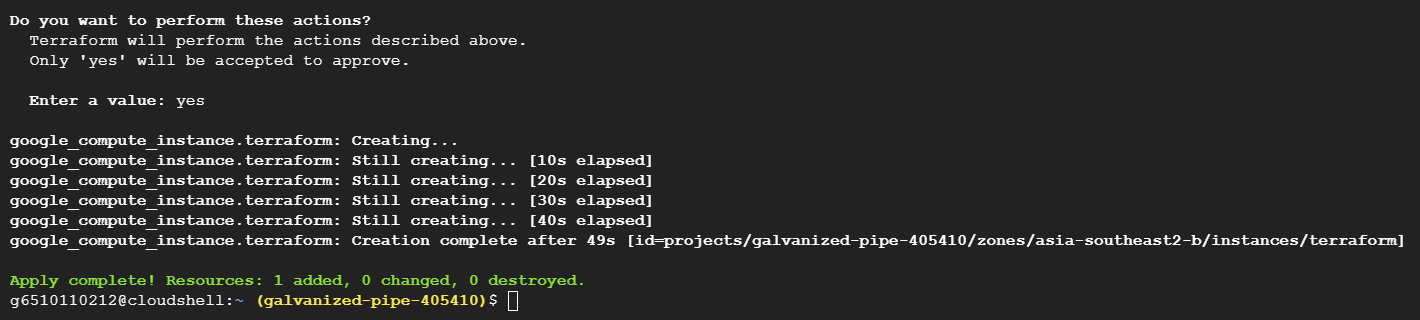
In this lab, you will learn how to perform the following tasks:

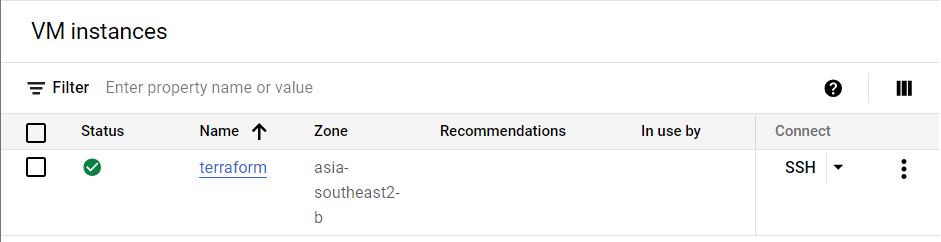
* Get started with Terraform in Google Cloud.
* Install Terraform from installation binaries.
* Create a VM instance infrastructure using Terraform.

**ดูลิงค์และวิดีโอใน ชีทไฟล์ 6.3**

**Checkpoint 3** ส่งผลลัพธ์ตามขั้นตอนใน GSP156 Terraform Fundamentals





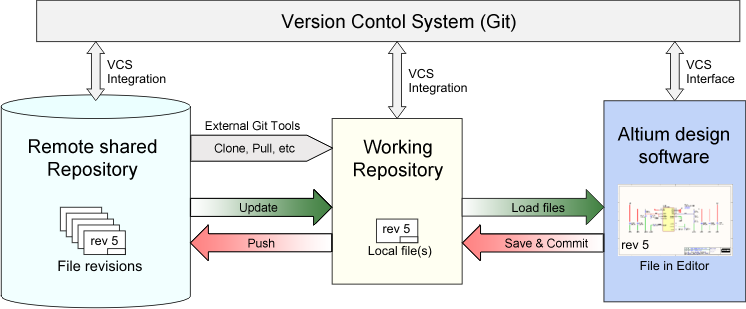


**Terraform** is an [infrastructure-as-code](https://en.wikipedia.org/wiki/Infrastructure_as_code) software tool created by [HashiCorp](https://en.wikipedia.org/wiki/HashiCorp).

* Users define and provide data center infrastructure using a declarative configuration language known as HashiCorp Configuration Language (HCL), or optionally [JSON](https://en.wikipedia.org/wiki/JSON).
* Terraform manages external resources (such as public cloud infrastructure, private cloud infrastructure, network appliances, [software as a service](https://en.wikipedia.org/wiki/Software_as_a_service), and [platform as a service](https://en.wikipedia.org/wiki/Platform_as_a_service)) with "providers".
* Users can interact with Terraform providers by declaring resources or by calling data sources.
* Rather than using [imperative commands](https://en.wikipedia.org/wiki/Imperative_programming) to provision resources, Terraform uses declarative configuration to describe the desired final state.
* Once a user invokes Terraform on a given resource, Terraform will perform [CRUD](https://en.wikipedia.org/wiki/Create,_read,_update_and_delete) actions on the user's behalf to accomplish the desired state.
* The infrastructure as code can be written as modules, promoting reusability and maintainability.
* Terraform supports a number of cloud infrastructure providers:
  + [Amazon Web Services](https://en.wikipedia.org/wiki/Amazon_Web_Services),
  + [Cloudflare](https://en.wikipedia.org/wiki/Cloudflare),
  + [Microsoft Azure](https://en.wikipedia.org/wiki/Microsoft_Azure),
  + [IBM Cloud](https://en.wikipedia.org/wiki/IBM_Cloud),
  + Serverspace,
  + [Selectel](https://en.wikipedia.org/wiki/Selectel) [Google Cloud Platform](https://en.wikipedia.org/wiki/Google_Cloud_Platform),
  + [DigitalOcean](https://en.wikipedia.org/wiki/DigitalOcean),
  + [Oracle Cloud Infrastructure](https://en.wikipedia.org/wiki/Oracle_Cloud),
  + [Yandex.Cloud](https://en.wikipedia.org/wiki/Yandex.Cloud),
  + [VMware vSphere](https://en.wikipedia.org/wiki/VMware_vSphere), and
  + [OpenStack](https://en.wikipedia.org/wiki/OpenStack).

**ตอนที่ 4 ใช้ GitHub สร้าง repository, gist และ workflow**

**Git** คือ ซอฟต์แวร์ Version Control เหมาะสำหรับผู้พัฒนาซอฟต์แวร์ ซึ่งสามารถใช้งานได้ทั้งในรูปแบบเดี่ยวหรือแบบกลุ่ม เป็นระบบที่จัดเก็บการเปลี่ยนแปลงของไฟล์ได้ โดยสามารถ กำหนดเป็น Project ต่าง ๆ และ backup งานต่าง ๆ ให้ได้ ผู้ใช้สามารถย้อนกลับไปดู version ต่าง ๆ ของ Project ที่ได้ ดังนั้น Version Control และตรวจสอบได้ว่าใครเป็นคนเขียนเริ่มต้นหรือใครเป็นคนเปลี่ยนแปลงแก้ไข เพิ่ม หรือลบอะไร Project ในส่วนต่าง ๆ



รูปที่ 1 ระบบการทำงานของ Git

(Source: https://www.freecodecamp.org/news/the-beginners-guide-to-git-github/)

**GitHub** คือ Platform หรือ Web Server ที่ให้บริการในการฝากไฟล์ Git ซึ่งจะทำให้ผู้ใช้สามารถใช้ Git ร่วมกับคนอื่นได้โดยผ่านเว็บไซต์ ซึ่งนิยมใช้กันมากใน Open Source Project ต่างๆ หรือในกลุ่มผู้พัฒนาซอฟต์แวร์

**There are four fundamental elements in the Git Workflow.**

* **Working Directory,**
* **Staging Area,**
* **Local Repository**
* **Remote Repository.**

**If you consider a file in your Working Directory, it can be in three possible states.**

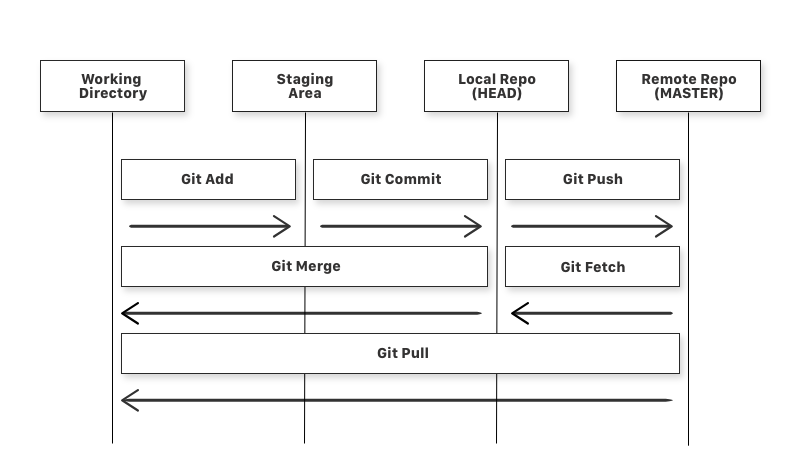
1. **It can be staged.** Which means the files with the updated changes are marked to be committed to the local repository but not yet committed.
2. **It can be modified**. Which means the files with the updated changes are not yet stored in the local repository.
3. **It can be committed**. Which means that the changes you made to your file are safely stored in the local repository.

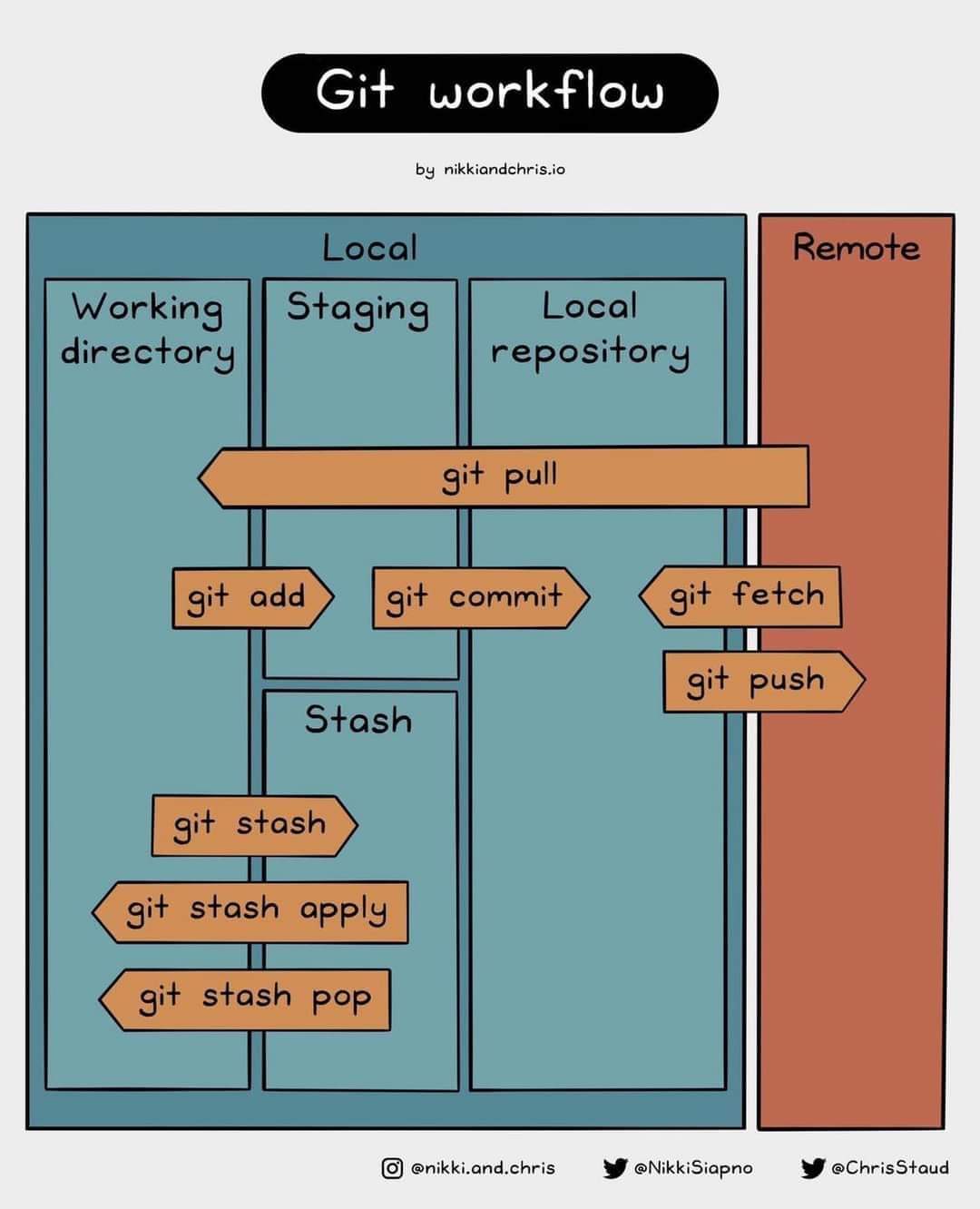
**Basic Git Commands**

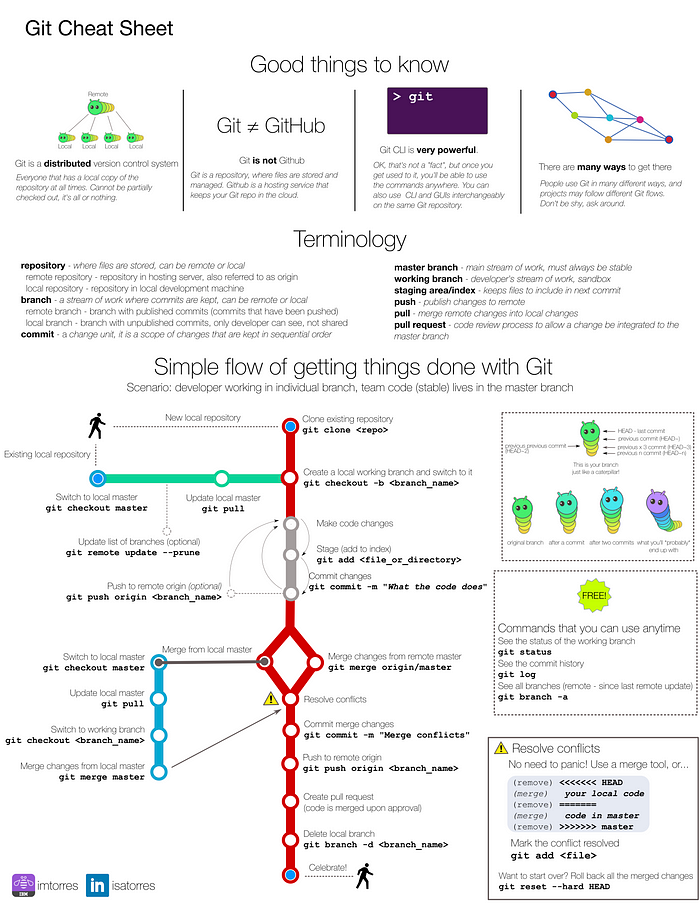
* git add is a command used to add a file that is in the working directory to the staging area.
* git commit is a command used to add all files that are staged to the local repository.
* git push is a command used to add all committed files in the local repository to the remote repository. So in the remote repository, all files and changes will be visible to anyone with access to the remote repository.
* git fetch is a command used to get files from the remote repository to the local repository but not into the working directory.
* git merge is a command used to get the files from the local repository into the working directory.
* git pull is command used to get files from the remote repository directly into the working directory. It is equivalent to a git fetch and a git merge .

**Gist**

* GitHub provides a platform to share an entire project as a repo among a team (private repo) or to the whole world (public repo).
  + A hosting service, the entire site, that houses a web-based git repository. It includes all the fucntionality of git with additional features added in.
  + Normally if you work on a project, you use GitHub.
* Gist is a simple way to share code snippets and pastes with others.
  + An additional feature added to github to allow the sharing of code snippets, notes, to do lists and more. You can save your Gists as secret or public.
  + More like a memo: It is used when you need to share a sample piece of code or technique with your co-workers or friends.

****

****



**คำสั่ง**

1. เปิดบัญชี GitHub โดยตั้งชื่อให้ประกอบด้วยรหัสนักศึกษาของตน
2. สร้าง Repository ใน Github เพื่อเชื่อมโยงกับ git (local)
3. เชื่อมต่อระหว่าง Local กับ Github โดยใช้คำสั่ง git remote add origin server <server>
4. ใน GitHub สร้าง gist เขียนฟังก์ชัน และตั้งชื่อ แชร์และสำเนา URL
5. สร้าง automated deployment workflow ตาม Link: How to build a CI/CD pipeline with GitHub Actions in four simple steps  
   <https://github.blog/2022-02-02-build-ci-cd-pipeline-github-actions-four-steps/>

**Checkpoint 4**

1. ส่ง URL ทั้ง Repository และ Gist ใน GitHub  
   ตัวอย่าง <https://gist.github.com/tpichaya/0900aa952223e5f42d5530aa58f9d42a>  
   ตัวอย่าง [tpichaya/gcp-flask-ml-deploy (github.com)](https://github.com/tpichaya/gcp-flask-ml-deploy)
2. ส่ง screenshot ของ workflow visualizer สำหรับ automated deployment workflow ที่สร้างขึ้นเองตามลิงค์ข้างบน

### Setup and use Github

To set up and use Github, you need a Github account and internet access. The minimal steps to start are:

1. Create a repository, for example, hello.
2. Add an [SSH key to your Github account](https://help.github.com/en/github/authenticating-to-github/adding-a-new-ssh-key-to-your-github-account).
3. Clone the repository locally, for example:
4. git remote add origin server Create a change and push it. This process would be an example of a good first change (inside the cloned repo).

echo "# hello" >> README.md

git add README.md

git commit -m "adding name of repo to README"

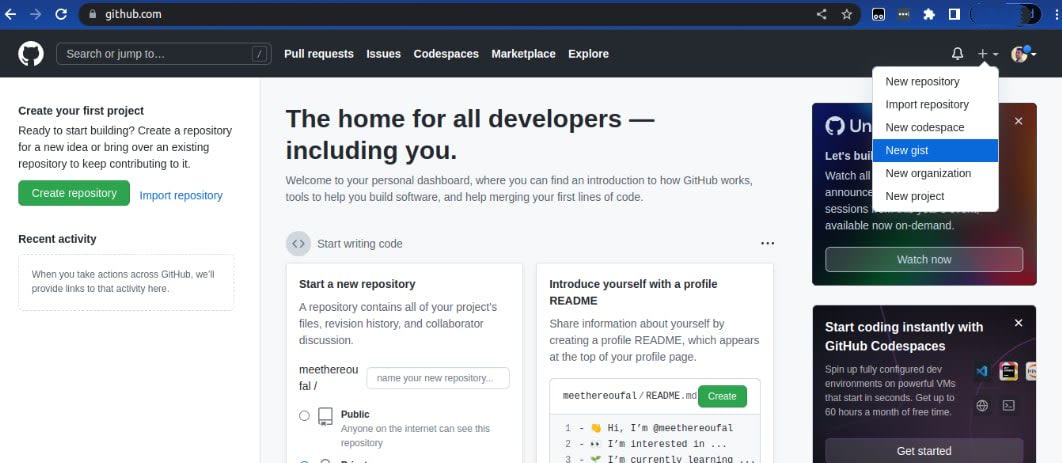
git push

### Create a gist

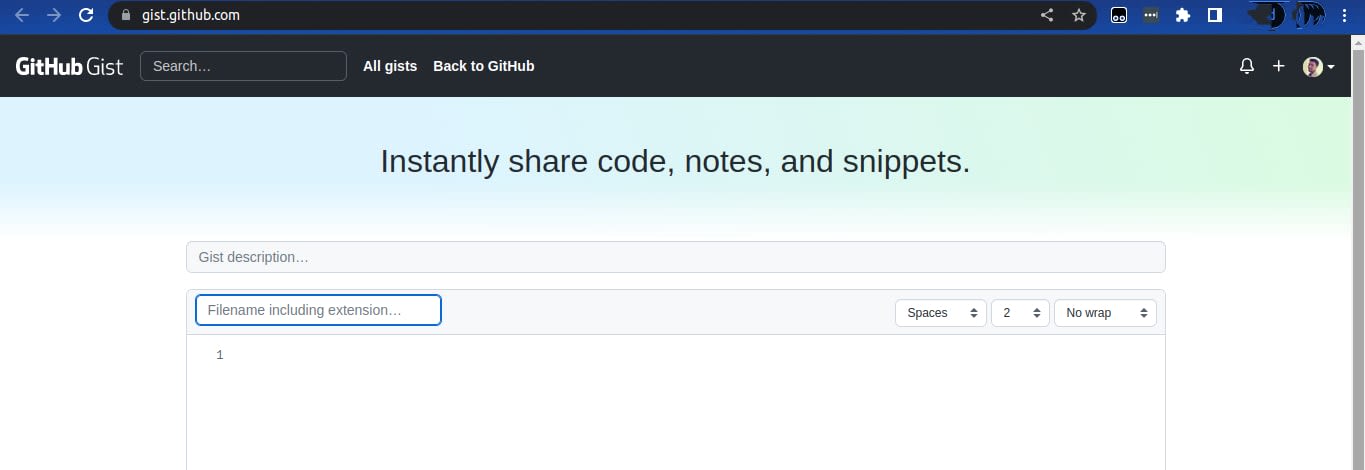
### You can use your gist homepage or GitHub account to create a gist.

#### Step 1:

From your [GitHub](https://github.com/) account, you can choose **New gist** from the Create Menu or the plus icon to the left of your profile icon in the upper right of the window.

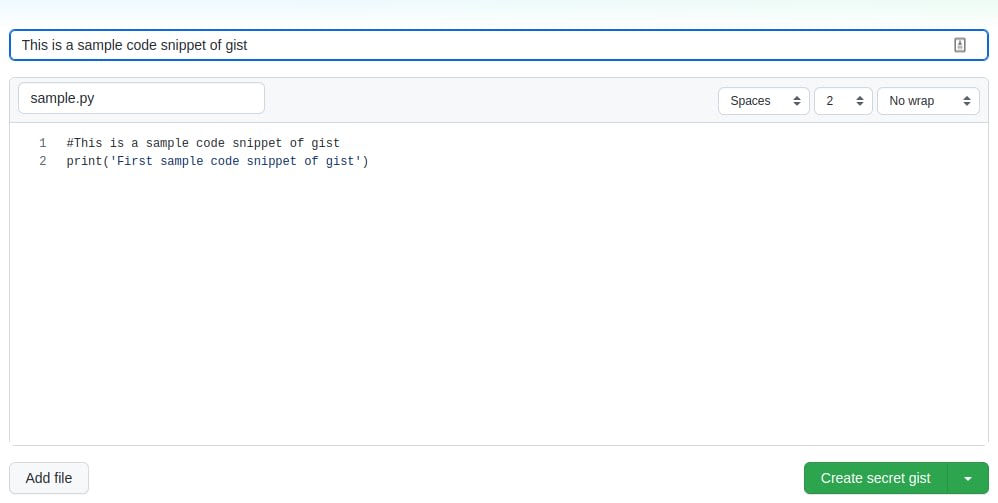


If you are already on the GitHub gist [website](https://gist.github.com/), you can easily create gists by clicking the **Create Menu** (plus icon).



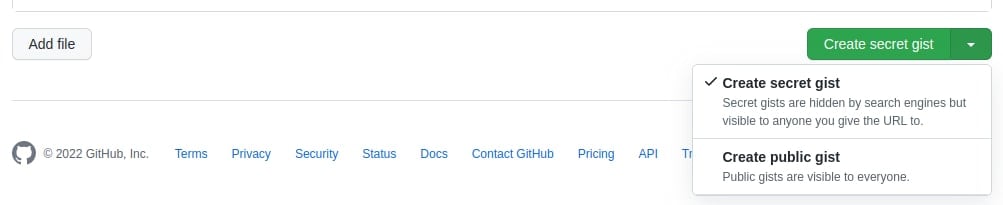
#### Step 2:

In the gist description section, you can enter a description of your gist (optional). In the Filename including extension section, enter the name of your file. Your file name should include the file extension, depending on the gist you create. For example, if you are creating a Python sample gist, you can add the .py extension at the end of your file, as in sample.py.

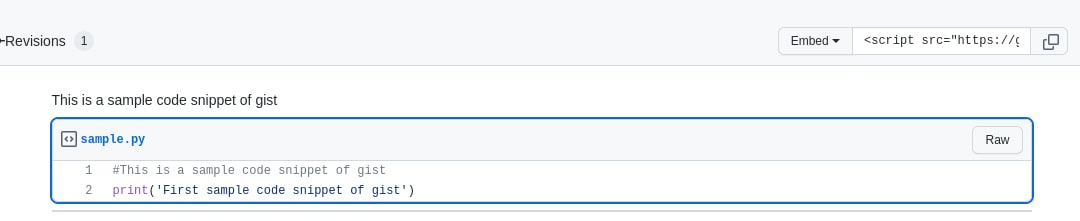


#### Step 3:

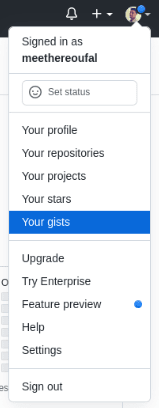
Enter your code in the numbered lines. Once you have completed your code, click **Create secret gist** or the down arrow next to it for further options. You can make your code public or secret after writing it. If you set it to secret, only the person with access to your gist URL can see it.



Your GitHub gist is now ready for sharing with whomever you choose. You can copy your gist's embed code if you want to embed it.



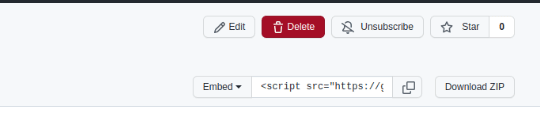
To view your gists, navigate to click your profile picture to open the dropdown menu and select **Your gists**.



Edit or Delete a gist

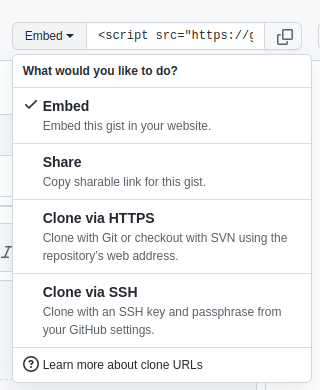
Here are some of the options you have with your GitHub gists in the upper right-hand of the window:

* Edit.
* Delete.
* Subscribe or Unsubscribe.
* Star.
* Embed, Copy, and Share.
* Download a raw or zipped copy.



You can use multiple methods to share a gist. You can:

* Embed it into a website.
* Copy a shareable link to send.
* Clone with Git or checkout with SVN using the repository's web address (Clone via HTTPS).
* Cloning with an SSH key and passphrase from your GitHub settings (Clone via SSH).



**More information**

***Git and GitHub***

[git - the simple guide - no deep shit! (rogerdudler.github.io)](http://rogerdudler.github.io/git-guide/)

[Set up Git - GitHub Docs](https://docs.github.com/en/get-started/quickstart/set-up-git)

[Hello World - GitHub Docs](https://docs.github.com/en/get-started/quickstart/hello-world)

[Adding a new SSH key to your GitHub account - GitHub Docs](https://docs.github.com/en/authentication/connecting-to-github-with-ssh/adding-a-new-ssh-key-to-your-github-account)

[An Intro to Git and GitHub for Beginners (Tutorial) (hubspot.com)](https://product.hubspot.com/blog/git-and-github-tutorial-for-beginners)

[Chapter01 Configuring the Environment | Pragmatic AI Labs and Solutions (paiml.com)](https://paiml.com/docs/home/books/testing-in-python/chapter01-configuring-the-environment/)

[วิธีใช้งาน Git และอัพไฟล์ขึ้น github ฉบับรวบรัด - Teekha Marketing (teekhawat.com)](https://teekhawat.com/%e0%b8%a7%e0%b8%b4%e0%b8%98%e0%b8%b5%e0%b9%83%e0%b8%8a%e0%b9%89%e0%b8%87%e0%b8%b2%e0%b8%99-git-%e0%b9%81%e0%b8%a5%e0%b8%b0%e0%b8%ad%e0%b8%b1%e0%b8%9e%e0%b9%84%e0%b8%9f%e0%b8%a5%e0%b9%8c%e0%b8%82/)

[ทำความรู้จัก Git & GitHub พร้อมการใช้งานร่วมกับ VS Code เบื้องต้น | by stackpython | Medium](https://stackpython.medium.com/%E0%B8%97%E0%B8%B3%E0%B8%84%E0%B8%A7%E0%B8%B2%E0%B8%A1%E0%B8%A3%E0%B8%B9%E0%B9%89%E0%B8%88%E0%B8%B1%E0%B8%81-git-github-%E0%B8%9E%E0%B8%A3%E0%B9%89%E0%B8%AD%E0%B8%A1%E0%B8%81%E0%B8%B2%E0%B8%A3%E0%B9%83%E0%B8%8A%E0%B9%89%E0%B8%87%E0%B8%B2%E0%B8%99%E0%B8%A3%E0%B9%88%E0%B8%A7%E0%B8%A1%E0%B8%81%E0%B8%B1%E0%B8%9A-vs-code-%E0%B9%80%E0%B8%9A%E0%B8%B7%E0%B9%89%E0%B8%AD%E0%B8%87%E0%B8%95%E0%B9%89%E0%B8%99-f848f41a39e9)

[เรียนรู้ Git และ Github ฉบับเด็กมหาลัย | by Thanatcha Kromsang | Medium](https://medium.com/@thanatchakromsang/%E0%B9%80%E0%B8%A3%E0%B8%B5%E0%B8%A2%E0%B8%99%E0%B8%A3%E0%B8%B9%E0%B9%89-git-%E0%B9%81%E0%B8%A5%E0%B8%B0-github-%E0%B8%89%E0%B8%9A%E0%B8%B1%E0%B8%9A%E0%B9%80%E0%B8%94%E0%B9%87%E0%B8%81%E0%B8%A1%E0%B8%AB%E0%B8%B2%E0%B8%A5%E0%B8%B1%E0%B8%A2-7311034c6527)

[Learn the Basics of Git in Under 10 Minutes (freecodecamp.org)](https://www.freecodecamp.org/news/learn-the-basics-of-git-in-under-10-minutes-da548267cc91/)

Git Stash

[Git - git-stash Documentation (git-scm.com)](https://www.git-scm.com/docs/git-stash)

[git stash - Saving Changes | Atlassian Git Tutorial](https://www.atlassian.com/git/tutorials/saving-changes/git-stash)

Video

[Learn the Basics of Git in Under 10 Minutes (freecodecamp.org)](https://www.freecodecamp.org/news/learn-the-basics-of-git-in-under-10-minutes-da548267cc91/)

***Gist***

[A Beginner’s Guide to Using GitHub Gists | Liquid Web](https://www.liquidweb.com/kb/how-to-use-github-gists/)

[Creating gists - GitHub Docs](https://docs.github.com/en/get-started/writing-on-github/editing-and-sharing-content-with-gists/creating-gists)

[What Is a GitHub Gist? (makeuseof.com)](https://www.makeuseof.com/share-code-github-gist/)

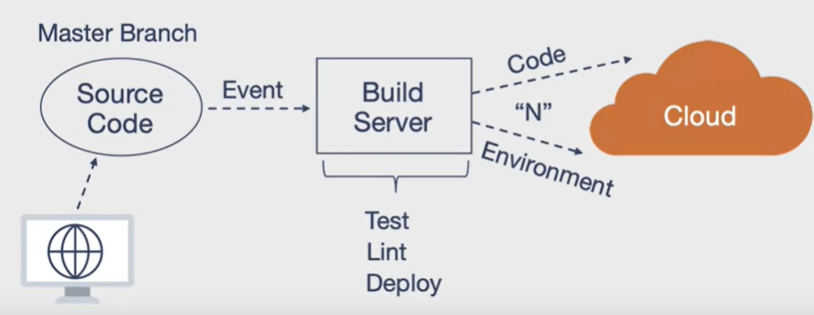
Discover Gists [Discover gists · GitHub](https://gist.github.com/starred)

***GitHub Actions and Workflow***

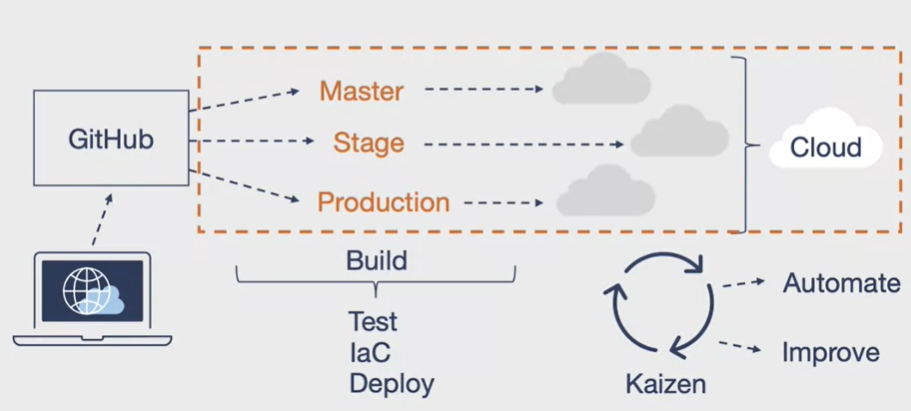
* 1. Workflow commands for GitHub Actions <https://docs.github.com/en/actions/using-workflows/workflow-commands-for-github-actions>
  2. Events that trigger workflows <https://docs.github.com/en/actions/using-workflows/events-that-trigger-workflows>
  3. Manually running a workflow <https://docs.github.com/en/actions/using-workflows/manually-running-a-workflow>
  4. Disabling and enabling a workflow <https://docs.github.com/en/actions/using-workflows/disabling-and-enabling-a-workflow>

**CI/CD Example**

CI/CD Video: <https://youtu.be/42UP1fxi2SY>



Source: Duke University, Cloud Computing Foundation



Source: Duke University, Cloud Computing Foundation