Git Configuration

Created By	
Stakeholders	
Status	
Type	Installation Guide
Created	@February 23, 2022 3:07 PM
Last Edited Time	@March 21, 2022 10:33 AM
Last Edited By	

Summary

Summary

This document is designed as a guide to configure a git repository on a remote server.

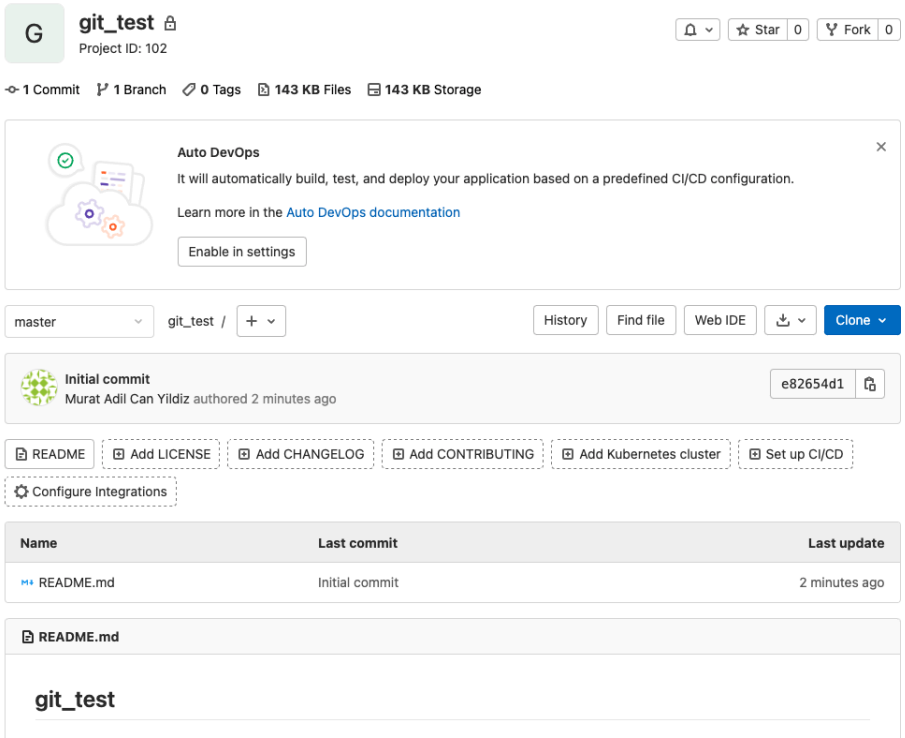
There are two core steps to achieve this:

1- Local machine - Git repository connection

2- Git - Remote Server connection

Local Machine - Git Repo Connection

1- Create an empty project under Gitlab:



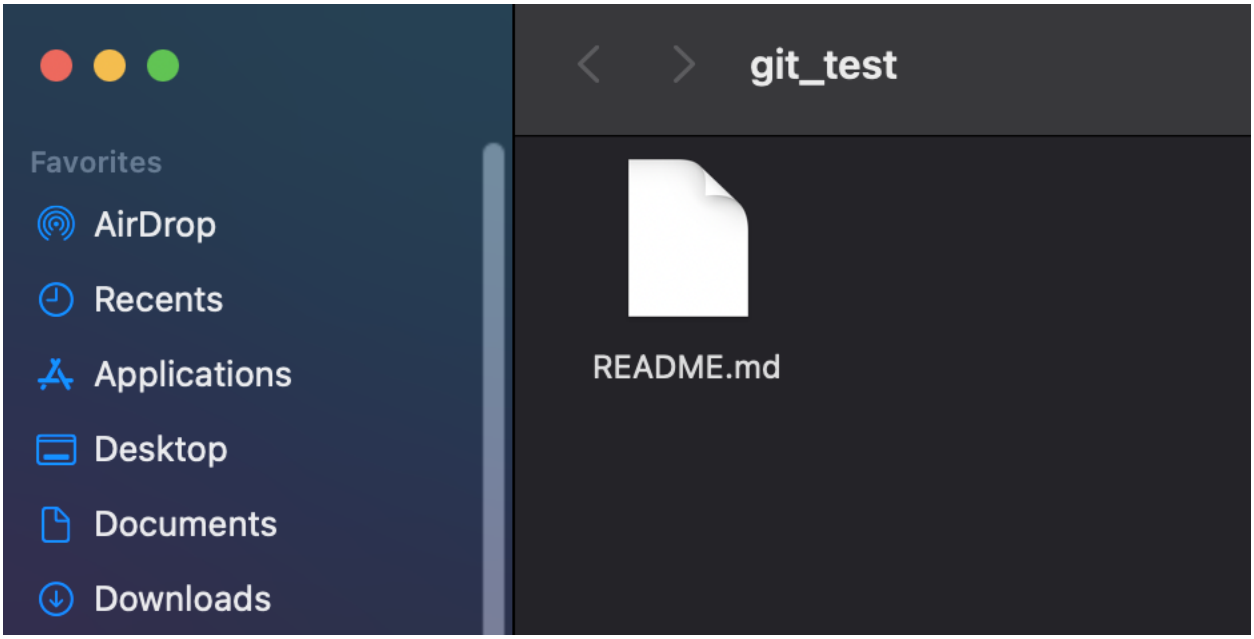
2- Open a new terminal. To set your global commit name and email address run the following commands:

```
## git config --global user.name "Your Name"
## git config --global user.email "youremail@yourdomain.com"

git config --global user.name "muratyildiz"
git config --global user.email "murat@ace.games"
```

4- Open a new terminal and navigate to the directory that you will store the repo files.

```
cd Desktop

# clones the repo to a new folder with the same name with git repo
git clone https://gitlab.ace.games/data/git_test.git
```

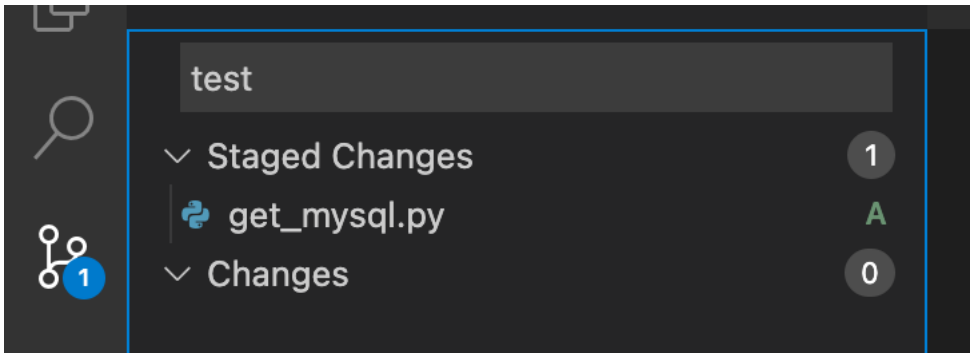
At this stage, there is only README file under the repo and the project folder seems as follows:



Open VS code and open the project folder:

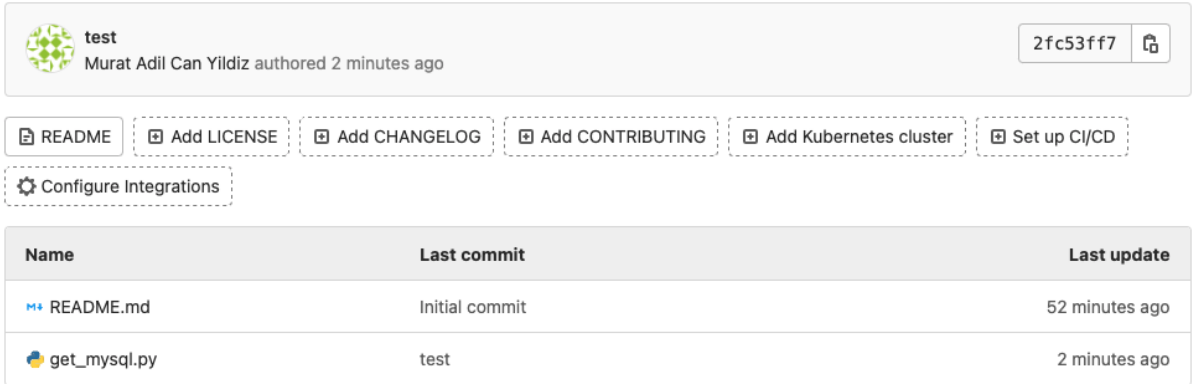
Add or create any file to push to original repo:

e.g: get_mysql.py



Then use following commands or VS code interface to push recent changes to the master branch at the origin (git server):

```
cd Desktop/git_test

git push origin master
```



Git - Remote Server Connection

To push any changes from the local machine to remote server, git - remote server connection side should be configured.

1- Create a user with sudo privileges.

After connecting the remote server through SSH keys, add a new git user with sudo privileges.

```
adduser git
## enter password, user information..

## add git user to sudo group
usermod -aG sudo git

sudo su git
cd

## create ..ssh directory and permissions: (U)ser / owner can read, write and execute.
mkdir .ssh && chmod 700 .ssh
```

2- create RSA keys and add them to .ssh file of the user.

```
## when using git user

cd .ssh/
ssh-keygen -t rsa

Enter file in which to save the key (/home/git/.ssh/id_rsa):
id_rsa_deploy
```

3- ssh-agent remembers and temporarily stores the passphrase in memory. Then as soon as you use the ssh command with the private key, ssh-agent will kick in to provide the passphrase for ssh session:

```
eval $(ssh-agent -s)

ssh-add id_rsa_deploy
```

4- When the user needs to access the remote servers frequently using SSH protocol, then the user will require to remember the IP addresses, usernames, different port numbers, and command-line options. But it is not an efficient way to do the tasks. This problem can be solved in multiple ways. The user can create the alias command of the bash for the remote connection that is easier to remember. Another solution is to create an SSH config file for each user to save the different SSH options for the different remote systems.

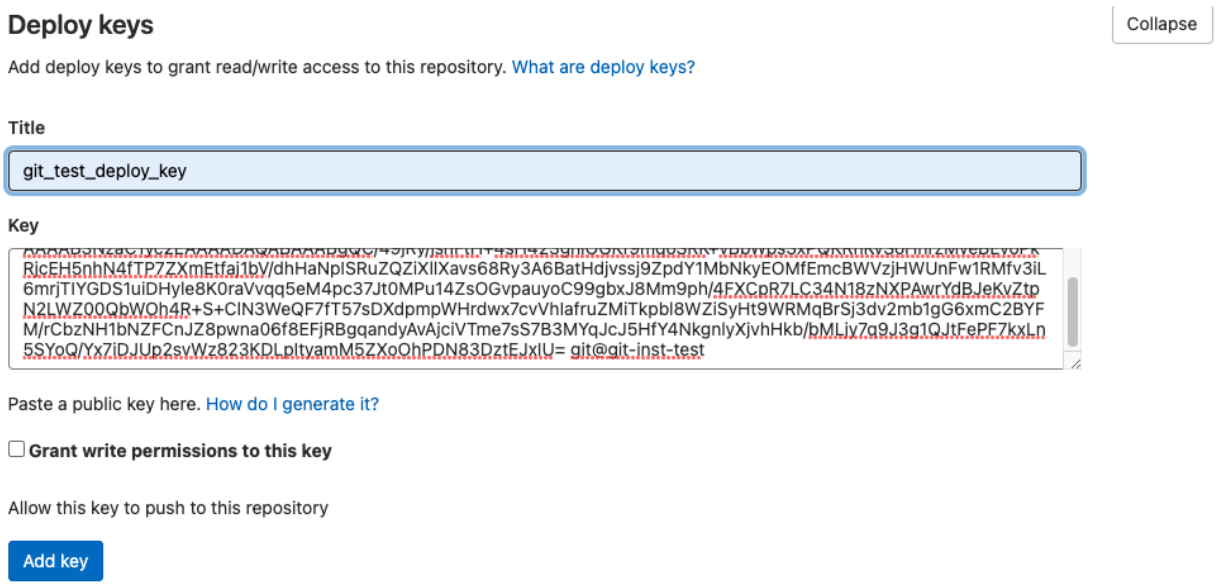
```
touch config
```

Enter the lines below and save the file:

https://man.openbsd.org/ssh_config.5

```
Host git_test
  ## used to define the hostname or IP address of your remote server.
  HostName gitunity.ace.games

  ## Specifies whether keys should be automatically added to a running ssh-agent.
  ## If this option is set to yes and a key is loaded from a file, the key and its passphrase are added to the agent with the default lifet
  AddKeysToAgent yes

  ## Specifies the order in which the client should try authentication methods
  PreferredAuthentications publickey

  ## Specifies a file from which the user's RSA authentication identity is read.
  IdentityFile ~/.ssh/id_rsa_deploy_key   ##
```

5- Go to the Gitlab and select the project. Under Settings/Repository expand Deploy_Keys section and enter the public key that is generated in the previous steps:,



6- Git, by default, looks for key thats name is id_rsa. Unless we create a new pair with the name id_rsa pipeline will return error. We create a new key pair with the id_rsa and repeat steps 3, 4, 5.

After both key pair is created, directory and config file should look like this:

```
git@git-inst-test:~/.ssh$ ls
authorized_keys  config  id_rsa  id_rsa.pub  id_rsa_deploy  id_rsa_deploy.pub  known_hosts
```

```
Host git_test_deploy
  HostName gitunity.ace.games
  AddKeysToAgent yes
  PreferredAuthentications publickey
  IdentityFile ~/.ssh/id_rsa_deploy

Host git_test
  HostName gitunity.ace.games
  AddKeysToAgent yes
  PreferredAuthentications publickey
  IdentityFile ~/.ssh/id_rsa
```

7- At this point, make sure that remote server don't have any network restriction when accesing git servers. (e.g. Cloudflare may prevent this connection even if remote server has all the permissions to use 22 port..)

After adding remote server IP address to the allowed IP's lists on cloudflare test the ssh connection:

```
ssh -T git@git_test
Welcome to GitLab, @murattica!

ssh -T git@git_test_deploy
Welcome to GitLab, @murattica!
```

8- Configure directories and folders that repo files will be stored.

Here, there are a couple of options to proceed:

- 1- When the remote server is used as a client (instead of host) one can choose to repeat above process under root user (By creating key pairs for root user). This way, user can access any directory.
- 2- One can proceed with the created git user and access the directories under home/git/. Under this directory, user have all the accesses to directories to clone repos.

  However, git user can only clone to directories which it has rwe permissions. For example, user can't clone any directory under home. To achieve this, targeted directory's permissions should be reset for the git user. (this document follows this option)

  *** Trying **sudo git clone** under any directory which git user has no access results with **Permission Denied** since using sudo means ssh client will try to use ssh keys under root/.ssh whereas we generated key pairs for git user.

  *** Copying key pairs from git user to root/.ssh directory also fails.

In this case, I created a sample directory under home. I wanted to create a directory under home which is accessible for specified users (in our case git user only)

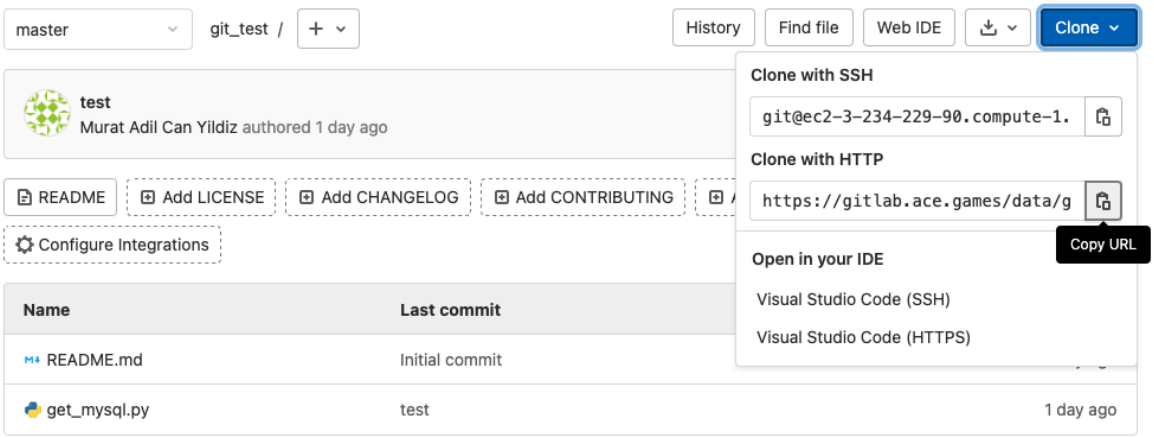
```
##with root user
cd
cd ../home/
mkdir git_repos
```

However, git user do not have access to this directory. To give necessary permissions use I used setafcl function from acl module.

```
## With root user
sudo apt install acl

## navigate to /home
setfacl -m u:git:rwx git_repos
```

Now, git user has all the accesses on git_repos directory.

9- Next step is to clone repo files to git_repos directory. To achieve this, copy URL on the Clone with HTTP option.




```
## git user
cd ~/../git_repos
git clone git@gitunity.ace.games:data/git_test.git
```

9- So far, we have all the necessary connections:

- local machine ↔ git repos
- git repos ↔ remote server

- First step to create a CICD pipeline is to create a gitlab runner.

GitLab Runner is an application that works with GitLab CI/CD to run jobs in a pipeline.

- Gitlab runner needs an executer to operate. There are multiple executer options:

e.g.: docker-ssh, docker+machine, parallels, shell, ssh, virtualbox, docker-ssh+machine, kubernetes, custom, docker

In this guide, we proceed with “docker” executer. (**GitLab Runner in a container)**

Before you install Docker Engine for the first time on a new host machine, you need to set up the Docker repository. Afterward, you can install and update Docker from the repository.

https://docs.docker.com/engine/install/ubuntu/

```
# Set up the repository

# Remove old versions
sudo apt-get remove docker docker-engine docker.io containerd runc

sudo apt-get update

##instlal necessary packages
sudo apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release

# Add Docker’s official GPG key:
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

# Use the following command to set up the stable repository.
# To add the nightly or test repository, add the word nightly or test (or both)
# after the word stable in the commands below

echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubunt
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

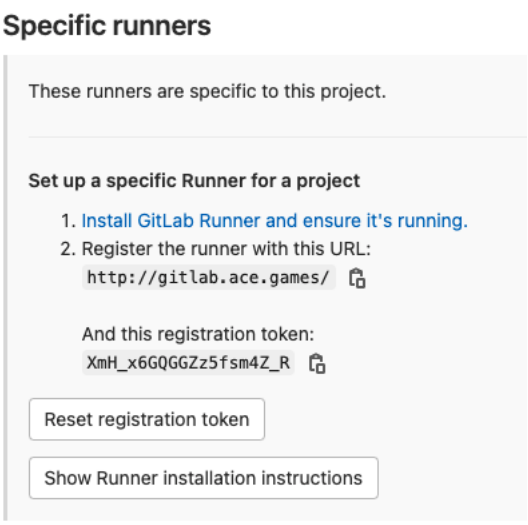
```
# Install Docker Engine

sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io

# Verify that Docker Engine is installed correctly by
# running the hello-world image.

sudo docker run hello-world
```

Now we can initiate gitlab runner and make necessary configurations:

```
# Registering gitlab runner
gitlab-runner register
```

After running gitlab-runner register, gitlab will ask some questions:

Token and URL should be taken under Settings → CI/CD → expand runners




```
Enter the GitLab instance URL (for example, https://gitlab.com/):

# http://gitlab.ace.games/

Enter the registration token:

XmH_x6GQGGZz5fsm4Z_R


Enter an executor: docker-ssh, docker+machine, parallels, shell, ssh, virtualbox, docker-ssh+machine, kubernetes, custom, docker:

docker

Enter the default Docker image (for example, ruby:2.7):

ruby:2.7
```

```
# Start the gitlab-runner
gitlab-runner start
```

10 - Now, we are ready to initiate a CI/CD:

When configuring .yml file, we will need some variables regarding VM IP, SSH user and private key of the user.

Under Settings → CICD → Expand Variables define the following variables:

Type	↑ Key	Value	Protected	Masked	Environments
Variable	GIT_TEST_SSH_PRIVATE_KE...	**********	✓	✕	All (default)
Variable	SSH_USER	**********	✓	✕	All (default)
Variable	VM_IPADDRESS	**********	✓	✕	All (default)

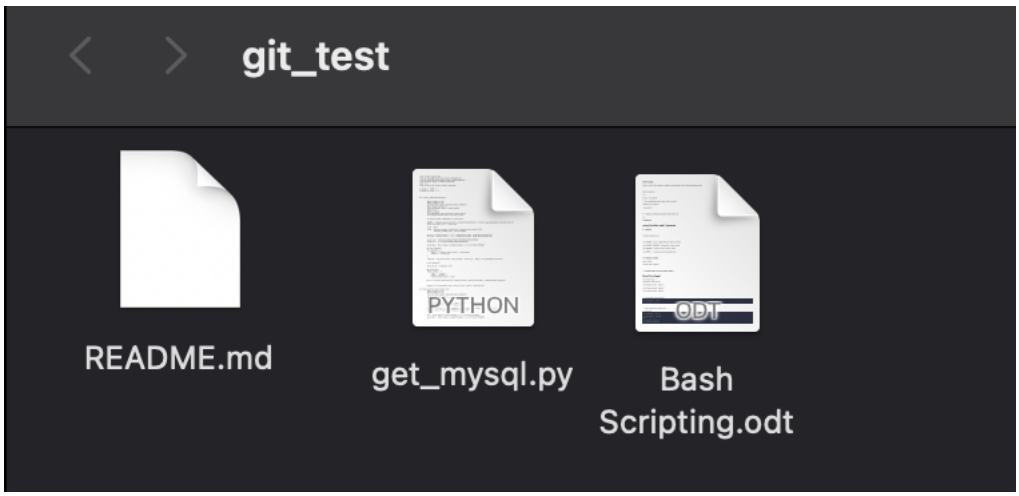
Add variable Reveal values

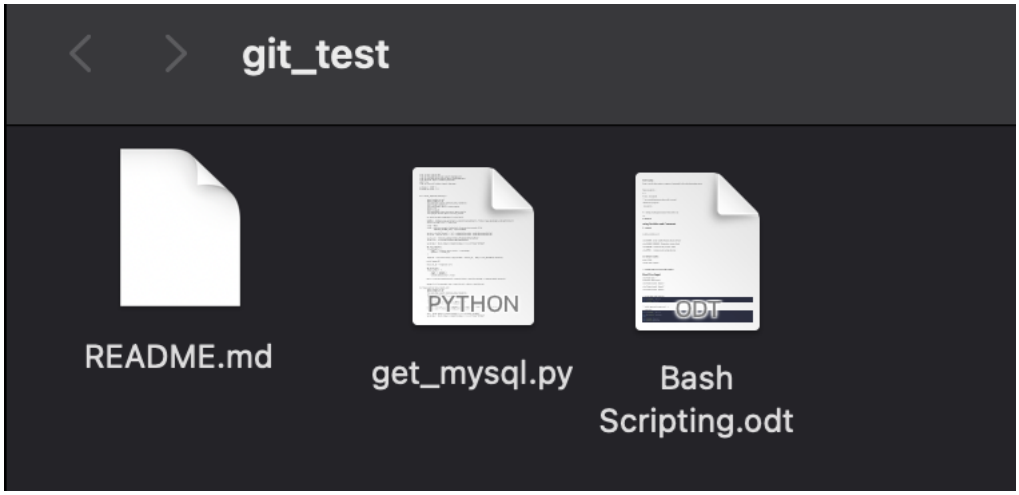
https://docs.gitlab.com/ee/ci/yaml/gitlab_ci_yaml.html

```
# Linux Distribution
image: alpine

# we don't have any test or build stages
stages:
  - deployment

# This job runs in the deploy stage.
deploy:
  stage: deployment
  before_script:     # Override a set of commands that are executed before job.
      - apk add openssh-client
      - eval $(ssh-agent -s)
      - echo "$GIT_TEST_SSH_PRIVATE_KEY_GL_DEPLOY" | tr -d '\r' | ssh-add -
      - mkdir -p ~/.ssh
      - chmod 700 ~/.ssh
      - ssh-keyscan $VM_IPADDRESS >> ~/.ssh/known_hosts
      - chmod 644 ~/.ssh/known_hosts
      - '[[ -f /.dockerenv ]] && echo -e "Host *\n\tStrictHostKeyChecking no\n\n" > ~/.ssh/config'
  script:        # Perform ssh connection
    - ssh -v $SSH_USER@$VM_IPADDRESS "cd ../home/data/fionas-farm && git checkout master && git pull origin master && exit"
  tags:
    - data_cicd
  only:
    - master
```

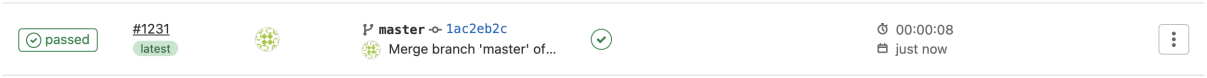
Push Test:

- Let's try to add a new file Bash Scripting.odt





Push Succesful:



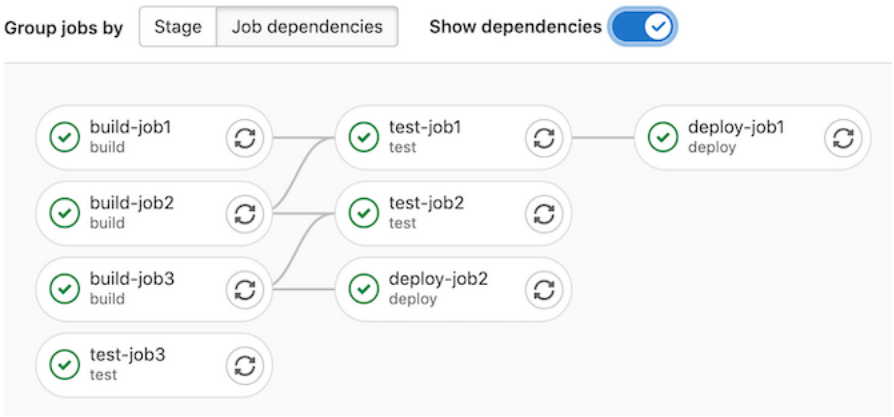
In the remote server, we can see that the committed file is present.

```
git@git-inst-test:/home/git_repos/git_test$ ls
'Bash Scripting.odt'   README.md   get_mysql.py
```

Additional Notes

1- In the yml file, stages and jobs are defined and associated.

e.g.:



We can also define dependency between jobs so that a job can trigger another job or it can be triggered manually.

2- Git → Runner

When commited changes are pushed , if auto devops is enabled, Git instantly creates the job pipeline based on the yml file under a TAG. Then it awaits a runner with the same TAG to collect and execute the pipeline.

Git : Creates the jobs on the pipeline
Runner: Collects and executes predefined jobs.