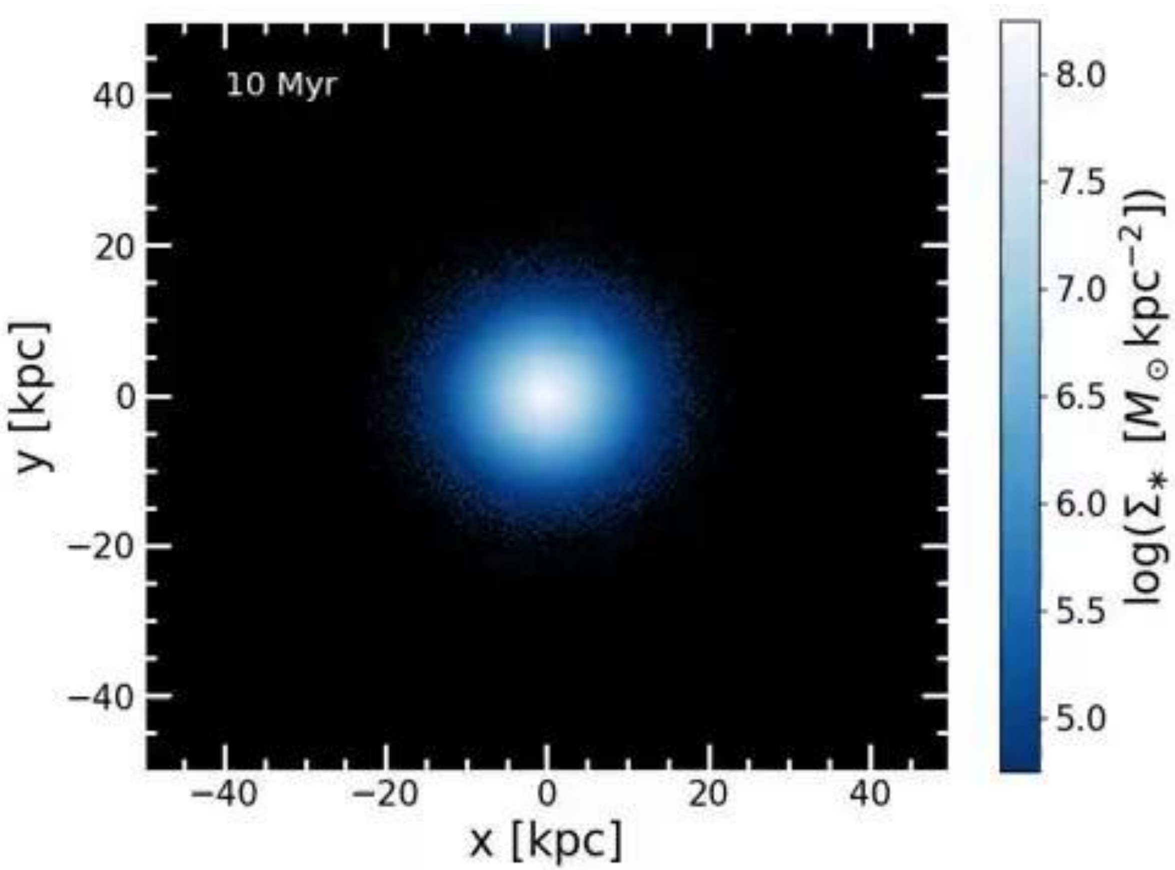
Version Control Using GitHub

Himansh Rathore
TIMESTEP Lecture Series
Oct 25 and Nov 8, 2024

What is github ?

- Have you all used drive before ? What does it allow you to do ?
- Github is a code developer platform:
  - Create
  - Store/Maintain
  - Manage
  - Share/Collaborate
- Github is based on a version control system called git
  - Keeps track of versions of files and the modifications therein
  - Keeps track of version conflicts
  - Allows multiple people to collaborate on one project
- Efforts to make research and code development:
  - Accessible
  - Reproducible
  - Collaborative

What drives scientific research ?

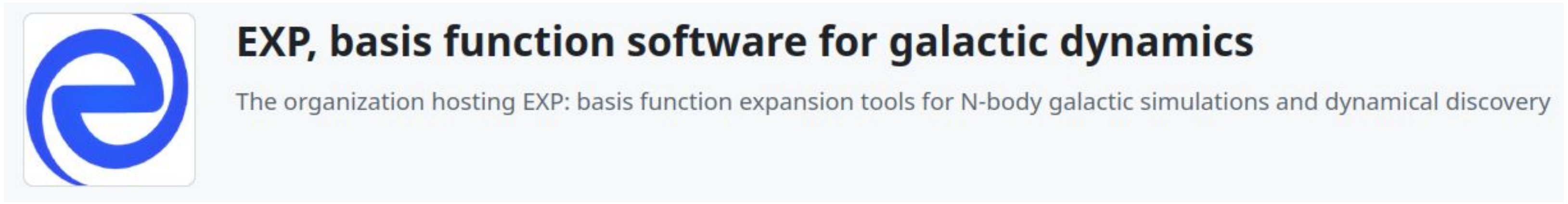


Availability and Accessibility to sophisticated tools and large collaborative projects !

- Gadget-4: publicly available on gitlab (https://gitlab.mpcdf.mpg.de/vrs/gadget4)

Gadget4

- EXP: publicly available on github (https://github.com/EXP-code)


EXP, basis function software for galactic dynamics

The organization hosting EXP: basis function expansion tools for N-body galactic simulations and dynamical discovery

Other examples

A small collaborative project:

-
  himanshrathore / pymargay
  (https://github.com/himanshrathore/pymargay)

Store and maintain your own private codes:

-
  himanshrathore / magclouds
  (https://github.com/himanshrathore/magclouds)

Educational materials:

-
  krittikaiitb / tutorials
  (https://github.com/krittikaiitb/tutorials)

Personal Website:

- Me: (https://himanshrathore.github.io/index.html)

Are there other version control languages ?

git

CVS

mercurial

SUBVERSION

Are there other hosting services ?

GitHub

Bitbucket

GitLab

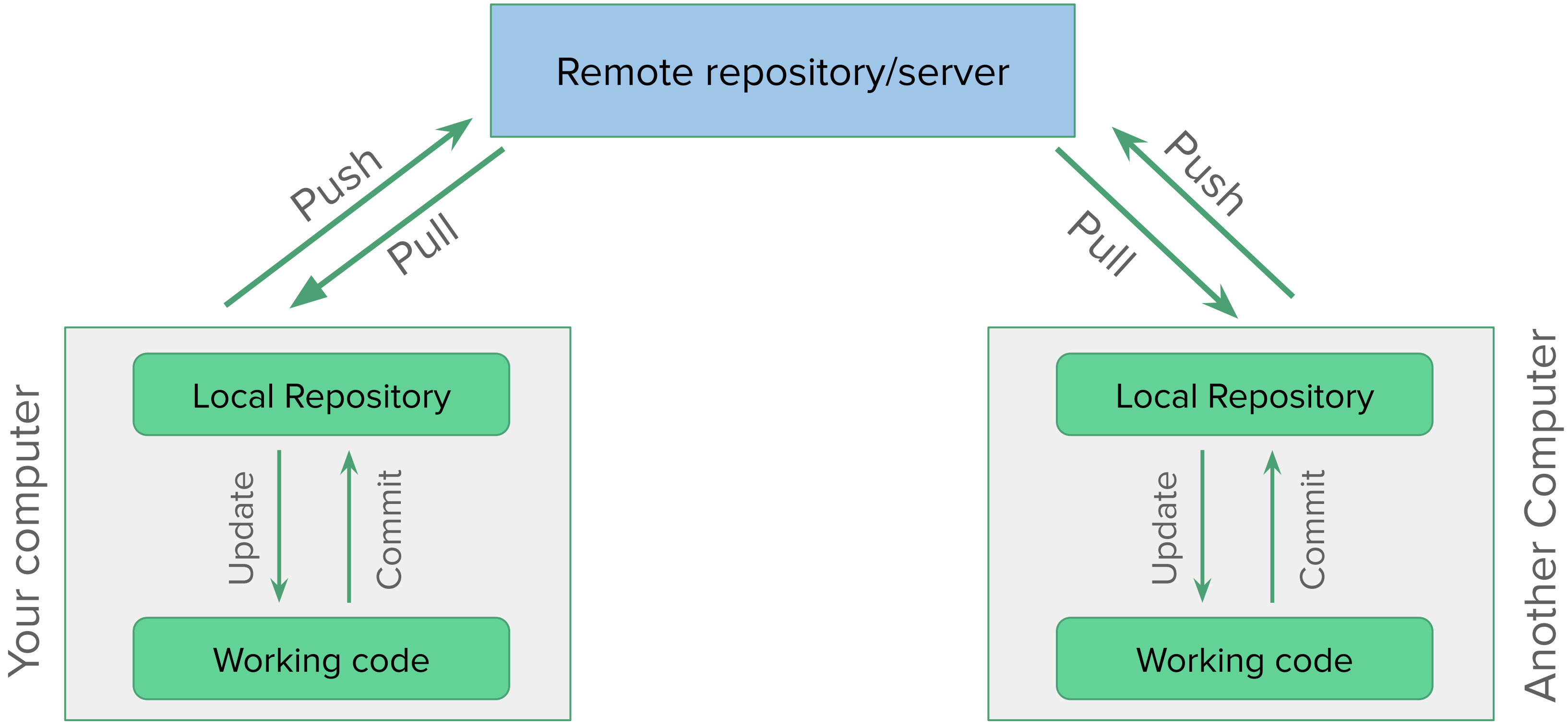
Our goals for the two lectures …

- Appreciate the importance of open-source software development and technologies like git and github which enable that
- Create an account on github
- Learn different components of github and the technical background
- Learn and play around with the github web interface
- Learn and play around with github using the terminal
- Learn how to manage individual projects as well as collaborative projects with github

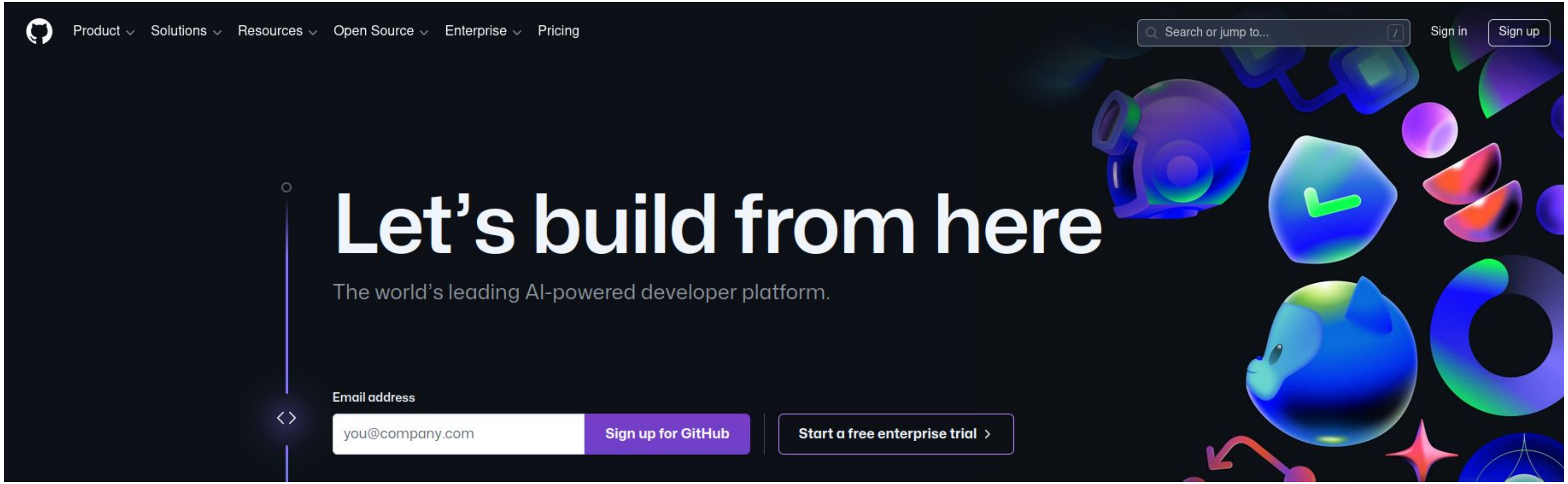
We will do lots of hands on exercises to demonstrate the above concepts. Help each other out !

Some basic technical background behind git

Repository - the directory that hosts your project



Slide adapted from Hayden Foote



The github web interface (github.com)

Lets create a github account

Were you able to attempt the homework ?

- You need a github username
- You need an email address associated with your account
- You need a github password (DO NOT SHARE WITH ANYONE)
- You need 2-factor authentication set up with the help of an external device (like mobile)

Anyone who has not been able to do this ?

Accessing a repository

- You need to access the subsequent slides and instructions from github itself !
- Go to the repository that I created (it is Public):
  - In the search bar on the top, search "himanshtimestep" and press enter. OR, directly use the following URL: https://github.com/himanshrathore/himanshtimestep
  - You will find the subsequent slides there
  - This is the only way you will have access to my slides !
- Clone my repository - which means downloading an exact copy to your computer
  - Go to "Code" (green button with drop down arrow)
  - Select "Download zip"
  - Extract the downloaded zip file to see the contents of my repo - including the slides

Creating a repository

- Log in to github and go to “Dashboard”
- Select the “New” button
- Give a short repository name (like <name>timestep)
- Give a short description
  - “TIMESTEP - learning how to use github”
- Choose repository access. Choose “Private”. Can change it later.
  - “Public” - anyone can see the repository and download stuff
  - “Private” - only you and the collaborators you explicitly invited can see the repo and download stuff
- Initialize the repository with a README file
  - Give a longer description to your project
  - Can write a documentation/installation instructions if you have code
- License - for our purposes choose NONE
- Click on “Create repository” !

Forking a repository

Forking - having an exact copy of another github repository to your own github account. Changes you make to your copy will not affect the original. Likewise, changes the original author makes will not affect your copy.

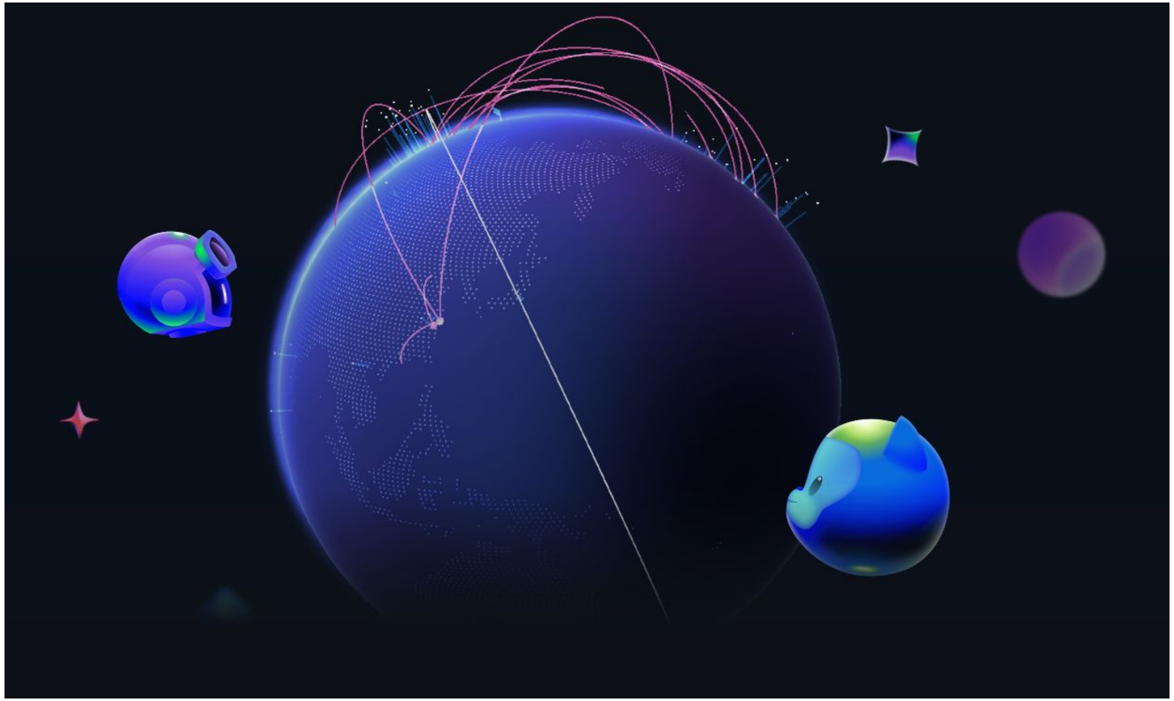
- When you fork a repo, always make sure you are not violating the LICENSE

Lets fork the “himanshtimestep” repository to your own github accounts.

- Find my repository, and click on “Fork” on the right hand side.
- Give the fork a name. Example “himanshtimestepfork”
- Give a short description. Example “Forking Himansh’s timestep repo”
- Then click on “Create fork”

Lets upload a file to your repository

- In your computer, open your favourite text editor
- Create a file
    - Example - ‘test.txt’
    - Write some content in it. Example - ‘This is a test upload.’
    - Save the file in your computer
- Go to your repository
- Click on the “+” sign to the left of the “Code” button
    - Click on “Upload files”
    - Select “choose your files”
    - Select the file from your computer
    - Add a short commit message. Example: “test file added”
    - Select “Commit changes”



Using github with the terminal

Why use github with a terminal ?

- Terminal offers a lot more control of your system and the tasks that you want the system to do
- Using git and github with the terminal offers you a lot more functionality and you feel the actual power of git !
- Doing stuff with the terminal is -
    - Faster
    - More secure
    - Sometimes the only way you can use a computer !
        - Specially true for supercomputers and datacenters
    - And honestly, just more cool !

Basic idea

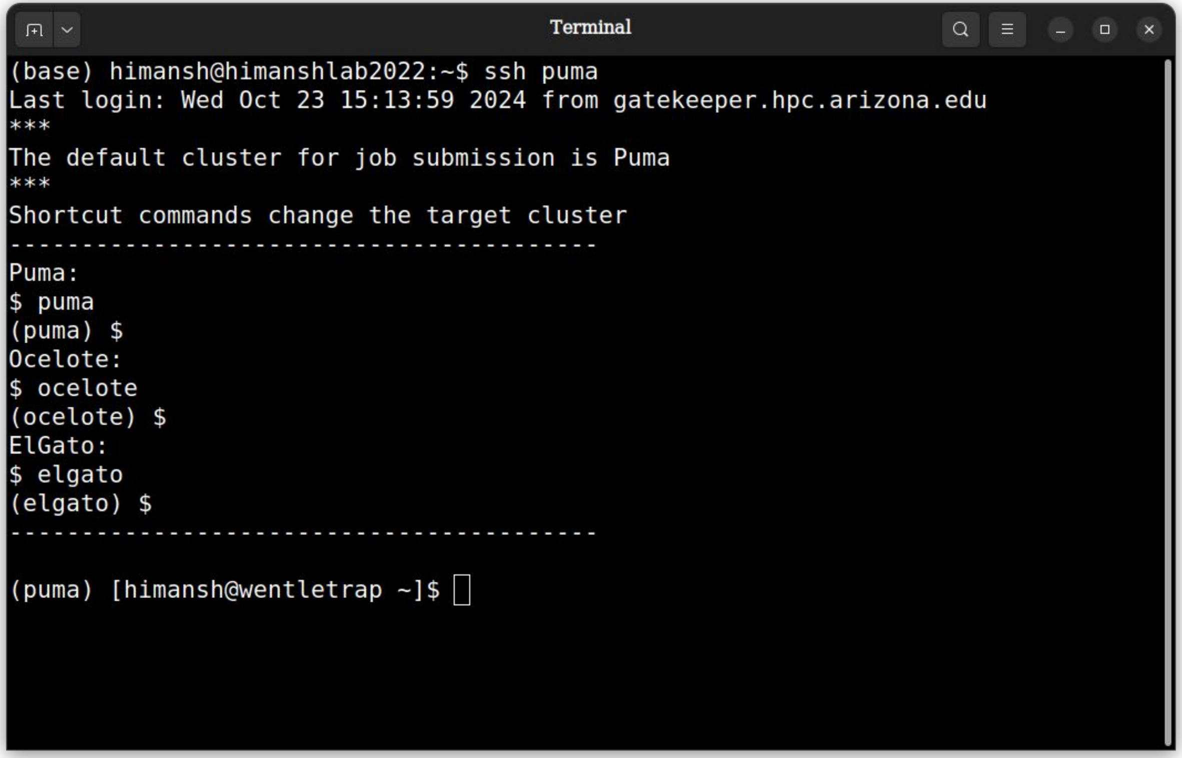
- Connect the local files in your computer to your remote server
- So that you can sync repositories between your computer and the github remote server
- You can modify and create code on your computer, and then automatically sync it to the remote server, while keeping track of versions
- Your collaborators can create or modify code on their computers and automatically sync it to the remote server, while keeping track of versions
- This requires a communication protocol between the remote server and your computer
    - We will use SSH for this protocol

What is SSH ?

- Stands for Secure SHell. Establishes a secure communication protocol between two systems. In this case - github remote server and your computer.
- Uses Public Key Encryption for security.
- You generate a pair of keys - a public key and a private key on your computer.
    - A key is basically a complex mathematical string, like a code
- You copy the public key to the remote server
- You keep the private key in your computer. You can copy it to other computers as well.
- When a computer tries to access the github remote server, github checks if the private key corresponding to the public key it has exists or not in the computer
    - If yes, communication permission is granted
    - If no, communication is blocked

Lets log-in to the University supercomputer

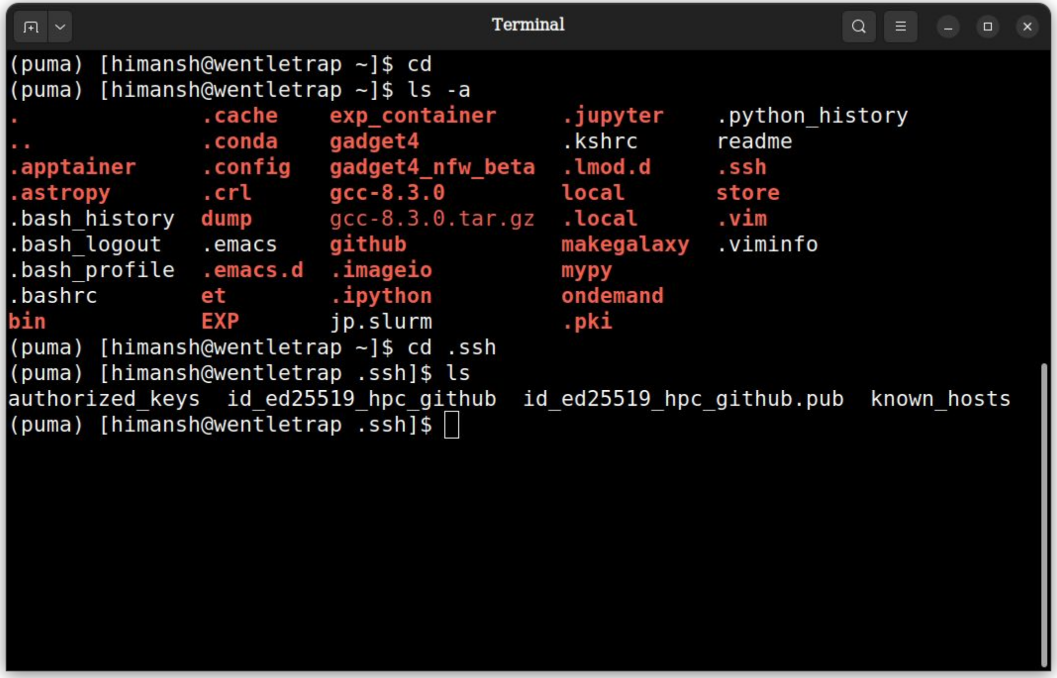
- Choose an OS that you have allocation to



```
(base) himansh@himanshlab2022:~$ ssh puma
Last login: Wed Oct 23 15:13:59 2024 from gatekeeper.hpc.arizona.edu
***
The default cluster for job submission is Puma
***
Shortcut commands change the target cluster
-----------------------------------------
Puma:
$ puma
(puma) $
Ocelote:
$ ocelote
(ocelote) $
ElGato:
$ elgato
(elgato) $
-----------------------------------------

(puma) [himansh@wentletrap ~]$ 
```

Lets create a public-private key pair

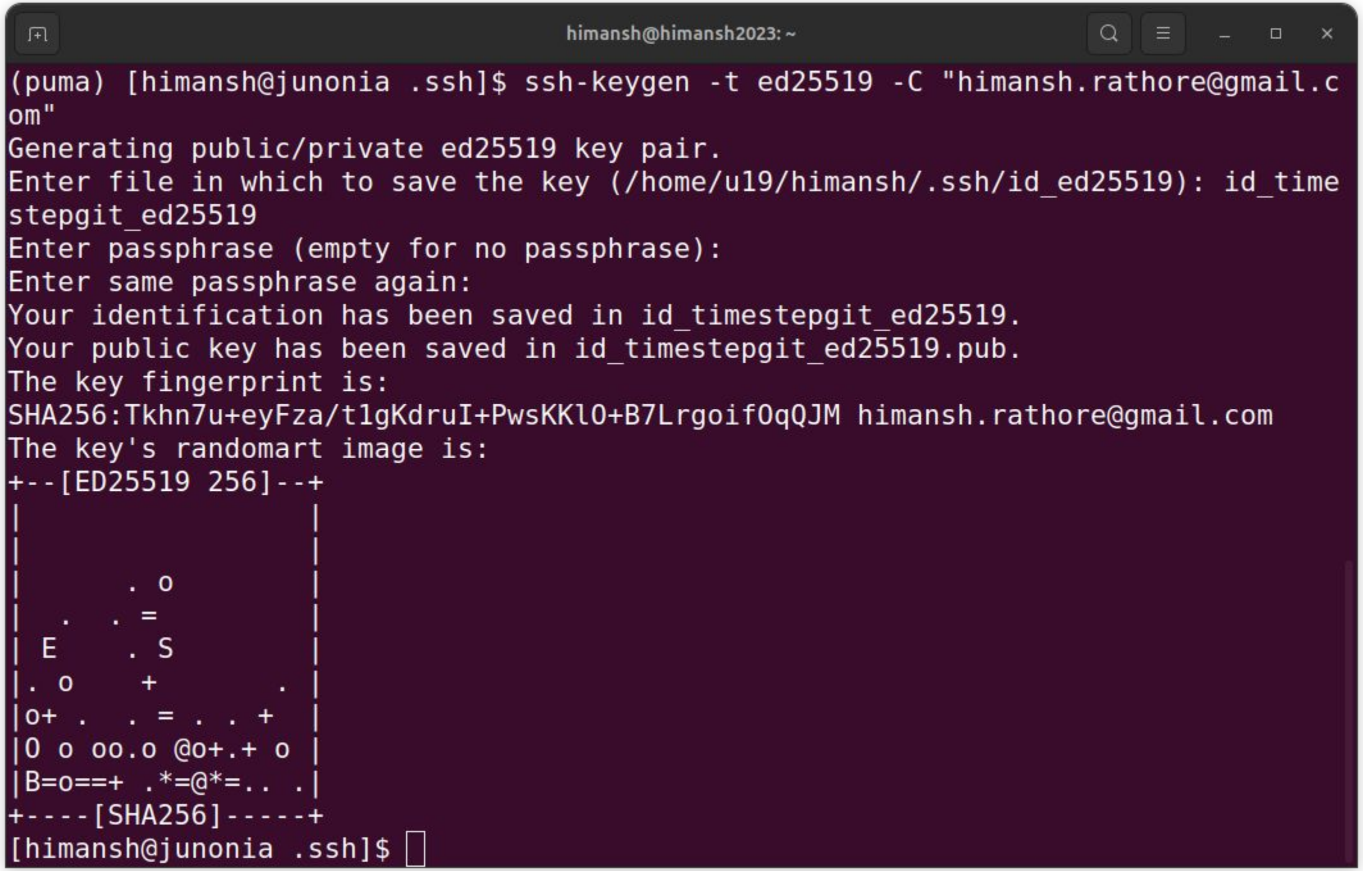
- Navigate to the home directory
  - $ cd
- See the hidden directories
  - $ ls -a
- Navigate to .ssh
  - $ cd .ssh
- Observe contents
  - $ ls


Terminal

```
(puma) [himansh@wentletrap ~]$ cd
(puma) [himansh@wentletrap ~]$ ls -a
.               .cache     exp_container      .jupyter     .python_history
..              .conda     gadget4            .kshrc       readme
.apptainer      .config    gadget4_nfw_beta   .lmod.d      .ssh
.astropy        .crl       gcc-8.3.0          local        store
.bash_history   dump       gcc-8.3.0.tar.gz   .local       .vim
.bash_logout    .emacs     github             makegalaxy   .viminfo
.bash_profile   .emacs.d   .imageio           mypy
.bashrc         et         .ipython           ondemand
bin             EXP        jp.slurm           .pki
(puma) [himansh@wentletrap ~]$ cd .ssh
(puma) [himansh@wentletrap .ssh]$ ls
authorized_keys  id_ed25519_hpc_github  id_ed25519_hpc_github.pub  known_hosts
(puma) [himansh@wentletrap .ssh]$ 
```

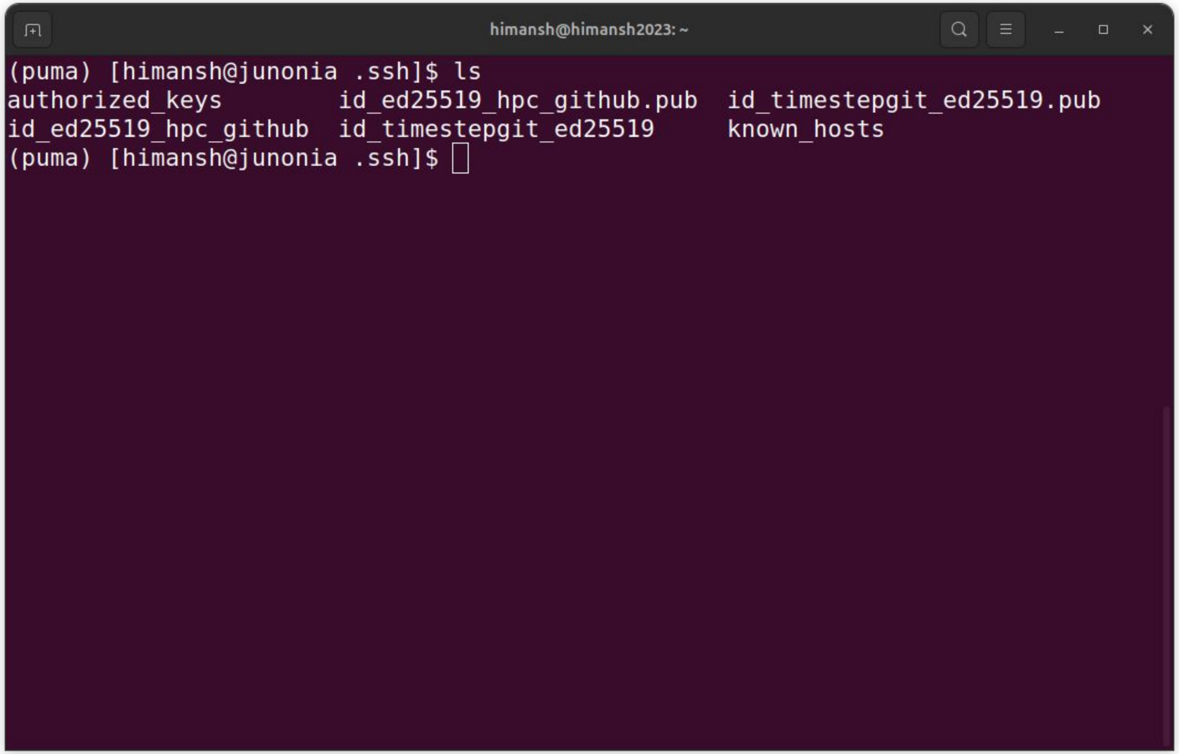
Lets create a public-private key pair

- $ ssh-keygen -t ed25519 -C “your_github_email@example.com”
  - ssh-keygen : ssh key generator
  - -t : selects the encryption algorithm
  - ed25519 : the encryption algorithm
  - -C : user authentication through email/password/2FA is required
- Enter a filename for the key. Example “id_timestepgit_ed25519”
- Enter a password - I would recommend to keep it the same as your github password
- Confirm the password
- You should see something like the next slide



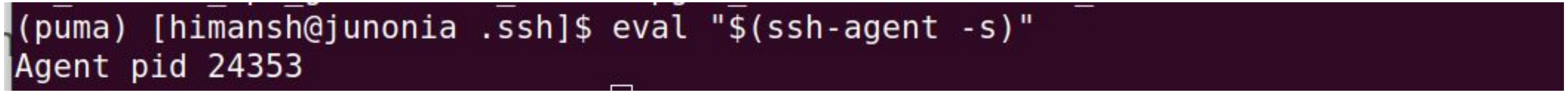
```
(puma) [himansh@junonia .ssh]$ ssh-keygen -t ed25519 -C "himansh.rathore@gmail.c
om"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/u19/himansh/.ssh/id_ed25519): id_time
stepgit_ed25519
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in id_timestepgit_ed25519.
Your public key has been saved in id_timestepgit_ed25519.pub.
The key fingerprint is:
SHA256:Tkhn7u+eyFza/t1gKdruI+PwsKKlO+B7LrgoifOqQJM himansh.rathore@gmail.com
The key's randomart image is:
+--[ED25519 256]--+
|                 |
|                 |
|      . o        |
|  .  . =         |
| E    . S        |
|. o    +       . |
|o+ .  . = . . +  |
|O o oo.o @o+.+ o |
|B=o==+ .*=@*=.. .|
+----[SHA256]-----+
[himansh@junonia .ssh]$
```

Public-Private key pair generated (.pub is the public) !


```
(puma) [himansh@junonia .ssh]$ ls
authorized_keys        id_ed25519_hpc_github.pub  id_timestepgit_ed25519.pub
id_ed25519_hpc_github  id_timestepgit_ed25519     known_hosts
(puma) [himansh@junonia .ssh]$ 
```

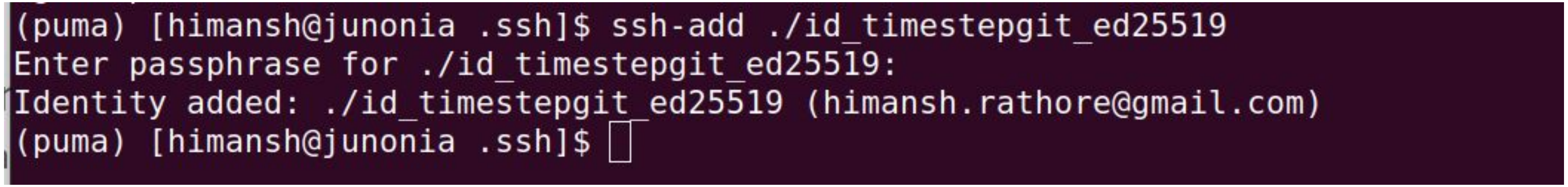
Add the key to the ssh-agent

- ssh-agent manages the keys. You need to tell it that you have created a new key. You may have to do this everytime you log in to the supercomputer.
- First, start the ssh-agent:
  - $ eval “$(ssh-agent -s)”



```
(puma) [himansh@junonia .ssh]$ eval "$(ssh-agent -s)"
Agent pid 24353
```

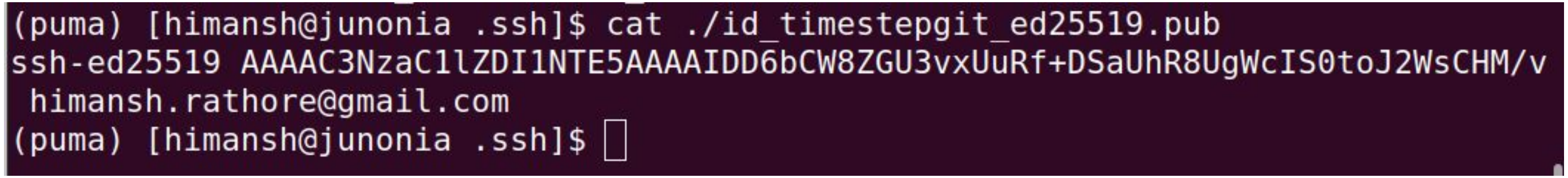
- Add the ssh-key to the agent (it will ask for your password):
  - $ ssh-add ./id_timestepgit_ed25519



```
(puma) [himansh@junonia .ssh]$ ssh-add ./id_timestepgit_ed25519
Enter passphrase for ./id_timestepgit_ed25519:
Identity added: ./id_timestepgit_ed25519 (himansh.rathore@gmail.com)
(puma) [himansh@junonia .ssh]$
```

Lets add the public key to the github remote server

- Copy the contents of the public key
  - $ cat ./id_timestepgit_ed25519.pub
  - Then manually copy

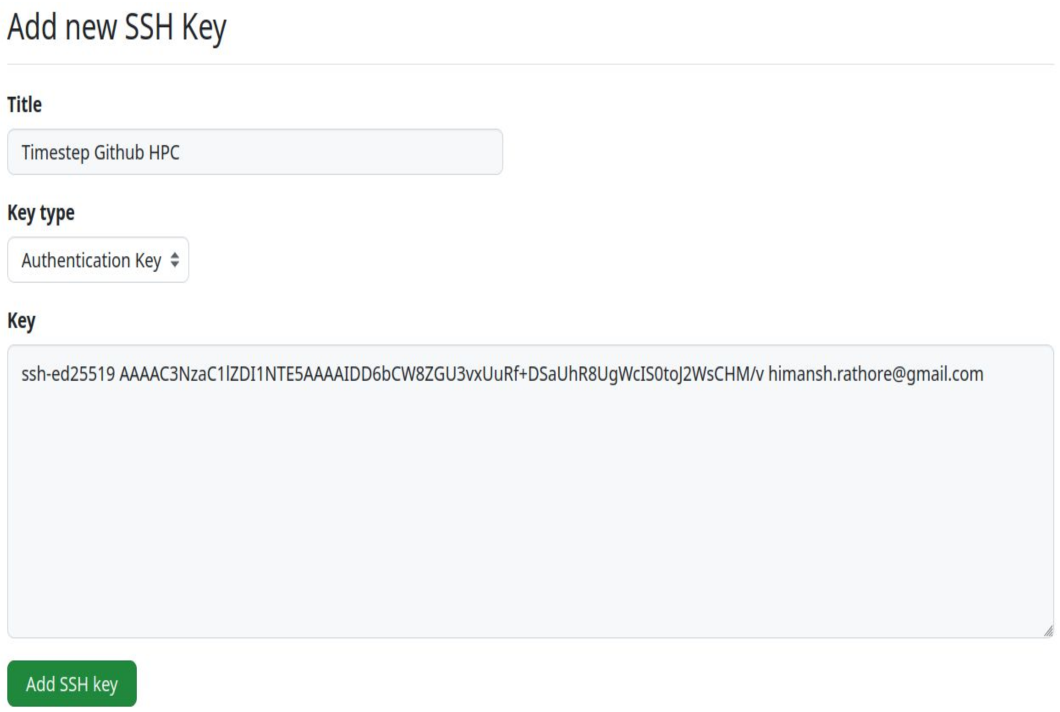


```
(puma) [himansh@junonia .ssh]$ cat ./id_timestepgit_ed25519.pub
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIDD6bCW8ZGU3vxUuRf+DSaUhR8UgWcIS0toJ2WsCHM/v
 himansh.rathore@gmail.com
(puma) [himansh@junonia .ssh]$ 
```

- Log-in to your github account on the web
- On the right hand side, select your profile icon and go to “Settings”
- In the “Access” tab on the left, go to “SSH and GPG keys”
- Click on “New SSH key”

Lets add the public key to the github remote server

- Put in a title - Example: “Timestep Github HPC”
- Let the key type be “Authentication Key”
- Copy the contents of your key into the textbox
- Click on “Add SSH key”
- Verify 2FA
- You will see your newly added key in the list !
- You are ready !



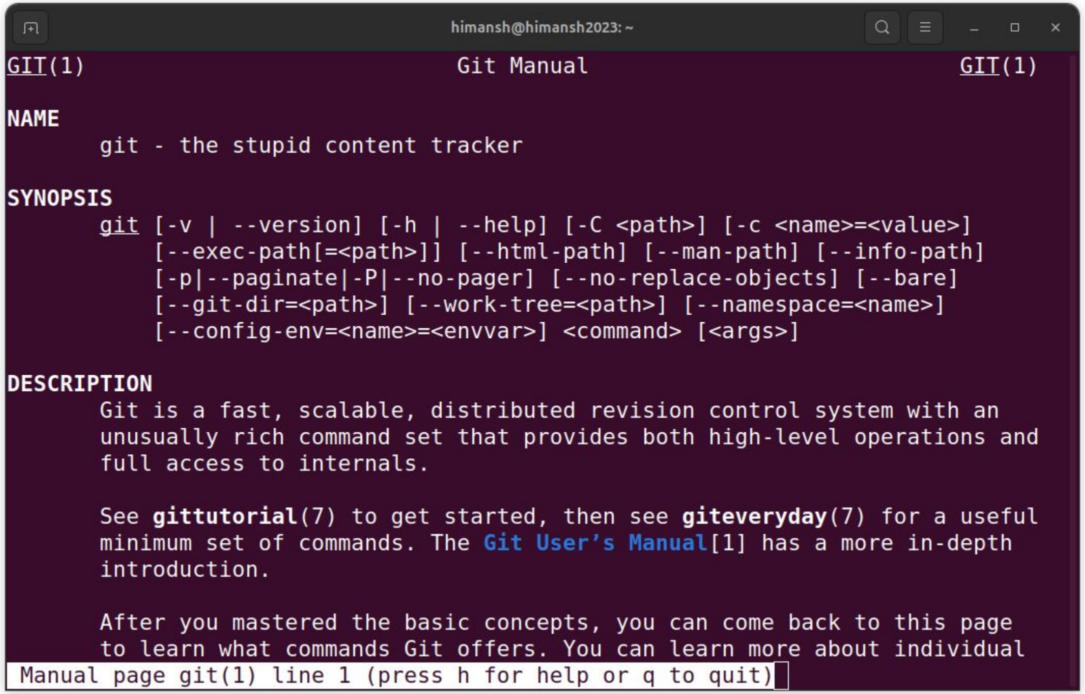
Add new SSH Key

Title: Timestep Github HPC

Key type: Authentication Key

Key:

```
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIDD6bCW8ZGU3vxUuRf+DSaUhR8UgWcIS0toJ2WsCHM/v himansh.rathore@gmail.com
```

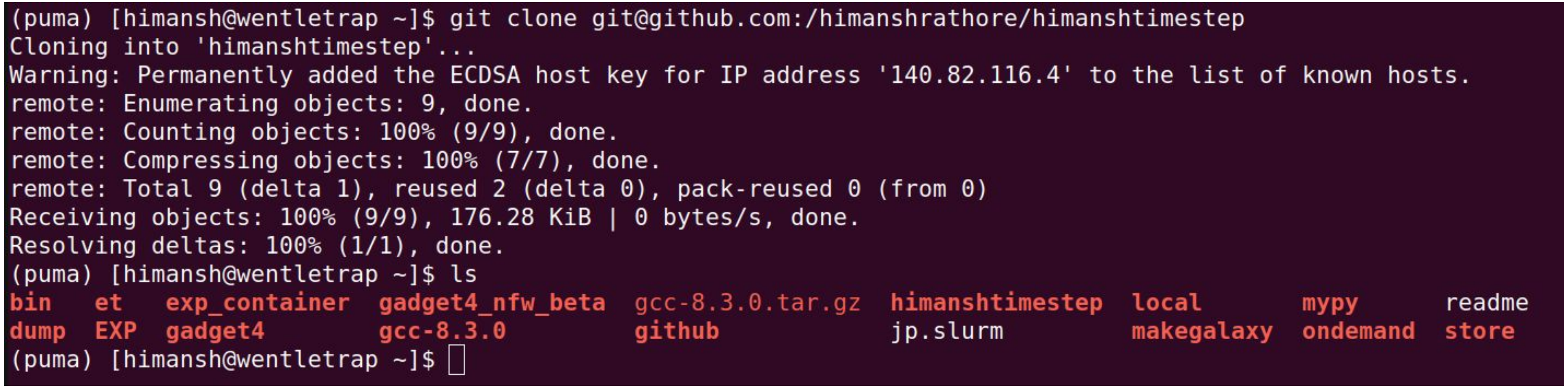


```
GIT(1)                           Git Manual                           GIT(1)

NAME
       git - the stupid content tracker

SYNOPSIS
       git [-v | --version] [-h | --help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p|--paginate|-P|--no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           [--config-env=<name>=<envvar>] <command> [<args>]

DESCRIPTION
       Git is a fast, scalable, distributed revision control system with an
       unusually rich command set that provides both high-level operations and
       full access to internals.

       See gittutorial(7) to get started, then see giteveryday(7) for a useful
       minimum set of commands. The Git User's Manual[1] has a more in-depth
       introduction.

       After you mastered the basic concepts, you can come back to this page
       to learn what commands Git offers. You can learn more about individual
 Manual page git(1) line 1 (press h for help or q to quit)
```

Writing git scripts

Cloning a repository using git

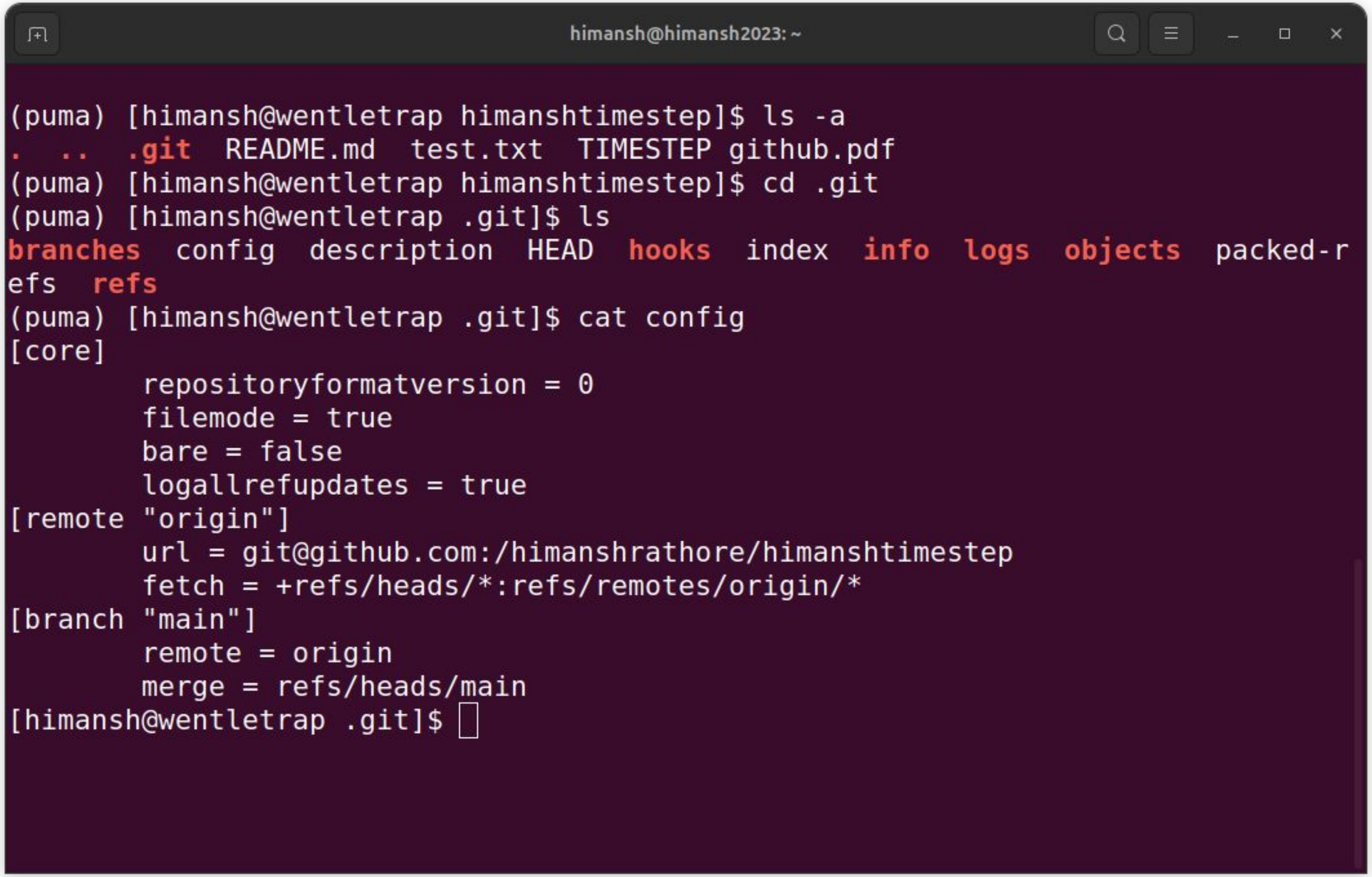
- Go back to the supercomputer terminal and navigate to the home directory
  - $ cd
- Now, on the web interface, go to the repository you created
  - Click the green “Code” button and access the drop down list
  - Choose the SSH option (HTTPS is also possible but I prefer SSH)
  - Copy the contents of the textbox
    - “git@github.com:himanshrathore/himanshtimestep.git”
- In the terminal, run the following:
  - $ git clone git@github.com:himanshrathore/himanshtimestep.git
- You should have a clone of your repository as a new directory. Check using
  - $ ls



```
(puma) [himansh@wentletrap ~]$ git clone git@github.com:/himanshrathore/himanshtimestep
Cloning into 'himanshtimestep'...
Warning: Permanently added the ECDSA host key for IP address '140.82.116.4' to the list of known hosts.
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 9 (delta 1), reused 2 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (9/9), 176.28 KiB | 0 bytes/s, done.
Resolving deltas: 100% (1/1), done.
(puma) [himansh@wentletrap ~]$ ls
bin   et   exp_container  gadget4_nfw_beta  gcc-8.3.0.tar.gz  himanshtimestep  local       mypy      readme
dump  EXP  gadget4        gcc-8.3.0         github            jp.slurm         makegalaxy  ondemand  store
(puma) [himansh@wentletrap ~]$ 
```

What is special about this cloned directory ?

- What is the difference between this cloned directory and any regular directory that already exists on the computer ?
- Access the directory
  - $ cd himanshtimestep
  - $ ls
- You will see the contents of your repository as it is
- But, now access the hidden files
  - $ ls -a
- What do you see ? Access the .git directory
  - $ cd .git
  - $ ls
- Thats the magic of github ! This directory is somehow connected to your repository on the web server. The connection is accessed using git !
  - $ cat config


himansh@himansh2023: ~

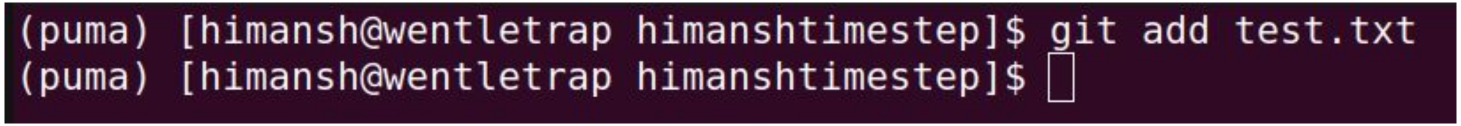
```
(puma) [himansh@wentletrap himanshtimestep]$ ls -a
.  ..  .git  README.md  test.txt  TIMESTEP github.pdf
(puma) [himansh@wentletrap himanshtimestep]$ cd .git
(puma) [himansh@wentletrap .git]$ ls
branches  config  description  HEAD  hooks  index  info  logs  objects  packed-r
efs  refs
(puma) [himansh@wentletrap .git]$ cat config
[core]
        repositoryformatversion = 0
        filemode = true
        bare = false
        logallrefupdates = true
[remote "origin"]
        url = git@github.com:/himanshrathore/himanshtimestep
        fetch = +refs/heads/*:refs/remotes/origin/*
[branch "main"]
        remote = origin
        merge = refs/heads/main
[himansh@wentletrap .git]$
```

Lets make some edits to the files

- Go one level up in directory
  - $ cd ..
- Open the text.txt file with a text editor
  - $ vi text.txt
- Modify the file. Example - insert a line
  - Press “i” to start the insert mode
  - “This is my first modification”
  - Press “esc” to get out of insert mode and then “:wq” to write and quit
- Now, how do we sync this to the github web server/remote repository ?
  - First, we need to tell git that we have made some modification to a file

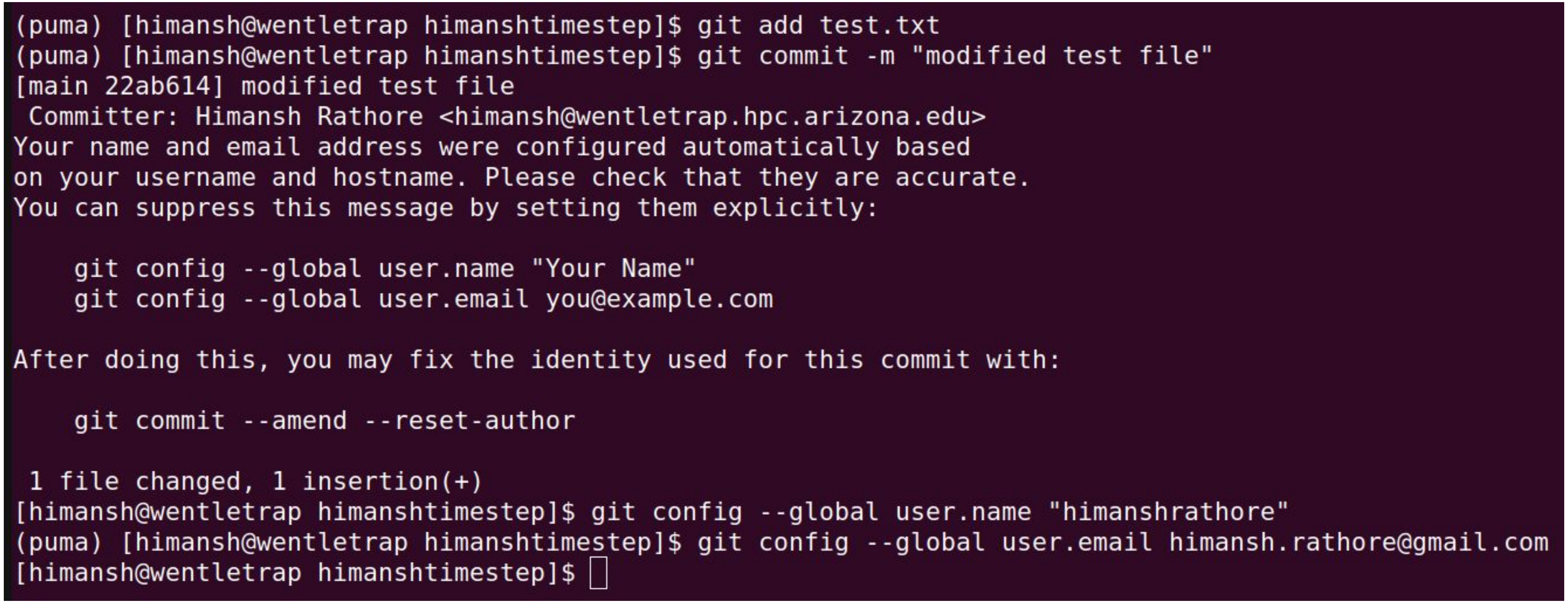
Using git add and git commit

- Use git add for this
  - $ git add test.txt



```
(puma) [himansh@wentletrap himanshtimestep]$ git add test.txt
(puma) [himansh@wentletrap himanshtimestep]$ 
```

- Note: if you modified multiple files, or created some additional files, you can use '.' to add all the files or all the modifications to git at once
  - $ git add .
- Lets commit our changes. For this, do:
  - $ git commit -m "modified test file"
  - "-m" is for a message - the commit message should be short and informative
- You may get a message from git asking you to set your username and email explicitly. Follow the steps. This is because it is your first commit.



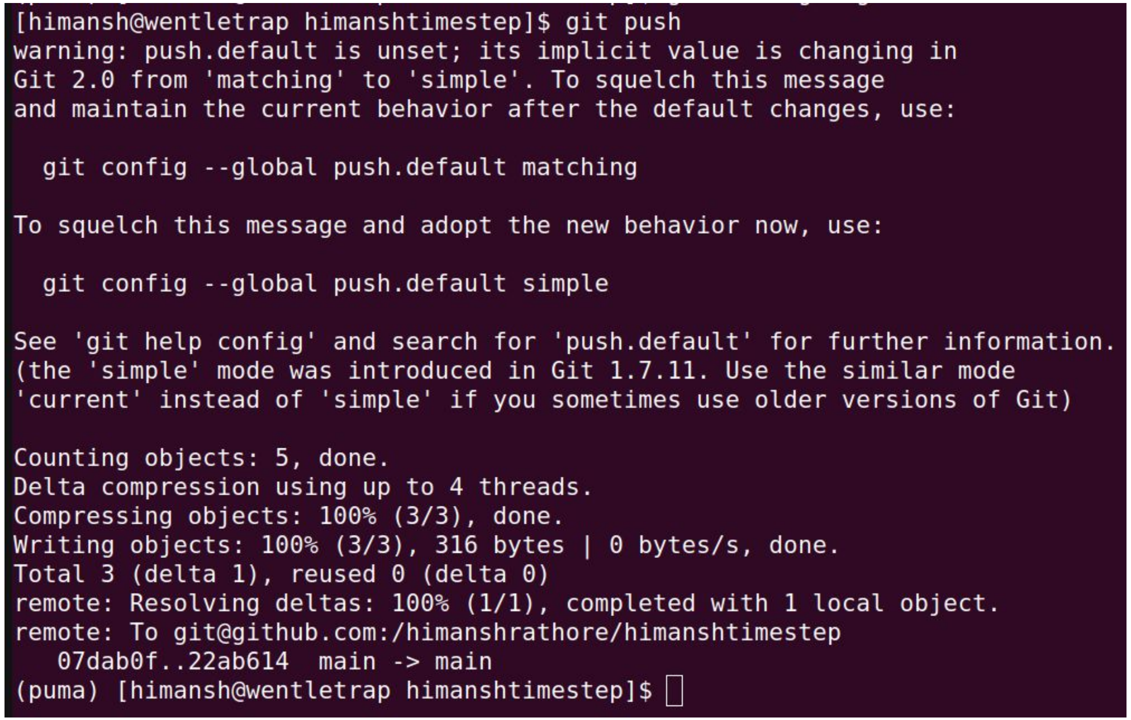
```
(puma) [himansh@wentletrap himanshtimestep]$ git add test.txt
(puma) [himansh@wentletrap himanshtimestep]$ git commit -m "modified test file"
[main 22ab614] modified test file
 Committer: Himansh Rathore <himansh@wentletrap.hpc.arizona.edu>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

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

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 1 file changed, 1 insertion(+)
[himansh@wentletrap himanshtimestep]$ git config --global user.name "himanshrathore"
(puma) [himansh@wentletrap himanshtimestep]$ git config --global user.email himansh.rathore@gmail.com
[himansh@wentletrap himanshtimestep]$ 
```

Using git push

Push the changes to your remote repository by.

$ git push

Now go to the web interface and check the file (you may need to refresh the browser page)!



```
[himansh@wentletrap himanshtimestep]$ git push
warning: push.default is unset; its implicit value is changing in
Git 2.0 from 'matching' to 'simple'. To squelch this message
and maintain the current behavior after the default changes, use:

  git config --global push.default matching

To squelch this message and adopt the new behavior now, use:

  git config --global push.default simple

See 'git help config' and search for 'push.default' for further information.
(the 'simple' mode was introduced in Git 1.7.11. Use the similar mode
'current' instead of 'simple' if you sometimes use older versions of Git)

Counting objects: 5, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 316 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote: To git@github.com:/himanshrathore/himanshtimestep
   07dab0f..22ab614  main -> main
(puma) [himansh@wentletrap himanshtimestep]$ 
```

How to access the previous version of a file ?

- Suppose for some reason you realized you need a previous version of a file
  - Its going to happen way too often in research !
- In the main page of the repository, click on the file
- Click on the “History” button at the right hand side
- You will see the history of this file with the corresponding commit messages along with date/time
  - Thats why the commit messages are important !
- In each history tab, on the left you have a variety of options
  - You can view the file at a specific instance of the past
  - You can view the entire repository at that instance
- This is a very powerful feature of github and version control !