Productivity at the KEKCC

- **Perquisite**
  - OS:- preferred Ubuntu (Linux based) or MacOS (Unix-based)
  - A desy account
  - Access to the KEKCC work server
  - The SSH key generated
  - Optional
    - A grid certificate
- **Basic Linux commands**
  - Navigation and Information
    - **pwd**: Print the working directory (shows where you are).
    - **ls**: List the files in a directory.
      - ls -l: Detailed listing with file permissions, size, and modification date.
      - ls -a: Show hidden files (files starting with .).
    - **cd [directory]**: Change directory to the specified location.
      - cd .. : Move up one level.
      - cd ~ : Go to the home directory.
    - **man [command]**: Show the manual for a command (e.g., man ls).
  - File and Directory Management
    - **mkdir [directory]**: Create a new directory.
    - **rmdir [directory]**: Remove an empty directory.
    - **rm [file]**: Remove a file.
      - **rm -r [directory]**: Remove a directory and its contents.
      - **rm -i [file]**: Interactive prompt before removing each file.
    - **cp [path to the source] [path to the destination]**: Copy files or directories.
      - **cp -r [path to the source] [path to the destination]**: Copy directories recursively.
    - **mv [source] [destination]**: Move or rename files and directories.
    - **touch [file]**: Create an empty file or update the modification date of an existing file.
  - File Viewing and Editing
    - **cat [file]**: Display the contents of a file.
    - **more [file]**: View file content page by page (press spacebar to scroll).
    - **less [file]**: Similar to more, but allows backward scrolling with arrow keys.
    - **nano [file]**: Open a simple, user-friendly terminal-based text editor.
    - **vim [file]**: Open a powerful, feature-rich terminal-based text editor (for advanced users).
  - Networking and Connectivity
    - **ssh [user]@[hostname]**: Log in to a remote machine using SSH.
    - **scp [file] [user]@[hostname]:[path]**: Securely copy files between local and remote machines.
    - **ping [hostname]**: Test network connectivity to a host (e.g., ping google.com).
    - **ifconfig or ip a**: Show network interfaces and IP addresses.
  - Permissions and Ownership
    - **chmod [permissions] [file]**: Change the permissions of a file (e.g., chmod 755 file.sh)
  - Process and System Information
    - **top**: Display the running processes and system resource usage.
    - **ps**: List current running processes.

    - **ps aux**: Detailed list of all processes.
  - **kill [PID]**: Terminate a process by its Process ID (PID).
    - **kill -9 [PID]**: Forcefully kill a process.
  - **df -h**: Show disk space usage in human-readable format.
  - **du -sh [directory]**: Show the size of a directory.

▼ Search and Find

  - **grep [pattern] [file]**: Search for a pattern within a file.
    - **grep -r [pattern] [directory]**: Search recursively in a directory.
  - **find [directory] -name [filename]**: Find files or directories by name.
  - **locate [filename]**: Quickly locate files by searching an index (requires updatedb to update).

▼ File Compression and Archiving

  - **tar -czvf [archive.tar.gz] [directory]**: Compress and archive a directory.
  - **tar -xzvf [archive.tar.gz]**: Extract a compressed archive.
  - **zip [archive.zip] [file]**: Compress files into a zip archive.
  - **unzip [archive.zip]**: Extract a zip archive.

▼ Others

  - **alias [name]='[command]'**: Create a shortcut for a command.
    - e.g., alias ll='ls -l' to quickly run a detailed list of files.
  - **sudo [command]**: Run a command with superuser (root) privileges.
  - **echo [text] > [file]**: Write text to a file (overwrites existing content).
  - **echo [text] >> [file]**: Append text to a file.

▼ **Basics of SSH (Secure Shell)**

SSH (Secure Shell) is a cryptographic network protocol used to securely connect to a remote system, often over an unsecured network.

SSH is commonly used for:

- Securely accessing remote servers.
- Managing servers and transferring files between computers.
- Executing commands on a remote machine.

▼ **How SSH Works**

When you initiate an SSH connection, the following process occurs:

- **Client-Server Model**: You use an SSH client (e.g., the ssh command in a terminal) to connect to an SSH server (which is running on the remote machine).
- **Encryption**: The data exchanged between the client and server is encrypted to ensure security.
- **Authentication**: The server verifies your identity, typically through a password or SSH key (a more secure method).
- **Secure Communication**: Once authentication is complete, all communication between your client and the remote server is securely encrypted.

The basic SSH command to log into a remote system is:

```
ssh username@hostname
```

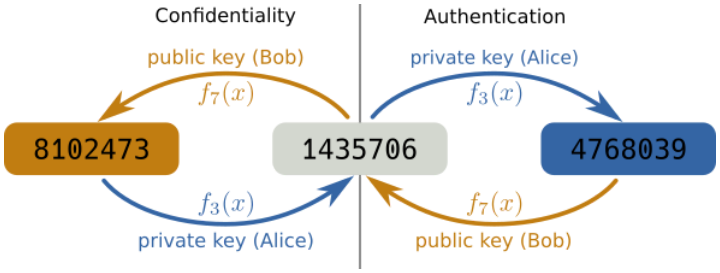
- username: The username on the remote system (your desy username).
- hostname: The IP address or domain name of the remote system.
  - Workserver - login.cc.kek.jp
  - Access Server- sshcc1.kek.jp

▼ **SSH Key Authentication**

While you can use passwords to authenticate via SSH, SSH key authentication is a more secure method. It relies on a pair of cryptographic keys:

- **Public key**: Stored on the remote server.
- **Private key**: Stored on your local machine and kept secure. This key is never shared.

When you connect via SSH, the server uses the public key to encrypt a message, and only the corresponding private key can decrypt it. This ensures secure authentication without needing a password.



▼ Generating SSH Keys

To set up SSH key-based authentication, you need to generate an SSH key pair (a public key and a private key). This is typically done using the ssh-keygen command.

Here's how to generate an SSH key pair:

1. Open your terminal and type:

```
ssh-keygen -C "Gaurav Laptop"
```

- -C "<some name >": Adds a comment to the key (often your email), which can help identify the key later.
  - The "Gaurav Laptop" can be replaced with any other name as per your convenience.

2. Save the key
   - After running the command, you will be prompted to save the key:

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

Press Enter to save it to the default location (~/.ssh/id_rsa) or specify a different location.

- It will generate two files:
- **id_rsa**: Your private key (this should be kept secure and not shared).
- **id_rsa.pub**: Your public key (this is copied to the server you want to access).

3. Set a Passphrase (Optional):

   You will be asked if you want to set a passphrase for added security. This passphrase will be required whenever you use your private key:

   Enter passphrase (empty for no passphrase):

▼ Copying Your Public Key to a Serve

Once you have generated your SSH key pair, you need to copy the public key (id_rsa.pub) to the server you want to connect to.

1. Automatic Method Using ssh-copy-id:

   If you have password-based SSH access to the server, you can use ssh-copy-id to automatically copy your public key:

```
ssh-copy-id username@hostname
```

   This command appends your public key to the ~/.ssh/authorized_keys file on the remote server.

2. Manual Method:

   If ssh-copy-id is not available, you can manually copy the key:

   a. Use cat to display the public key on your local machine:

```
cat ~/.ssh/id_rsa.pub
```

b. Copy the output.

c. Log into the remote server and open the ~/.ssh/authorized_keys file (create it if it doesn't exist)

1. ssh username@hostname
2. mkdir -p ~/.ssh
3. nano ~/.ssh/authorized_keys
4. Paste the public key into the file and save it.
5. Ensure the correct permissions:
   - chmod 600 ~/.ssh/authorized_keys
   - chmod 700 ~/.ssh

▼ Logging In Using SSH Key Authentication

Once your public key is set up on the remote server, you can log in without a password. Use the following command:

```
ssh username@hostname

# ssh gsharma@login.cc.kek.jp
```

SSH will automatically use your private key (if it is located in ~/.ssh/id_rsa).

- If your private key is stored in a non-standard location, you can specify it:

```
ssh -i /path/to/private_key username@hostname
```

▼ **SSH Agent for Convenience**

You can use an SSH agent to avoid typing your passphrase every time you log in. The SSH agent holds your private keys in memory and automatically provides them when needed:

1. Start the agent:

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

2. Add your private key to the agent:

```
ssh-add ~/.ssh/id_rsa
```

- Summary of Key SSH Commands

```
•   ssh username@hostname: Connect to a remote server.
•   ssh-keygen: Generate an SSH key pair.
•   ssh-copy-id username@hostname: Copy your public key to the server for key-based authenticatio
n.
•   ssh-add: Add your private key to the SSH agent for easier access.
•   scp [file] username@hostname:/path/to/destination: Securely copy files  to a remote server.
```

SSH key-based authentication is more secure than passwords because it uses strong cryptography, and the private key never leaves your local machine.

▼ **Automating the kekcc login process**

▼ **The SSH config file**

The SSH config file allows you to define configuration settings that make managing SSH connections more convenient. It is especially useful if you regularly connect to multiple servers, as it can streamline login commands, specify default settings, and set up custom behavior for different hosts.

▼ **Using the .bashrc (or .zshrc for Mac)**

The .bashrc file is a script that runs every time a new terminal session is started in an interactive shell (like when you open a terminal window or start a new SSH session). It is located in the user's home directory (~/.bashrc).

- This file allows you to customize your shell environment, define variables, **create aliases**, and set preferences like your prompt's appearance.

```
alias alias_name='command'
```

> 💡 If you have not setup the ssh config and Alias in the .bashrc to login to kekcc work server, please run the shell script in the next section.

▼ Shell Script for the above task

1. Download the shell script from the BAW2024 Indigo page.
2. Open the terminal and navigate to the folder which has the shell script.
3. Run the following cmd:

```
chmod +x setup_ssh.sh
```

4. Once done, Run the cmd

```
./setup_ssh.sh
```

▼ The script

```
#!/bin/bash

# Function to check if XQuartz is installed on macOS
install_xquartz_if_needed() {
  if ! command -v xauth &> /dev/null; then
    echo "XQuartz is not installed. Installing XQuartz..."
    brew install --cask xquartz
  else
    echo "XQuartz is already installed."
  fi
}

# Function to setup X11 forwarding for Ubuntu
setup_x11_forwarding_ubuntu() {
  echo "Setting up X11 forwarding for Ubuntu..."
  sudo apt-get install -y xauth
}

# Get user inputs
read -p "Enter your SSH username: " username
read -p "Do you have Keychain enabled? (yes/no): " use_keychain
if [[ "$use_keychain" == "yes" ]]; then
   read -p "Keychain name? (press enter for the id_rsa): " key_chain
  fi
 if [ -z "$key_chain" ]; then
   key_chain="id_rsa"
   fi
read -p "Enter the port for Jupyter connection (xxxx): " jupyter_port

# Check if the operating system is macOS or Ubuntu
os_type=$(uname)
if [[ "$os_type" == "Darwin" ]]; then
  echo "Detected macOS system."
  install_xquartz_if_needed
  shell_rc="$HOME/.zshrc"
elif [[ "$os_type" == "Linux" ]]; then
  echo "Detected Linux system."
  shell_rc="$HOME/.bashrc"
  if [ -f "/etc/lsb-release" ] && grep -q "Ubuntu" /etc/lsb-release; then
    setup_x11_forwarding_ubuntu
  else
    echo "Unsupported Linux distribution. Script supports only Ubuntu for now."
    exit 1
  fi
else
  echo "Unsupported operating system. Script supports macOS and Ubuntu only."
```

```
  exit 1
fi

# File where SSH config is stored
config_file="$HOME/.ssh/config"

# Function to check if a pattern exists in a file
check_if_exists() {
  grep -q "$1" "$config_file"
}

# Check if 'ServerAliveInterval', 'X11 Forwarding', and 'Host Kekcc' are already present
server_alive_interval_exists=$(check_if_exists "ServerAliveInterval")
x11_forwarding_exists=$(check_if_exists "ForwardX11 yes")
kekcc_block_exists=$(check_if_exists "Hostname login.cc.kek.jp")

# Append 'ServerAliveInterval' if not present
if ! check_if_exists "ServerAliveInterval"; then
  echo "Adding ServerAliveInterval to SSH config..."
  echo "
# try to keep the connection alive, avoids connection timeouts
ServerAliveInterval 60
" >> "$config_file"
else
  echo "ServerAliveInterval already present in SSH config."
fi

# Add X11 forwarding configuration based on the OS type if not already present
if ! check_if_exists "ForwardX11 yes"; then
  echo "Adding X11 Forwarding to SSH config..."
  if [[ "$os_type" == "Darwin" ]]; then
    echo "
    ForwardX11 yes
    ForwardX11Trusted yes
    XAuthLocation /opt/X11/bin/xauth
" >> "$config_file"
  elif [[ "$os_type" == "Linux" ]]; then
    echo "
ForwardX11 yes
ForwardX11Trusted yes
" >> "$config_file"
  fi
else
  echo "X11 Forwarding already present in SSH config."
fi

# Append the SSH configuration for KEKCC if not already present
if ! check_if_exists "Host Kekcc"; then
  echo "Adding SSH configuration for KEKCC..."
  config_block="

# SSH Configuration for KEKCC
Host Kekcc
    User $username
    Hostname login.cc.kek.jp
    Compression yes
    ProxyJump $username@sshcc1.kek.jp
"

  # Append UseKeychain line if the user has Keychain enabled
  if [[ "$use_keychain" == "yes" ]]; then
    config_block+="    IdentityFile ~/.ssh/${key_chain}
    IdentitiesOnly yes
    UseKeychain yes\n"
```

```
  fi

  # Append the SSH configuration block to the config file
  echo -e "$config_block" >> "$config_file"
else
  echo "KEKCC SSH configuration already present in SSH config."
fi

# Ask if the user wants to add aliases to the shell configuration file
read -p "Do you want to add aliases to your $shell_rc file? (yes/no): " add_aliases

if [[ "$add_aliases" == "yes" ]]; then
  # Add aliases to .bashrc or .zshrc based on the OS
  if [[ -f "$shell_rc" ]]; then
    echo "
# Aliases for KEKCC
alias kekcc=\"ssh -Y Kekcc\"
alias connectjupyter='ssh -L ${jupyter_port}:localhost:${jupyter_port} Kekcc'
" >> "$shell_rc"

    echo "Aliases added to $shell_rc. Please restart your terminal or run 'source $shell_rc
  else
    echo "Shell configuration file ($shell_rc) not found!"
  fi
else
  echo "Skipping alias addition."
fi

echo "SSH configuration successfully appended to $config_file."
```

5. Answer the questions honestly !!

```
Enter your SSH username: gsharma
Do you have Keychain enabled? (yes/no): yes
Keychain name? (press enter for the id_rsa):
Enter the port for Jupyter connection (xxxx): 5768
Do you want to add aliases to your /Users/gauravsharma/.zshrc file? (yes/no):
```

For the Jupyter connection (xxxx): you can only use number above 1024 unless you have administrator privileges. 8080 is a “typical” port used for forwardings and is usually free. But any number is fine.

6. Wallah !! It’s Done

▼ **Welcome to the KEKCC**

If you’ve followed everything above, Thank you. Your MC requests will be given higher priority !!

- To login, open a new terminal and write `kekcc` .
- It’ll ask for the password for the ssh key.
  - Type yes to the following msg and ignore the warning. This only comes when you login into a server for the first time.

```
The authenticity of host 'bastion.desy.de (131.169.5.82)' can't be established.
RSA key fingerprint is SHA256:WbkI/Ko+FdCbIAVn6ky2odyWxCvCL3+5XqWSZQ6PynE.
Are you sure you want to continue connecting (yes/no)?
```

Once you login, it should look like this,

```
gauravsharma@Gauravs-MacBook-Pro ~ % kekcc
********************************************************************************
* KEKCC Work Server (2024)                                                     *
```

```
*                                                                              *
* Support: https://wiki.kek.jp/display/kekcc                                   *
* User Guide: https://kekcc.kek.jp/service/kekcc/support/ja/ (ja)              *
*             https://kekcc.kek.jp/service/kekcc/support/en/ (en)              *
*                                                                              *
* Caution:                                                                     *
* If you login the system with ssh public key authentication,                  *
* please use a public/private key pair with a passphrase.                      *
* The system administrator deletes your .ssh directory                         *
* when a private key without a passphrase is detected.                         *
********************************************************************************
* Work Servers :                                                               *
* cw01-cw08.cc.kek.jp, login.cc.kek.jp                                         *
********************************************************************************
* Your home directory in the previous system is stored in the read-only dir,   *
* /gpfs/home/old/<group_namne>/<user_name>/.                                   *
* Please copy your data to the current home directory by yourself.             *
********************************************************************************
Last login: Sat Oct 19 13:24:12 2024 from 130.87.104.133
HOME directory usage: 4/100 GB (4%)
Your usage in /group/belle/users/: 163/1024 GB (16%)
```

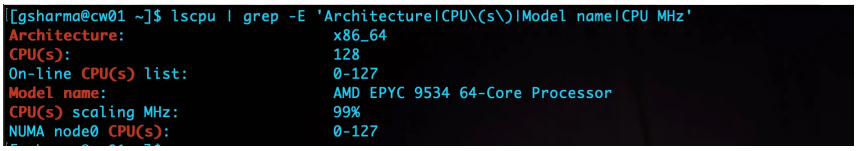
- By default, the available space per user is 100GB.

> 💡 You can get extra 1TB of space by following these cmds-
>
> 1. cd /group/belle/users
> 2. mkdir <your kekcc username>
> 3. Logout and login again.
>
> Q - add an alias for the path of this extra space.

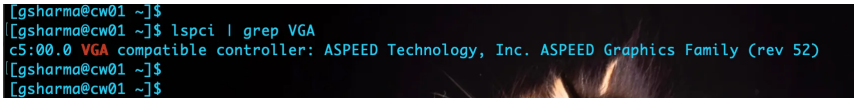
▼ **Working inside the KEKCC**

Once you've logged into the KEKCC, you have access to run your codes in a very powerful system.

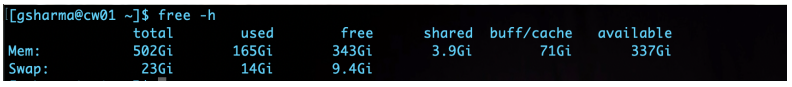
- CPU info



- GPU info



- RAM info



People often prefer using Jupyter Notebooks over text editors like vim or nano when working on a server for writing Python scripts, especially for specific tasks such as data analysis, scientific computing, or machine learning. The main reasons are:-

- User-Friendly Interface, Interactive Environment, Inline Visualisations, Markdown and Documentation, Ease of Debugging and Experimentation

▼ **Using the Jupyter Notebook at the KEKCC**

Follow these steps:-

1. Open a new terminal.
2. IF you ran the above mentioned shell script, run the command `connectjupyter` .

   If you have set some different Alias for this thn run that one. Basically we need the following cmd to be ran,

   ```
   ssh -L xxxx:localhost:XXXX username@servername
   ```

   - The first `xxxx` is the local port on your machine.
   - The second `xxxx` is the port on the remote server.
   - You do not need to use the same number on both sides, though it's often done for simplicity.

   > 💡 It'll be convenient if you use same number for the both port.

3. Now you are in the KEKCC. Use the `cd` cmd to navigate to the directory where you want to work.
4. Run the following cmds

   ```
   1. source /cvmfs/belle.cern.ch/tools/b2setup light-2409-toyger

   2. jupyter notebook --no-browser --port=xxxx

   Note:- xxxx = number that you've used in the ssh cmd to login.
   ```

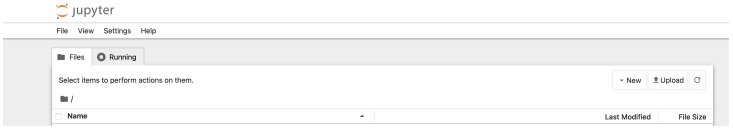
   The output will look like,

   ```
   [I 2024-10-19 18:27:03.083 ServerApp] jupyter_lsp | extension was successfully linked.
   [I 2024-10-19 18:27:03.086 ServerApp] jupyter_server_terminals | extension was successful
   [I 2024-10-19 18:27:03.089 ServerApp] jupyterlab | extension was successfully linked.
   [I 2024-10-19 18:27:03.089 ServerApp] jupyterlab_jupytext | extension was successfully li
   [I 2024-10-19 18:27:03.091 ServerApp] notebook | extension was successfully linked.
   [I 2024-10-19 18:27:03.363 ServerApp] notebook_shim | extension was successfully linked.
   [I 2024-10-19 18:27:03.379 ServerApp] notebook_shim | extension was successfully loaded.
   [I 2024-10-19 18:27:03.381 ServerApp] jupyter_lsp | extension was successfully loaded.
   [I 2024-10-19 18:27:03.381 ServerApp] jupyter_server_terminals | extension was successful
   [I 2024-10-19 18:27:03.385 LabApp] JupyterLab extension loaded from /cvmfs/belle.cern.ch/
   [I 2024-10-19 18:27:03.385 LabApp] JupyterLab application directory is /cvmfs/belle.cern.
   [I 2024-10-19 18:27:03.385 LabApp] Extension Manager is 'pypi'.
   [I 2024-10-19 18:27:03.402 ServerApp] jupyterlab | extension was successfully loaded.
   [W 2024-10-19 18:27:03.402 ServerApp] [Jupytext Server Extension] Async contents managers
   [I 2024-10-19 18:27:03.402 ServerApp] [Jupytext Server Extension] Deriving a JupytextCont
   [I 2024-10-19 18:27:03.403 ServerApp] jupyterlab_jupytext | extension was successfully lo
   [I 2024-10-19 18:27:03.406 ServerApp] notebook | extension was successfully loaded.
   [I 2024-10-19 18:27:03.406 ServerApp] Serving notebooks from local directory: /gpfs/home/
   [I 2024-10-19 18:27:03.406 ServerApp] Jupyter Server 2.14.1 is running at:
   [I 2024-10-19 18:27:03.407 ServerApp] http://localhost:1972/tree?token=3756638bf8c528f625
   [I 2024-10-19 18:27:03.407 ServerApp]     http://127.0.0.1:1972/tree?token=3756638bf8c528
   [I 2024-10-19 18:27:03.407 ServerApp] Use Control-C to stop this server and shut down all
   [C 2024-10-19 18:27:03.410 ServerApp]

       To access the server, open this file in a browser:
           file:///gpfs/home/belle2/gsharma/.local/share/jupyter/runtime/jpserver-737838-ope
       Or copy and paste one of these URLs:
           http://localhost:xxxx/tree?token=3756638bf8c528f6257d3845e627ae18ec585b2a862960f4
   ```

```
        http://127.0.0.1:xxxx/tree?token=3756638bf8c528f6257d3845e627ae18ec585b2a862960f4
[I 2024-10-19 18:27:03.426 ServerApp] Skipped non-installed server(s): bash-language-serv
```

5. Copy the links at the end to your browser and you have the Jupyter notebook running at the KEKCC.

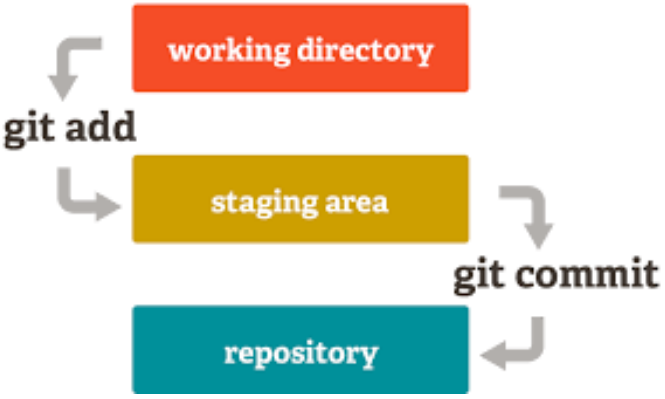


▼ Version control using the git

Why Use Git:

- Git is a distributed version control system designed to track changes in code, documents, or any set of files over time.

▼ Working within the local repository

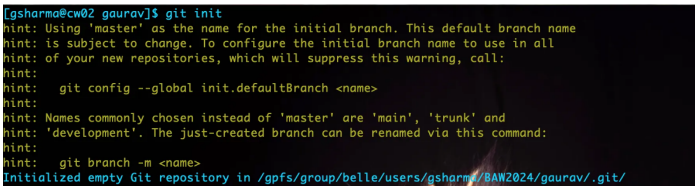


▼ How to Get Started with Git

1. Initialising the git

```
cd /path/to/your/project
git init
```

The output is



2. Adding the files (staging)

```
git add filename
```

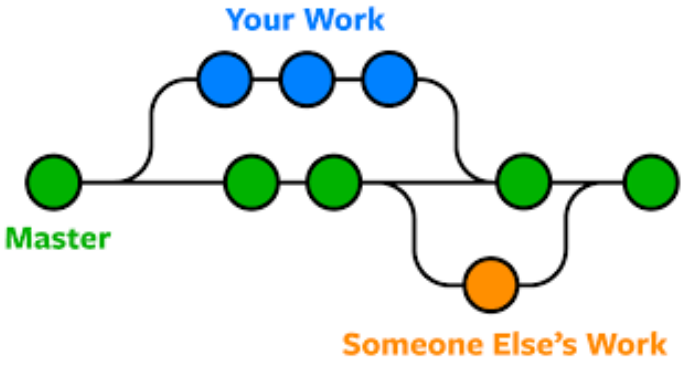
Stage all changes (all modified, added, or deleted files):

```
git add .
```

3. Commit Changes
    - Once files are staged, commit them to the repository. Commits act like snapshots, allowing you to track changes over time.

```
git commit -m "Initial commit"
```

6. Creating a branch



- Run the following cmd to create a new branch

```
git checkout -b <branch name>
```

- The output:- `Switched to a new branch 'branch name'`
- Run `git branch` to check the current branch.
- Run `git checkout <branch name>` to switch to existing branch.
    - You can checkout to someone else's branch and work.
- You can commit your work to keep track of changes. A meaningful commit message is very important, mainly when you working with other people.

7. If something goes wrong, you can always switch back to some existing
    - Identify the commit you want to reset to:
        - Use `git log` to view the commit history. This will display a list of commits.
        - Each commit has a unique hash.
        - Find the commit hash you want to reset to.
    - **Perform the reset**
      Use one of the following commands depending on how much you want to undo:
        1. **Soft reset**:

           This will move the HEAD to the specified commit and keep all your changes staged (ready for the next commit).

```
git reset --soft <commit-hash>
```

        2. **Mixed reset** (default):

           This will reset the HEAD to the specified commit but leave the changes in the working directory (they will no longer be staged).

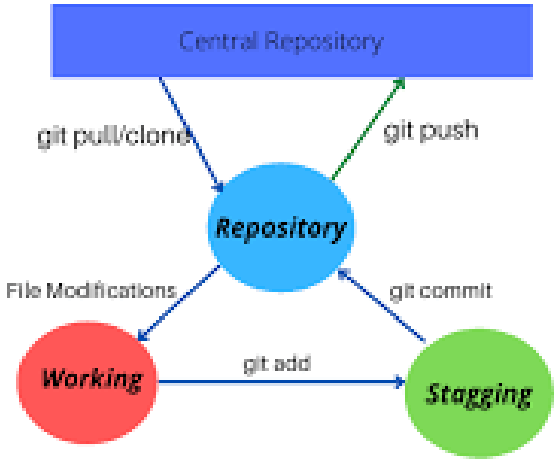
```
git reset --mixed <commit-hash>
```

        3. **Hard reset**:

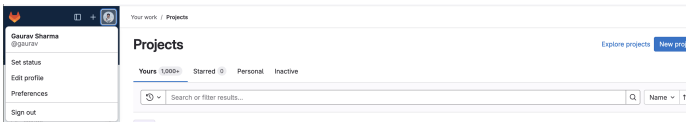
This will reset the HEAD to the specified commit and discard all changes after that commit, both from the staging area and the working directory. **Be cautious**: This will permanently delete any changes made after the commit you're resetting to.

```
git reset --hard <commit-hash>
```

▼ **Using the remote repository**



1. Set Up a Remote Repository
   - open the link in a browser:- https://gitlab.desy.de/users/sign_in
   - Sign-in using the desy credentials
   - click on the logo on the top left pannel to know your gitlab username.



   - run the following cmd in your main git folder (where you ran the git init)

```
git remote add origin git@gitlab.desy.de:<gitlab username.>/<project name>.git
```

1. Pushing the changes to the remote repo
   - Run the cmd

```
git push --set-upstream origin <branch name>
```

▼ **Clubbing the VS Code with KEKCC**

- Using VS Code + SSH, you can interact with files on the remote server just as you would with local files. You can open, edit, save, and run code seamlessly.
- Convenience of working in your preferred development environment while accessing remote server resources, allowing for high productivity without switching contexts.
- VS Code provides an integrated terminal for the remote server via SSH.

- You can run commands, execute scripts, and manage server processes directly within the editor, providing a unified interface.

▼ Install the VS Code

- For Ubuntu:

  ```
  sudo snap install --classic code
  ```

  Enter the sudo user password i.e. system login password.

- For the MacOS:

  ⚠ Must have the Homebrew installed

  ```
  brew install --cask visual-studio-code
  ```

  Or, Install the image from the VS Code webpage and install it in the Mac's cool way 😉.