**Add any new repo or specific folder in repo:**

* git init
* git remote add remote\_name repo\_url
* git push –u remote\_name branch\_name
* git add .
* git commit –m “message or comment”

# Basic Git Commands:

**Setting up Git:**

**Changing the username/email:**

1. **change Global username/email**

* *git config –global user.name “temi"*
* *git config –global user.email “taimoorrauf607@gmail.com"*

1. **change local username/email (current repository)**

* *git config – local unset user.name “temi2"*
* *git config –local user.email “taimoor607607@gmail.com"*

**Deleting the username/email:**

1. **Delete global username/email**

* *git config -- global --unset user.name*
* *git config -- global --unset user.email*

1. **Delete local username/email**

* *git config --unset user.name*
* *git config --unset user.email*

**Verifying username/ email:**

1. **Check global username/email**

* git config --global user.name
* git config --global user.email

1. **Check local username/email**

* git config --local user.name
* git config --local user.email
* **Set your username**  
  git config --global user.name temii
* **Set your email**  
  git config --global user.email taimoorrauf607@gmail.com

**Repository Commands:**

* **Initialize a new Git repository**  
  git init
* **Clone an existing repository**  
  git clone <repository\_url>
* **Check the status of the repository**  
  git status
* **Check repository info (like remotes)**  
  git remote -v
* **Add a file to staging**  
  git add <filename>
* **Add all files to staging**  
  git add .
* **Commit changes with a message**  
  git commit -m "Your commit message"

git commit -a -m "message" command to commit changes in Git with a message.

* **View commit history**  
  git log
* **Delete files from stage area:**  
   git rm –cached filename it will transfer file to untracked
* **Delete files:**

Git rm filename

* Switch to last commit

git checkout –f is used to **force** the checkout operation for all files.

**Branching & Merging:**

* **List all branches**  
  git branch
* **Create a new branch**  
  git branch <branch\_name>
* **Switch to a branch – match last commit status (recover data)**  
  git checkout <branch\_name>
* **Create and switch to a new branch**  
  git checkout -b <branch\_name>
* **Merge a branch into the current branch**  
  git merge <branch\_name>
* **Delete a branch**  
  git branch -d <branch\_name>
* **Rename a branch**  
  git branch -m <old\_branch\_name> <new\_branch\_name>

**Remote Repositories:**

* **View remote repositories**  
  git remote -v
* **Add a remote repository**  
  git remote add <name> <repository\_url>
* **Push changes to a remote repository**  
  git push –u <remote\_name> <branch\_name>
* **Pull changes from a remote repository**  
  git pull <remote\_name> <branch\_name>
* **Fetch changes from a remote repository**  
  git fetch <remote\_name>
* **Remove a remote repository**  
  git remote remove <name>
* **Download any repository in your pc**
* git colne url foldername

**Viewing Changes:**

* **Show differences between staged and working directory**  
  git diff
* **Show differences between commits**  
  git diff <commit1> <commit2>
* **Show commit history with details**  
  git log --oneline
* **Show commit history with differences**  
  git log –p –no of commit
* **Show the status of tracked and untracked files**  
  git status
* **Show the status of tracked and untracked files**  
  git status -s
* **Show the status of tracked and untracked files**  
  git status -s

**Undoing Changes:**

* **Unstage a file (remove from staging area)**  
  git reset <filename>
* **Undo a commit (soft reset, keep changes)**  
  git reset --soft HEAD~1
* **Undo a commit (hard reset, discard changes)**  
  git reset --hard HEAD~1
* **Revert a commit (create a new commit to undo the changes)**  
  git revert <commit\_id>
* **Remove untracked files**  
  git clean -fcheck

**Other Useful Commands:**

* **Show details about the last commit**  
  git show
* **Tag a commit**  
  git tag <tag\_name>
* **Push tags to remote repository**  
  git push --tags
* **Stash changes (save work temporarily)**  
  git stash
* **Apply stashed changes**  
  git stash apply
* **Delete stashed changes**  
  git stash drop
* **Show the history of a specific file**  
  git log -- <file\_name>

**Common Aliases (to save time):**

* **Create alias for status**  
  git config --global alias.st status
* **Create alias for commit**  
  git config --global alias.ci commit
* **Create alias for checkout**  
  git config --global alias.co checkout
* **Create alias for log**  
  git config --global alias.lg "log --oneline --graph --decorate"

**Git Help:**

* **Get help for a specific command**  
  git <command> --help
* **Get general Git help**  
  git help

# .gitignore file cheatsheet

**Basic Syntax:**

* Blank lines are ignored.
* Lines starting with # are comments.
* Wildcard (\*) matches any number of characters.
* Wildcard (?) matches a single character.
* Directory separator (/) indicates a directory.
* ! negates a pattern, including the file or directory.

**Common Patterns:**

* \*.log: Ignore all .log files.
* example.txt: Ignore a specific file.
* build/: Ignore a directory named build.
* docs/\*.txt: Ignore .txt files in the docs folder.
* .gitignore: Do not ignore .gitignore itself.
* !\*important.txt: Exclude important.txt from being ignored.

**Wildcard Examples:**

* \*: Ignore all files.
* \*.log: Ignore all .log files.
* \*/: Ignore all directories.
* \*.tmp: Ignore all .tmp files.
* temp\*: Ignore files starting with temp.
* \*/\*.log: Ignore .log files in all subdirectories.

**Common Use Cases:**

* **Node.js**: node\_modules/, .env
* **Python**: \*.pyc, \_\_pycache\_\_/, venv/
* **Java**: \*.class, target/
* **macOS**: .DS\_Store, Thumbs.db
* **IDE files**: .idea/, .vscode/

**Negating Files (Inclusion After Exclusion):**

* \*: Ignore everything.
* !important.txt: Exclude important.txt from being ignored.

**Examples:**

* **Ignore logs and node\_modules**:  
  \*.log, node\_modules/
* **Ignore .DS\_Store (macOS)**:  
  .DS\_Store
* **Ignore Python cache files**:  
  \*.pyc, \_\_pycache\_\_/
* **Ignore a folder but not a specific file**:  
  docs/\*, !docs/README.md

This concise .gitignore guide helps you efficiently manage which files Git tracks and which it ignores!