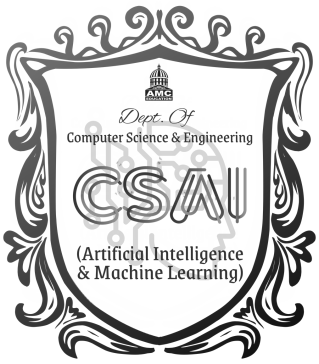
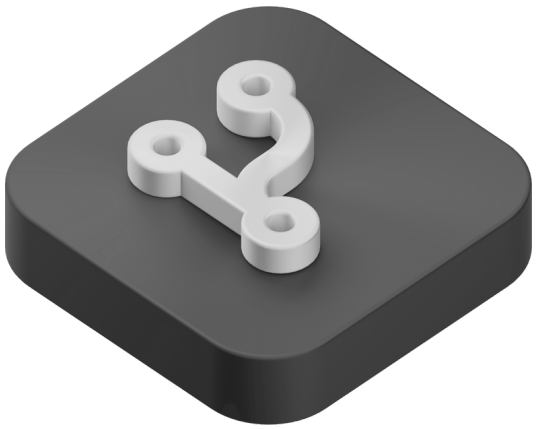


Lab Manual

Project Management With Git

SUB CODE: BCS358C



Beginner's Cheatlist & All Experiments

Compiled By Tarun Balaji K S

INDEX

SETTING UP

It it important to Setup GIT in your machine once,
so that you can use it whenever you like.

You can use the following commands in GIT BASH
(which you need to install from https://git-scm.com/downloads)

git config --global user.name "Your Name"

This is used to set your name as the author for all future commits globally across your machine.

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

This is used to set your email for future commits globally.

git config --global core.editor "code --wait"

This is used to configures the default editor for commit messages (in this case, VS Code).

INITIALIZING

This command is used to Accept all files available in that Directory, and start a new repository for tracking purposes or to basically continue the process.

git init

This is used to initialize a new Git repository in your current directory, creating a .git directory.

git clone {url}

This is used to clone the contents of a Git repository that is remotely hosted (in platforms like Github or Gitlab)

git remote or git remote -y

This is used to see which remote repository are you making changes into. -y Flag shows more information.

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

This is used to set your email for future commits globally.

git remote add upstream {url}

This is used to add the Git upstream to the mentioned URL.

STATUS

This command is used to check the Status of the file with respect to the version tracking in Git. All file fall under one category - modified, staged, or committed.

It tells us if any new file has been added, is it being tracked, is it modified etc.

git status

This is used to display which files are modified, staged, or untracked in the working directory.

ADDING CHANGES

This command is used to Add the files to the staging area.

git add filename

This is used to stage specific file changes for the next commit

git add .

This is used to display which files are modified, staged, or untracked in the working directory.

git reset filename

This is used to remove a added file from the staging area.

COMMITING

This command is used to save the made changes that have been added to the staging area, into the Remote Repository under your name.

git commit -m "Your message"

This is used to record the staged changes in the local repository with a message describing the changes.

git commit --amend -m "Updated message"

This is used to modifies the last commit message without changing the commit content.

HISTORY

This command is used to view the history of the files. The whole point of version control is to be able to see the history and rollback to the previous version incase of problems.

git log

This is used to see the commit history of the current branch, including commit IDs, authors, and messages.

git log --oneline --graph

This is used to display a simplified, one-line summary of each commit along with a visual representation of the branch structure.

BRANCHING

Many of times, we don't want to disturb the code which is working perfectly. In such situations, we Isolate a copy of the whole codebase and experiment in that.
The branch you are currently working on is called as "Active Branch"

git branch

This is used to list all current branches. An asterisk (*) will appear next to your currently active branch.

git branch branch_name

This is used to creates a new branch with the specified name.

git switch branch_name

This is used to switch to the specified branch, making it the active branch.

git merge branch_name

This is used to combine the specified branch into the current branch. Git will try to automatically merge any changes.

git branch -d branch_name / -D branch_name

This is used to delete the specified branch locally, if it's fully merged into another branch. -D Flag is used to forcefully delete the specified branch, even if it hasn't been merged.

MAKING CHANGES

Now that we have made changes to the files in our system, or in our local repository, We need to save this change into the Remote Repository.

Usually, there is the push action - To send all the changes and the pull action - To fetch all the changed files

git push origin branch_name (main)

This is used to send committed changes in your local repository to the remote repository.

git pull origin branch_name (main)

This is used to creates a new branch with the specified name.

git switch branch_name

This is used to fetch and integrate changes from the remote repository into your local working directory.

STASHING

git stash

This is used to temporarily save uncommitted changes without adding them to a commit, allowing you to work on something else.

git stash apply/pop

Re-applies the most recent stash to the working directory without removing it from the stash list / by removing it from the stash list.

EXPERIMENT 1

Setting Up and Basic Commands:

Initialize a new Git repository in a directory. Create a new file and add it to the staging area and commit the changes with an appropriate commit message.

Solution:

To initialize a new Git repository in a directory, create a new file, add it to the staging area, and commit the changes with an appropriate commit message, follow these steps:

1. Open your terminal and navigate to the directory where you want to create the Git repository.
2. Initialize a new Git repository in that directory:

$ git init

3. Create a new file in the directory. For example, let's create a file named "my_file.txt." You can use any text editor or command-line tools to create the file.

4. Add the newly created file to the staging area. Replace "my_file.txt" with the actual name of your file:

$ git add my_file.txt

This command stages the file for the upcoming commit.
5. Commit the changes with an appropriate commit message. Replace "Your commit message here" with a meaningful description of your changes:

$ git commit -m "Your commit message here"

Your commit message should briefly describe the purpose or nature of the changes you made. For example:

$ git commit -m "Add a new file called my_file.txt"

After these steps, your changes will be committed to the Git repository with the provided commit message. You now have a version of the repository with the new file and its history stored in Git.

EXPERIMENT 2

Creating and Managing Branches:

Create a new branch named "feature-branch." Switch to the "master" branch. Merge the "feature-branch" into "master."

Solution:

To create a new branch named "feature-branch," switch to the "master" branch, and merge the "feature-branch" into "master" in Git, follow these steps:
1. Make sure you are in the "master" branch by switching to it:

$ git checkout master

2. Create a new branch named "feature-branch" and switch to it:

$ git checkout -b feature-branch

This command will create a new branch called "feature-branch" and switch to it.
3. Make your changes in the "feature-branch" by adding, modifying, or deleting files as needed.
4. Stage and commit your changes in the "feature-branch":

$ git add .
$ git commit -m "Your commit message for feature-branch"

Replace "Your commit message for feature-branch" with a descriptive commit message for the changes you made in the "feature-branch."
5. Switch back to the "master" branch:

$ git checkout master

6. Merge the "feature-branch" into the "master" branch:

$ git merge feature-branch

This command will incorporate the changes from the "feature-branch" into the "master" branch. Now, your changes from the "feature-branch" have been merged into the "master" branch. Your project's history will reflect the changes made in both branches

EXPERIMENT 3

Creating and Managing Branches:

Write the commands to stash your changes, switch branches, and then apply the stashed changes.

Solution:

To stash your changes, switch branches, and then apply the stashed changes in Git, you can use the following commands:
1. Stash your changes:

$ git stash save "Your stash message"

This command will save your changes in a stash, which acts like a temporary storage for changes that are not ready to be committed.

2. Switch to the desired branch:

$ git checkout target-branch

Replace "target-branch" with the name of the branch you want to switch to.

3. Apply the stashed changes:

$ git stash apply

This command will apply the most recent stash to your current working branch. If you have multiple stashes, you can specify a stash by name or reference (e.g., git stash apply stash@{2}) if needed.

If you want to remove the stash after applying it, you can use git stash pop instead of git stash apply.

Remember to replace "Your stash message" and "target-branch" with the actual message you want for your stash and the name of the branch you want to switch to.

EXPERIMENT 4

Collaboration and Remote Repositories:

Clone a remote Git repository to your local machine.

Solution:

To clone a remote Git Repository or Git Repo, first Navigate to the Directory where you want to save the files. Here open the Git bash.

Cope the URL of the Git Repo you want to copy and replace it with <repository_url> in the command below.

Use the following command to Clone

$ git clone <repository_url>

Here is a Full Example

$ git clone https://github.com/tarunbalajiks/Speech-Emotion-Recognition.git

Replace https://github.com/tarunbalajiks/Speech-Emotion-Recognition.git with the URL of the Repo you want to clone.

Git will clone the repository to your local machine. Once the process is complete, you will have a local copy of the remote repository in your chosen directory.

You can now work with the cloned repository on your local machine, make changes, and push those changes back to the remote repository as needed.

EXPERIMENT 5

Creating and Managing Branches:

Fetch the latest changes from a remote repository and rebase your local branch onto the updated remote branch.

Solution:

To fetch the latest changes from a remote repository and rebase your local branch onto the updated remote branch in Git, follow these steps:

1. Open your terminal or command prompt / Git Bash.
2. Make sure you are in the local branch that you want to rebase. You can switch to the branch using the following command, replacing with your actual branch name:

$ git checkout <branch-name>

3. Fetch the latest changes from the remote repository. This will update your local repository with the changes from the remote without merging them into your local branch:

$ git fetch origin

Here, origin is the default name for the remote repository. If you have multiple remotes, replace origin with the name of the specific remote you want to fetch from.

4. Once you have fetched the latest changes, rebase your local branch onto the updated remote branch:

$ git rebase origin/ <branch-name>

Replace <branch-name> with the name of the remote branch you want to rebase onto. This command will reapply your local commits on top of the latest changes from the remote branch, effectively incorporating the remote changes into your branch history.

5. Resolve any conflicts that may arise during the rebase process. Git will stop and notify you if there are conflicts that need to be resolved. Use a text editor to edit the conflicting files, save the changes, and then continue the rebase with:

$ git rebase --continue

6. After resolving any conflicts and completing the rebase, you have successfully updated your local branch with the latest changes from the remote branch.

7. If you want to push your rebased changes to the remote repository, use the git push command. However, be cautious when pushing to a shared remote branch, as it can potentially overwrite other developers' changes:

$ git push origin <branch-name>

Replace <branch-name> with the name of your local branch. By following these steps, you can keep your local branch up to date with the latest changes from the remote repository and maintain a clean and linear history through rebasing.

EXPERIMENT 6

Collaboration and Remote Repositories:

Write the command to merge "feature-branch" into "master" while providing a custom commit message for the merge.

Solution:

To merge the "feature-branch" into "master" in Git while providing a custom commit message for the merge,

Use the following command

$ git checkout master
$ git merge feature-branch -m "Your custom commit message here"

Replace "Your custom commit message here" with a meaningful and descriptive commit message for the merge. This message will be associated with the merge commit that is created when you merge "feature-branch" into "master."

EXPERIMENT 7

Git Tags and Releases:

Write the command to create a lightweight Git tag named "v1.0" for a commit in your local repository.

Solution:

To create a lightweight Git tag named "v1.0" for a commit in your local repository,

Use the following command

$ git tag v1.0

This command will create a lightweight tag called "v1.0" for the most recent commit in your current branch. If you want to tag a specific commit other than the most recent one, you can specify the commit's SHA-1 hash after the tag name. For example:

$ git tag v1.0

Replace with the actual SHA-1 hash of the commit you want to tag.

EXPERIMENT 8

Advanced Git Operations:

Write the command to cherry-pick a range of commits from "source-branch" to the current branch.

Solution:

To cherry-pick a range of commits from "source-branch" to the current branch,

Use the following command to Clone

$ git cherry-pick <start-commit>^.. <end-commit>

Replace <start-commit> with the commit at the beginning of the range, and <end-commit> with the commit at the end of the range. The ^ symbol is used to exclude the <start-commit> itself and include all commits after it up to and including . <end-commit>. This will apply the changes from the specified range of commits to your current branch.

For example, if you want to cherry-pick a range of commits from "source-branch" starting from commit ABC123 and ending at commit DEF456, you would use:

$ git cherry-pick ABC123^..DEF456

Make sure you are on the branch where you want to apply these changes before running the cherry-pick command.

EXPERIMENT 9

Analyzing and Changing Git History:

Given a commit ID, how would you use Git to view the details of that specific commit, including the author, date, and commit message?

Solution:

To view the details of a specific commit, including the author, date, and commit message, you can use the git show or git log command with the commit ID. Here are both options:

1. Using git show:

$ git show <commit-ID>

Replace <commit-ID> with the actual commit ID you want to view. This command will display detailed information about the specified commit, including the commit message, author, date, and the changes introduced by that commit.

For example,

$ git show abc123

2. Using git log:

$ git log -n 1 <commit-ID>

Replace <commit-ID> with the actual commit ID you want to view. This command will display a condensed view of the specified commit, including its commit message, author, date, and commit ID

For example,

$ git log -n 1 abc123

Both of these commands will provide you with the necessary information about the specific commit you're interested in.

EXPERIMENT 10

Analysing and Changing Git History:

Write the command to list all commits made by the author "JohnDoe" between "2023 01-01" and "2023-12-31."

Solution:

To list all commits made by the author "JohnDoe" between "2023-01-01" and "2023-12-31" in Git, you can use the git log command with the --author and --since and --until options.

Use the following command

$ git log --author="JohnDoe" --since="2023-01-01" --until="2023-12-31"

This command will display a list of commits made by the author "JohnDoe" that fall within the specified date range, from January 1, 2023, to December 31, 2023. Make sure to adjust the author name and date range as needed for your specific use case.

EXPERIMENT 11

Analysing and Changing Git History:

Write the command to display the last five commits in the repository's history.

Solution:

To display the last five commits in a Git repository's history, you can use the git log command with the -n option, which limits the number of displayed commits,

Use the following command

$ git log -n 5

This command will show the last five commits in the repository's history. You can adjust the number after -n to display a different number of commits if needed.

EXPERIMENT 12

Analysing and Changing Git History:

Write the command to undo the changes introduced by the commit with the ID "abc123".

Solution:

To undo the changes introduced by a specific commit with the ID "abc123" in Git, you can use the git revert command. The git revert command creates a new commit that undoes the changes made by the specified commit, effectively "reverting" the commit,

Use the following command

$ git revert abc123

Replace "abc123" with the actual commit ID that you want to revert. After running this command, Git will create a new commit that negates the changes introduced by the specified commit. This is a safe way to undo changes in Git because it preserves the commit history and creates a new commit to record the reversal of the changes.

CONCLUSION

This GIT Cheat list and Manual covers the essential commands to get you started, and all the Experiments listed in your VTU Syllabus, but there's a **lot more to explore.**

While it's not necessary to memorize every command, it's crucial to have the **basic commands at your fingertips.**

These form the foundation for most Git operations, allowing you to navigate version control efficiently.
As you grow more familiar with Git, you can gradually learn more advanced commands based on your needs, but mastering the basics is key to becoming proficient in version control.

Remember, Practice is the Key.
The more you use it, the more you get familiar with it.
Try Integrating Git with all Project you work on from now!

Hoping you found this useful,
Tarun Balaji K S

Dept. of Computer Science and Engineering (AI-ML)