

H05d3A: COMPUTER ARCHITECTURES

GIT INTRODUCTION

INTRO

Git is a version control system that has become ubiquitous in software development. This tool helps you to keep track of your code by saving an historic of the change you made to your code base. Git also allows you to keep multiple version of your code base available through a code branching system. Git also allows for remote deployment making it a powerfull tool to share and backup your code.

CORE CONCEPTS

To work with git, having a good understanding of its core concepts is important.

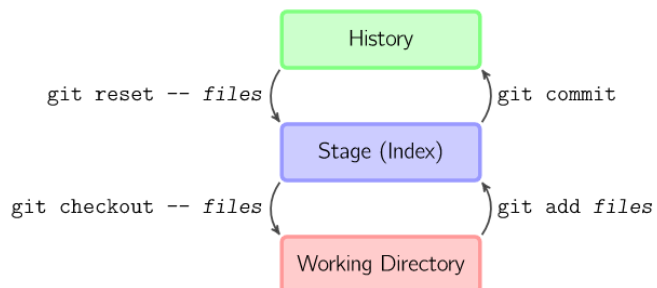
COMMIT GRAPH

As mentioned in the introduction, git helps you keep track of the changes that are made to your code. To do so, the changes made to your files in you code base are stored in a **commit graph**. When in a git repository, this graph is accessible to you using the **git log** command. In the graph, each commit is identified by a unique hash, its author, a date and a message describing the nature of the changes operated in the code base.

LOGICAL AREAS

To interact with this graph during developments, git works with 3 different logical areas.

1. The working directory contains the file that you can see in your file system and edit using you text editor.
2. The History contains the commit graph mentioned earlier.
3. The stage contains the files and modifications that we want to put in our next commit.



You can see the state of current logical areas using the **git status** command.

You can transfer files and changes to and from the different git area though git commands:

1. From working to stage:

`> git add <file>` adds the specified <file> to your staging area.

Alternatively, you can add all the files in your working tree by using:

`> git add -A`

2. From stage to working:

`> git checkout -- <file>` resets the specified file to the last staged changes.

3. From stage to History:

```
> git commit -m "commit message"
```

4. From History to Stage:

```
> git reset HEAD <file>
```

 resets the specified file to its last committed state in the stage.

5. Removing file:

```
> git rm <file>
```

 removes the file from your working directory and signals that it has been removed to stage area.

6. See differences between the working tree and the stage:

```
> git diff
```

 shows you the differences between your working tree and your stage area.

7. See differences between the stage and History:

```
> git diff --staged
```

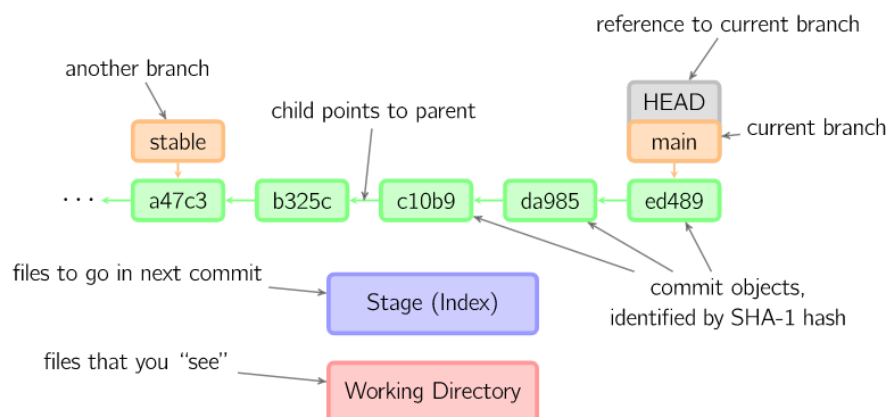
BRANCHING AND MERGING

Branches allow you to work on multiple versions of your code base at the same time. This allows you to support multiple code bases for multiple purposes. As an example, the master branch is generally dedicated to production-ready code. The dev branch contains the latest changes made to the code, etc.

Branches are implemented using pointers pointing at a unique commit hash indicating the latest commit done on that branch. The way Git knows on which branch you are, is through a special pointer called HEAD that generally points to a branch.

You can see where your HEAD pointer is currently at by using the `git log` command. Your HEAD pointer should appear next to a commit, pointing at a branch name (example: HEAD -> main).

When a new commit gets created, the branch pointer pointed by your HEAD pointer gets updated to point at this latest commit.



BRANCH CREATION

1. Create branch:

```
> git branch <branch name>
```

 creates a new branch at the location of your current location of your HEAD pointer.

Alternatively:

`> git checkout -b <branch_name>` will create a new branch and update your HEAD pointer to point at the newly created branch.

2. List branch:

`> git branch` will list the branches available and put an asterisk next to the one pointed by your HEAD pointer

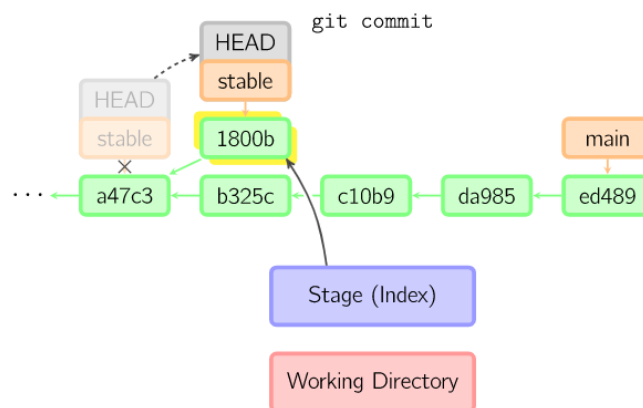
3. Go to branch:

`> git checkout <branch_name>` will move your HEAD pointer to the latest commit of branch <branch_name> but also update your staged files and working directory to match <branch_name> latest commit.

Note that the command will if staged changes have not been committed or stashed (see git stash command documentation)

Git checkout can also be used to checkout specific commits:

`> git checkout <commit_hash>` will move your HEAD to the specified commit and update your stage and working directory to match the commit. Doing so will result in what is called a detached HEAD. Any commit done in the detached head will be lost at the next checkout if a branch is not created on it.



MERGING

As the name suggests, the branching systems allows for parallel branches in your development tree. In the context of development, branches are often created to support new feature or keep the development separate from the production ready code. At some point though, these features will need to be incorporated to other branches.

To merge, checkout the branch to which you want to merge (example: `git checkout dev`) and call:

`> git merge <branch to merge in>` (example: `git merge my_cool_feature`)

From this command, 3 scenarios can occur:

1. Fast forward merge:

If all the commits on the branch you want to merge are posterior to the latest commit on your branch, your branch pointer gets simply moved to the latest commit on the branch you want to merge in.

Example:

You created a new branch from main called dev by calling `git branch dev`. You committed some changes but nothing changed on main. You can then checkout to main (`git checkout main`) and merge the two branches (`git merge dev`).

2. 3-way merge

If changes have been committed to each branches since their last common commit, a 3 way merge will happen. If none of the changes overlap (not the same files at the same lines edited), git will simply ask you to enter a message to create a merge commit.

3. 3 way merge with conflicts

If the same files have been modified, git will ask you to solve the conflicts before making the merge commit.

To do so, use `git status` to see the conflicting files. In the files, conflicting parts should be marked using the following syntax:

```
<<<<<<< HEAD
```

```
Your piece of code
```

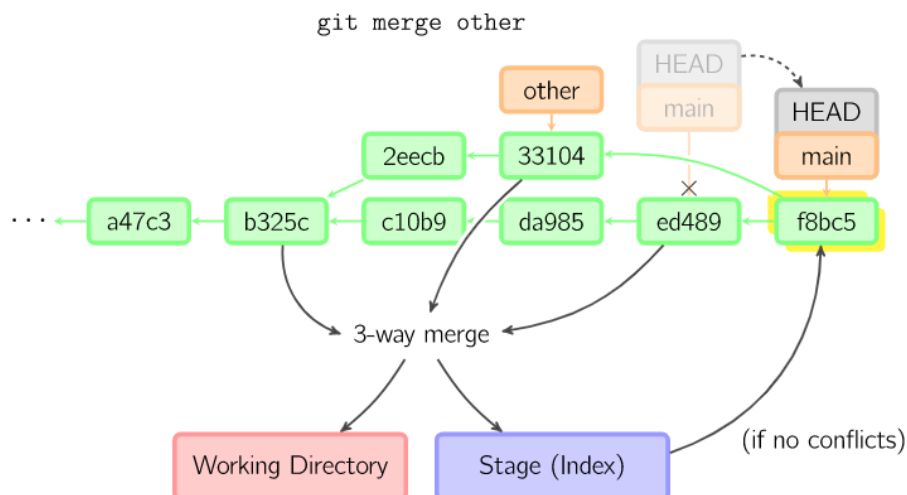
```
=====
```

```
Their piece of code
```

```
>>>>>>> their_branch
```

Decide what the code should do, remove the markers, stage the changes and make you commit.

Alternatively, stop the merging process by calling `git merge -abort`



REMOVING

Once you are done with your branch (it has been merged into an other branch) you can delete it by calling:

```
> git branch -d <branch to delete>
```

WORKING WITH REMOTES

One of the key advantages of git is the possibility to work with remotes, that is code residing on a distant server such as github. To do so, git introduces a new set of branch pointer to your commit graph corresponding to your remote branches.

If you followed the github instructions while setting up your repository on github, git log should show you a branch called origin/main or origin/master.

FETCHING

Remote branches do not get automatically synchronized on your local machine. To get the latest changes from your remote (new commits and branches) run the command:

```
> git fetch
```

After fetching, it is possible that your local branch is behind the equivalent remote branch. To check, run git status, a message should tell you if your branch is behind the remote. In which case, you can attempt to merge the remote changes to your local branch:

```
> git merge origin/<my_branch_name>
```

PUSHING

In order to make changes to the remote code and save your code online, use the git push command.

If a remote branch tracking your local branch already exist and your local branch is a head of the remote branch, simply call:

```
> git push
```

If the branch you are currently on is not tracked by a remote yet then call


```
> git push origin <my_branch_name>
```

 to create a new branch on your remote.

On github, you will be able to see your new branch with the code you committed on it.

RECOMMENDATION FOR PROJECT AND GOOD PRACTICES

PROJECT SET UP

For this project, we created a github repository to give you a good starting point with the code (https://github.com/KUL-CA-Exercise/CA_Exercise). We recommend that you create a fork of this repository on your own github account, by clicking on the icon  Fork 1

Install git if it is not already present on your local machine (<https://git-scm.com/downloads>)

Follow this setup using your local and ESAT remote machine: <https://docs.github.com/en/github/authenticating-to-github/adding-a-new-ssh-key-to-your-github-account>).

Clone the repository to both your local development machine and your ESAT machine using:

```
> git clone git@github.com:<your github account>/CA_Exercise.git
```

We recommend that your project should be organized with 3 branches session_1, session_2 and session_3. Start by creating the session_1 branch code and validate the functionality of the code. Branch from session_1 to create session_2 and repeat the process for session 2 and session 3. This way the branches will correspond to the 3 versions of the code that you will need to submit at the end of these sessions. This will also insure that at all point a clean version of the code exists somewhere.

GOOD PRACTICES

Do not push on main/master unless you are very sure of what you are doing. In a production environment, main/master is the code that works, that is production ready. A push to main/master should be done though github using the pull request system. If your changes are accepted, the pull request will be accepted, and your branch will be merged into main.

Avoid coding on the same branch as someone else if possible. 2 people coding on the same remote branch might end up bothering each other more than anything. Our suggestion is to make branches identified with the name of the person working on it (example: nlaubeuf/mycoolfeature). This should avoid issue such as not being able to push your daily code because someone committed on your branch and now you have to pull the changes and fix all the conflics that it created (never a good time).

SWEET REFERENCES

To go a bit further, we recommend:

https://www.youtube.com/watch?v=uR6G2v_WsRA

<https://www.youtube.com/watch?v=FyAAIHHClqI>

<https://www.youtube.com/watch?v=Gg4bLk8cGNo>

<https://marklodato.github.io/visual-git-guide/index-en.html>

<https://git-scm.com/book/en/v2>