Git and GitHub

1. Why Git?

- Backing of our project at different states, like
    - after we solve a bug?
    - before trying out a new feature?
- Working on a team, how
    - do we manage which copy is the *central* version if each developer works in its own?
    - can we allow for more than one developer to work in the same file?
    - can we track what changes have been done by each developer?

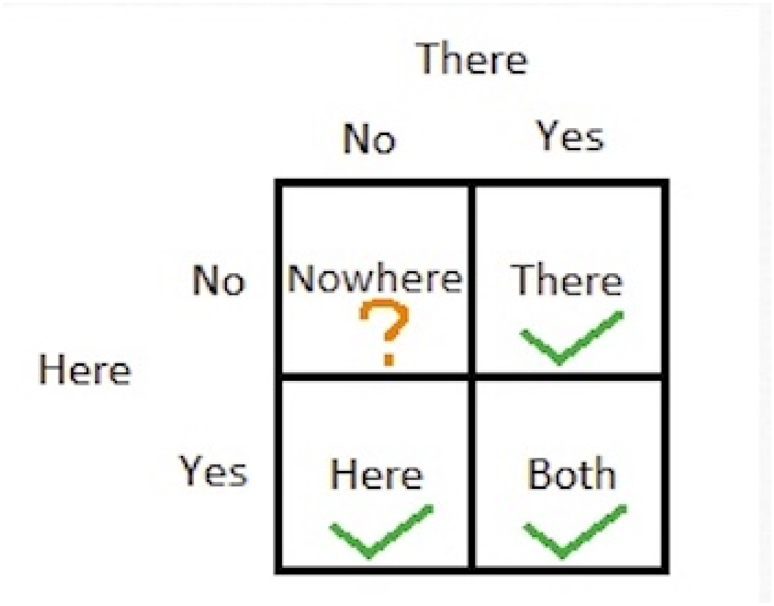
2. Version Control Systems (VCS)

- A complete long-term change history of every file
    - Creation, edition and deletion of files
- Revert files to previous states
- Revert entire projects back to previous states
- Compare changes over time
- Branching and mergin
    - Individuals working on independent streams of changes
    - Enables to verify that the changes on each branch do not conflict
- Traceability
    - Trace each change made to the software by each contributor
    - Connect it to project management and bug tracking software such JIRA[1].

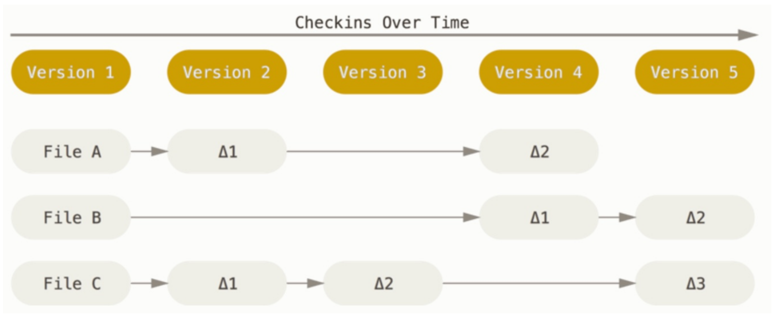
[1] https://www.web.com

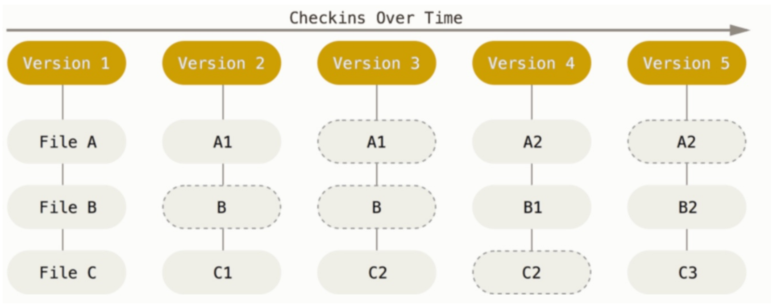
3. Different flavors:

- Non-Existing
- Local(here)
- Centralized(there) like for example CVS or Subversion (also known as SVN)
- Distributed(both) like Git or Mercurial



4. Diffs VS Snapshots





5. What is Git?

- By far, the most widely used modern version control system in the world.
- Is a mature, actively maintained open source project originally developed in 2005 by Linus Torvalds[2] (Linux operating system kernel).
- An example of a DVCS:
    - Everything is local
    - Is really fast
    - Snapshots, not diffs
    - Distributed not centralized

6. Installing Git

- **Windows**: https://git-scm.com/download/win
- **Mac**: https://git-scm.com/download/mac

Then we check the installation by checking git’s version:

```
$ git --version
```

Possible outputs:

- `git version 2.10.1`
- `bash: git: command not found`

[2] https://www.wikipedia.com/linux

7. Configuring Git:

- Our **identity**, so every time we make a commit, it is associated to our person:

```
$ git config --global user.name "your name"
$ git config --global user.email your.email@propulsion.ch
```

- The **editor** to be used if git wants to show us something:

```
git config --global core.editor <vim, emacs, subl, atom....>
git config --global core.editor "subl -n -w"
git config --global core.editor "atom --wait"
```

- Enable **color** in git, so the outputs will be easier to read:

```
git config --global color.ui auto
```

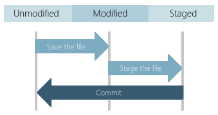
- Check all the settings:

```
git config --list
```

8. Git workflow

In a Git repository a file can be in one of these three states:

- Modified: When you modify files in your working directory
- Staged: You stage the files, adding a snapshot of them to the staging area
- Commited: You do a commit that stores snapshots permanently to your Git directory



9. Git basic operations (I)

- To create a new Git project:

  - `git init`
  - `git clone /path/to/repo`
- See state of the repository:
  - `git status`
- Add new files to staging area:
  - `git add <filename>`
  - `git add *`
- Commit changes:
  - `git commit -m "Commit message"`

10. Git basic operations (II)

- To see the history of a repo:
  - `git log`
  - `git log -n <number>`
  - `git log --stat`
  - `git log --pretty=oneline`
- See changes in a commit:
  - `git show <commitID>`
- or compare two commits:
  - `git diff commit1ID commit2ID`
- Revert to previous state:
  - `git checkout <commitID>`
- or return to current state:
  - `git checkout master`

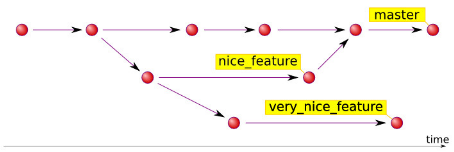
11. Exercise I: Basic commands in Git

1. Create a new repository
2. Add some files
3. Stage them
4. Modify one file
5. Check state of the repo
6. Commit changes
7. Repeat the process for other files
8. Go back to a previous state.
9. Inspect the directory
    - Which files were added by git?

12. Branching

Branches are used to develop features isolated from each other:

- The **master** branch is the "default" branch when you create a repository.
- We use other branches for development and merge them back to the master branch upon completition.



13. Operations

- To list all branches:
    - `git branch`
- To create a new branch:
    - `git branch <new_branch_name>`

- To switch to a different branch:
  - `git checkout <other_branch_name>`
- To create and swith to a new branch:
  - `git checkout -b <new_branch_name>`
- To delete a branch:
  - `git branch -d <branch_name>`
- To merge branches into the master:
  - `git checkout master`
  - `git merge <branch_a>`

14. Problems with branches

Sometimes auto-merge is not possible and the result are **conflicts**.

- Master and another branch modify the same file.

So we are responsible to merge those conflicts manually by editing the files shown by git.

- Editor of choice.
- **Tip**: Before mergin changes, preview changes.
  - `git diff`

15. Exercise II: Branching in Git

- Create a new branch
- Create a new file in this branch
  - Add some content
  - Commit it
- Navigate between branches and see the state of both from your text editor
- Generate a conflict by modifying the same file in both branches

- Merge branch into master

16. Working with remote repositories

- A remote URL is Git’s fancy way of saying "the place where your code is stored".
- That URL is going to be our repository on GitHub, but it could be another user’s fork, or even on a completely different server.
- Two types of URL addresses:
    - An HTTPS URL like `https://github.com/user/repo.git`
    - An SSH URL, like `git@github.com:user/repo.git`

Git associates a remote URL with a name, and your default remote is usually called `origin`.

17. Origin

- By convention the name it recives the new remote repository(GitHub).
- It could be multimple remote repositories.

When cloning a repository for the first time:

- It is the default name given to the original remote repository that you clone. It is where you want to pull and push changes.

18. Operations

- To add a remote:
    - `git remote add origin https://github.com/yhabib/gitTest.git`
- To send local changes from master to the remote:
    - `git push -u origin master`
- or from a particular branch:
    - `git push origin <branch_name>`
- To fetch remote changes:

  - `git pull`
- For a remote branch we first need to check it:
  - `git fetch`
  - `git checkout <branch_name>`

19. Markdown

- Adding `readme.md` files in a repository makes them look more profesional
- Markup lenguage used by GitHub
- Very easy to learn and use
- Cheatsheet[3]

20. Exercise III: Advance Git

- Create a new repository in GitHub
- Push your local repo to this remote
- Push as well branches
- Create a readme.md
  - With title and a list of elements
- Check all the information that GitHub offers about the repository

21. Exercise IV: Working on a team

1. Each member in the team should have a GitHub account.
2. A GitHub repository has to be created by a member of the group.
3. Inside the repository:
   - Settings
   - Collaborators
   - Add collaborators
4. Share url → slack sub-group

[3] https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet

5. Rest members in the team should clone the repo into their respective machines.
6. Play modifying files, commiting and pushing.

22. Feature Branch Workflow

- All feature development should take place in a dedicated branch instead of the `master` branch.
    - The `master` branch will never contain broken code
- `Pull requests` allows to initiate discussions around a branch
    - Code review by team members
    - Easy way for a developer to ask for help
    - Repository owner integrates the branch into the master by accepting the `pull request`.

23. Personal recommendations

- How to organize Propulsion Academy code
    - Projects based
    - Course based
- GitPages, easy way to create a web page for you or your page
    - Personal Page[4]
    - Project Page[5]
- GitHub daily contributions
    - Important for job searching
- Visual Studio Code[6]

[4] https://mapageka21.github.io/
[5] https://yhabib.github.io/JavaScript30/
[6] https://code.visualstudio.com/