UPskill

Version Control

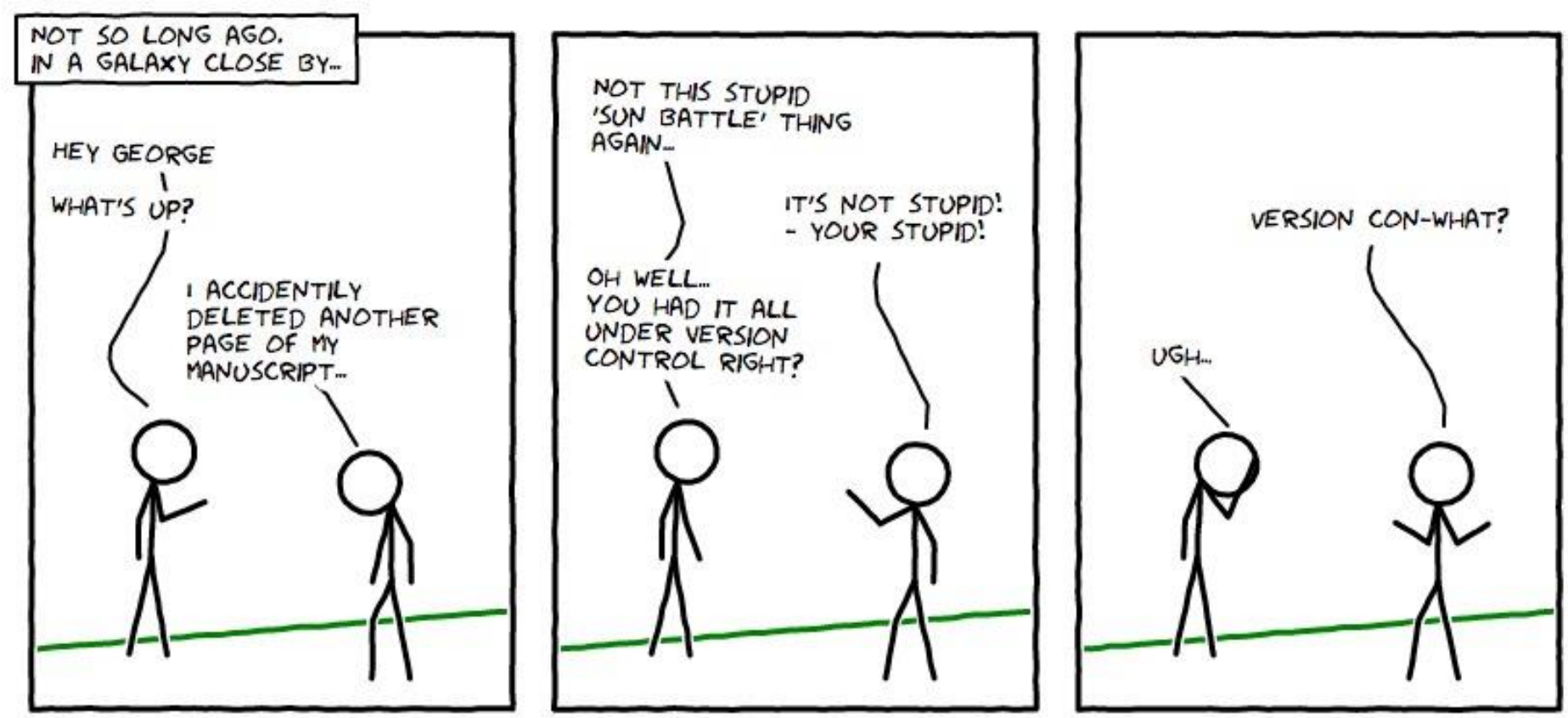
Principios de Desenvolvimento de Software

P.PORTO

isep Instituto Superior de Engenharia do Porto

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
Instituto Superior de Engenharia do Porto

Version Control



What is version control?

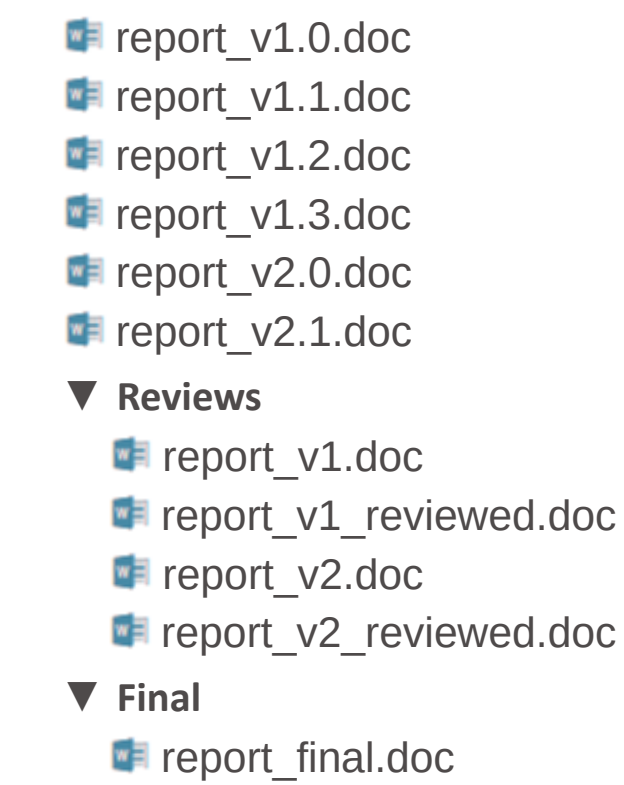
- System for recording and managing changes made to files and folders
- Used to manage source code
  - However, it is also well suited to tracking changes to any kind of file which contains mostly text.
- Used by a lone developer or as a means for many people to share and collaborate on projects efficiently and safely

What is version control?

- You probably already use a version control system, even without realizing it…
- This feature is built-in many modern editors such as:
  - Microsoft Word
  - Apple Pages
- Dropbox maintains a full history of all the files you have edited or deleted on the last month

What is version control?

- You have almost certainly employed your own simple form of a version control system in the past



Why you should use it?

- Reproducibility
  - You should be able to replicate every figure you have ever published, even if you have significantly developed your codes and tools since;
- Recoverability
  - Bring back that snippet you accidentally deleted;
- Experimentability
  - Try different approaches and simply disregard them if you don't like it;
- Collaborability
  - Concurrently work on a project with a collaborator and then automatically merge all of your changes together.

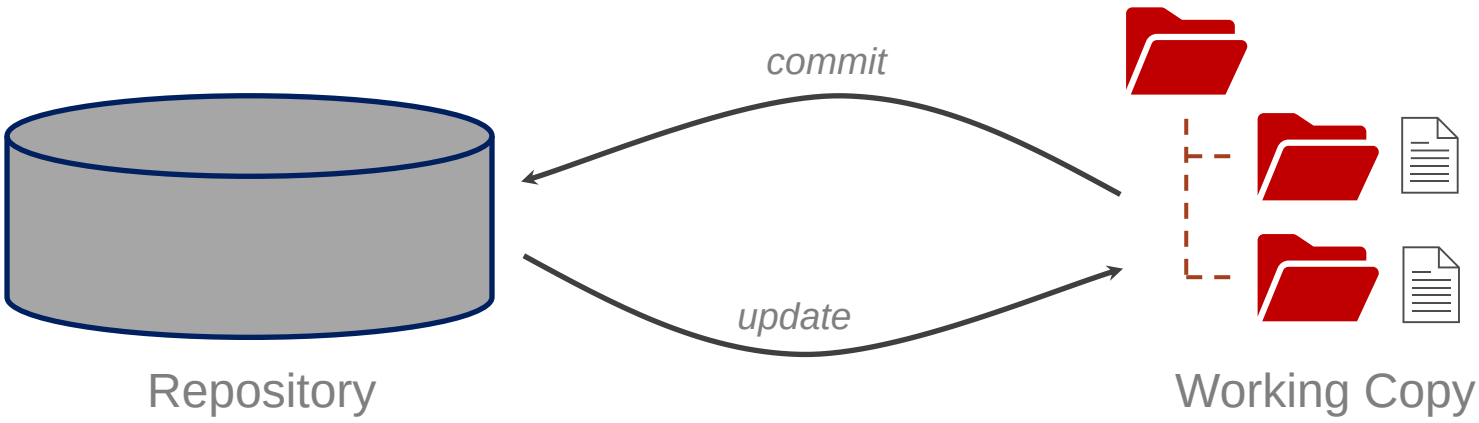
Why you should use it?

"In practice, everything that has been created manually should be put in version control, including programs, original field observations, and the source files for papers."

– Best Practices for Scientific Computing; Wilson et al. 2012 arXiv:1210.0530

Repositories and Working Copies

- Version control uses:
  - a Repository
    - Database of changes
  - a Working Copy (also colled *checkout*)
    - Personal copy of the project files



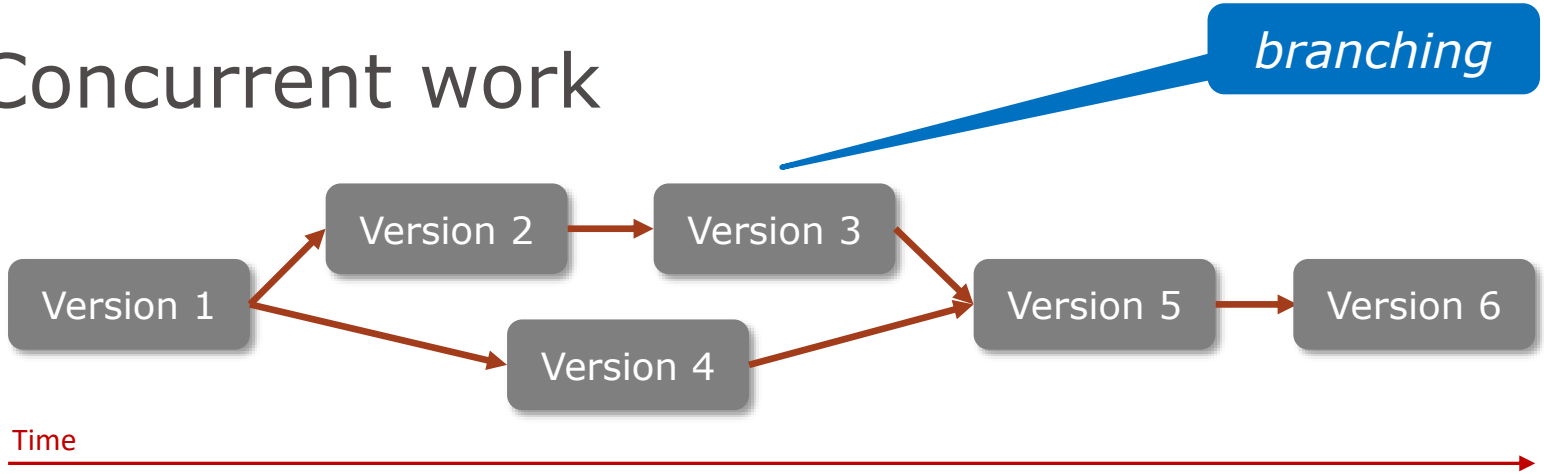
Repository

- Database of all the edits to, and/or historical versions (snapshots) of, your project.
- The repository can contain edits that have not yet been applied to your working copy.
- You can update your working copy to incorporate any new edits or versions that have been added to the repository since the last time you updated.

Repository - scenarios

- A linear history

- Concurrent work

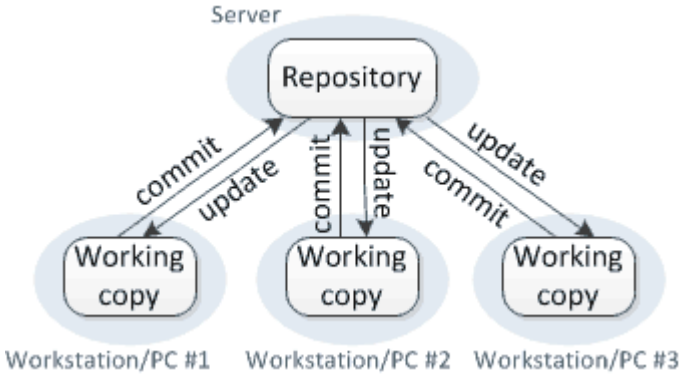


Version Control types

- Centralized
  - Slower
  - Easier to understand
  - Used by: *Subversion (SVN)*
- Distributed
  - Runs faster
  - Less prone to errors
  - Complex to understand
  - Used by: *Git*, *Mercurial*

Centralized version control

- Each user gets his own working copy, but there is just one central repository;
- As soon as you commit, it is possible for your co-workers to update and to see your changes.
- For others to see your changes, 2 things must happen:
  - You *commit*
  - They *update*



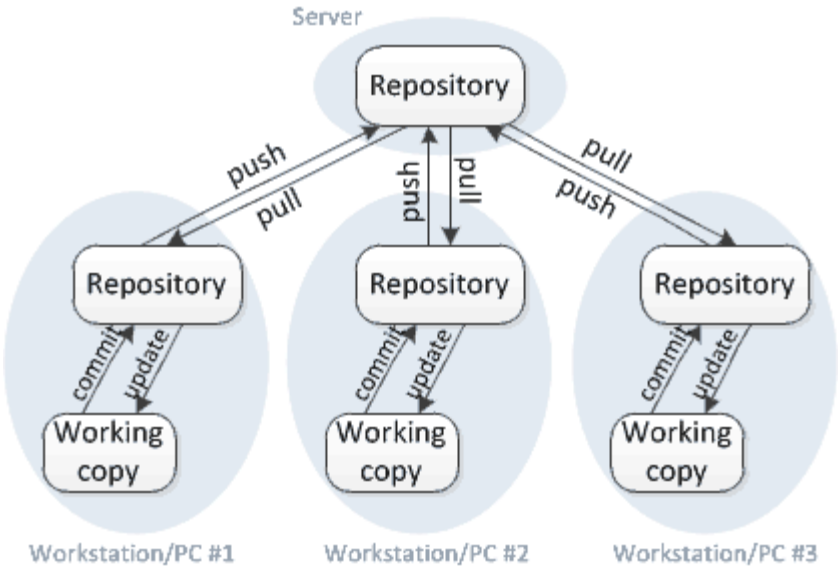
https://homes.cs.washington.edu/~mernst/advice/version-control.html

Distributed version control

- Each user gets his or her own repository and working copy.
- After you commit, others have no access to your changes until you *push* your changes to the central repository.
- When you update, you do not get others' changes unless you have first *pulled* those changes into your repository.

Distributed version control

- For others to see your changes, 4 things must happen:
  - You *commit*
  - You *push*
  - They *pull*
  - They *update*



https://homes.cs.washington.edu/~mernst/advice/version-control.html

Conflicts

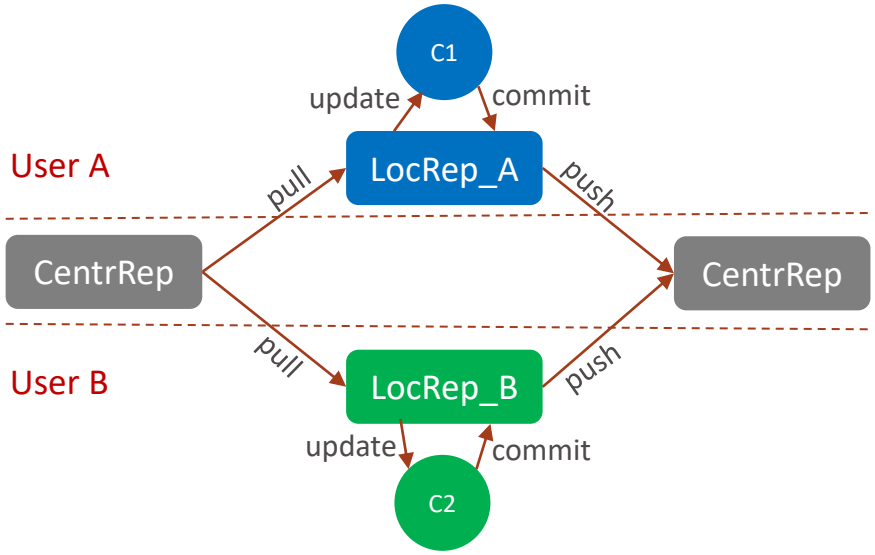
- A version control system lets multiple users simultaneously edit their own copies of a project.
- Usually, the version control system is able to merge simultaneous changes by two different users:
  - for each line, the final version is:
    - the original version if neither user edited it;
    - is the edited version if one of the users edited it.

Conflicts

- A conflict occurs when two different users make simultaneous, different changes to the same line of a file.
- In this case, the version control system cannot automatically decide which of the two edits to use (or a combination of them, or neither!).
- Manual intervention is required to resolve the conflict.

Conflicts

- *Change1* (*C1*) and *Change2* (*C2*) are considered simultaneous if:
  - User A makes *C1* before User A does an update that brings *C2* into User A's working copy;
  - User B makes *C2* before User B does an update that brings *C1* into User B's working copy.



Conflicts

- Merge operation combines simultaneous edits by different users.
- Sometimes merge completes automatically, but if there is a conflict, merge requests help from the user by running a merge tool.
- In centralized version control, merging happens implicitly every time you do update

Version control best practices

- Use a descriptive commit message
  - Indicates the purpose of the change;
  - Allow to look for changes related to a concept;
- Make each commit a logical unit
  - Each commit should have a single purpose and should completely implement that purpose.
- Avoid indiscriminate commits
  - Empty commits (with no explicit files supplied) commit every changed file.

Version control best practices

- Incorporate others' changes frequently
- Share your changes frequently
- Don’t commit generated files
  - Compiled binary files (such *.o* or *.class*)
  - Generated reports (*pdf*, *xml*, *txt*, ..)
  - Database files
  - IDE configuration files
- Don’t commit files that put the application in an unstable version.

Git

Principios de Desenvolvimento de Software

Introducing … Git!

"Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is easy to learn and has a tiny footprint with lightning fast performance. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient staging areas, and multiple workflows."

https://git-scm.com/

Git – First steps

- Install git on your system
  - https://git-scm.com/
- Git configuration
  - Several configuration files

Scope	Location	Filename
System	[path]/etc	gitconfig
Global	~/	.gitconfig
Local	[Repo]/.git	config

Git – First steps

- Configure user to sign commits

```
> git config --global user.name "Paulo Proenca"
> git config --global user.email prp@upskill.pt
```

- Configure Git editor

```
> git config --global core.editor nano
```

  - Opções: *vim*, *emacs*, *subl*, …

- Colorize Git messages

```
> git config --global color.ui true
```

Creating a repository

- Access the project folder or create a new one;
- Initialize the repository

```
> git init
```

- This command create a .git folder in the project folder
  - *.git* folder is where Git will store and manage the version control history of the project.

Creating a repository

- Create .gitignore file in the top of the project folder
- This file specifies intentionally untracked files that Git should ignore
  - A line starting with a # serves as a comment
  - Each line specifies a pattern

```
# excludes everything in directory foo
/foo/*
```

  - The prefix ! negates the pattern

```
# excludes everything except directory foo/bar
/foo/*
!/foo/bar
```

A typical *.gitignore* for .NET with VS

```
# compiled source
*.com
*.dll
*.suo
*.proj
# Directories
bin/
obj/
# Web publish log
*.Publish.xml
# Packages
*.rar
*.zip
# Logs and databases
*.log
*.sqlite
# OS generated files
.DS_Store?
```

A typical *.gitignore* for java netbeans

```
# NetBeans specific
nbproject/private/
build/
nbbuild/
dist/
nbdist/
nbactions.xml
nb-configuration.xml

# Class Files
*.class

# Package Files
*.jar
*.war
*.ear
```

Start adding files

- Add a new file (*file.js*) to the repository
- To check the status of the repository use:

```
> git status
```

- Should get:

```
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed) #
        file.js
nothing added to commit but untracked files present (use "git add" to track)
```

file.js is untracked

Start adding files

- To tell Git to start track the new file use:

```
> git add file.js
```

- Now:

```
> git status

 On branch master
 No commits yet

 Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   file.js
```

Committing changes

- Committing changes to the repository is the key step of version control.
- This is where we save a snapshot of the current state of all tracked files.
- To commit our current changes type:

```
> git commit –m "descriptive message"
```

- Provide a descriptive commit message

Staging modified files

- Edit a file (*file.js*)
- Now:

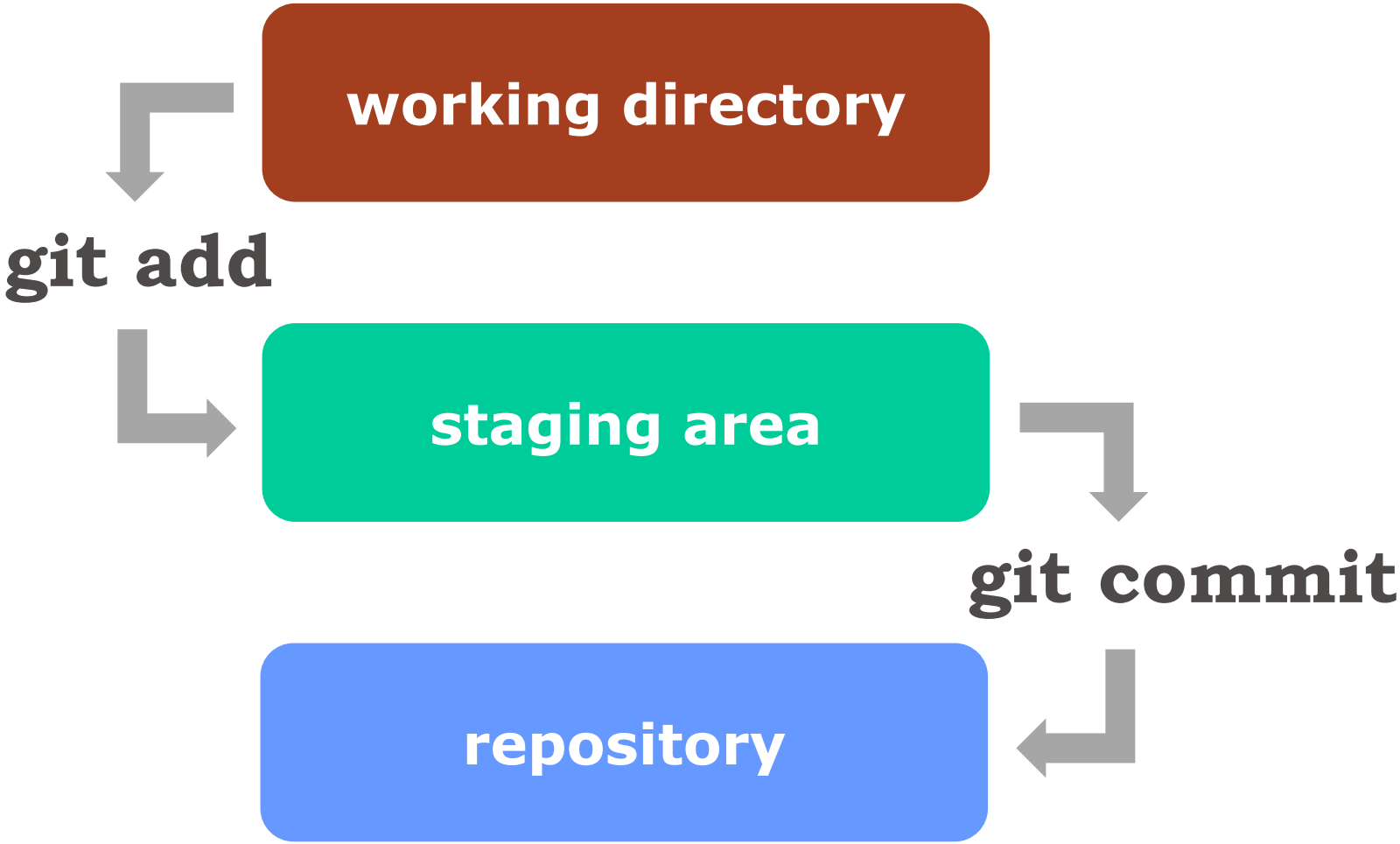
```
> git status

 On branch master
 Your branch is up to date with 'origin/master'.

 Changes to be committed:
  (use "git restore –staged <file> …" to unstage)
        new file: file.js
 Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
         modified:   file.js
```

Staging modified files

- What Git now tells us is that *file.js* falls under the category of "Changes not staged for commit".
- This means the file has changed since the last commit, however, we haven't told Git that we want to include these new changes in our next commit.
- To do that, we must "stage" the file using:

```
> git add file.js
```

Staging modified files



Dealing with mistakes

- If you make a typo in your commit message, or you forget to stage an important change before committing, you can easily amend your last commit using:

```
> git commit --amend
```

- Example:

```
> git add file.js
> git commit –m "Domain class file aded"
> git commit --amend
> git add fname.js
> git commit –m "Domain classes file and fname added"
```

typo

failure

Deleting and moving files

- To delete *file.js* from repository use:

```
> git rm file.js
```

  - This will both delete the file from the file system and stage this deletion action for your next commit.
- To stop tracking *file.js* (remove from repository) without deleting it from the file system use:

```
> git rm –cached file.js
```

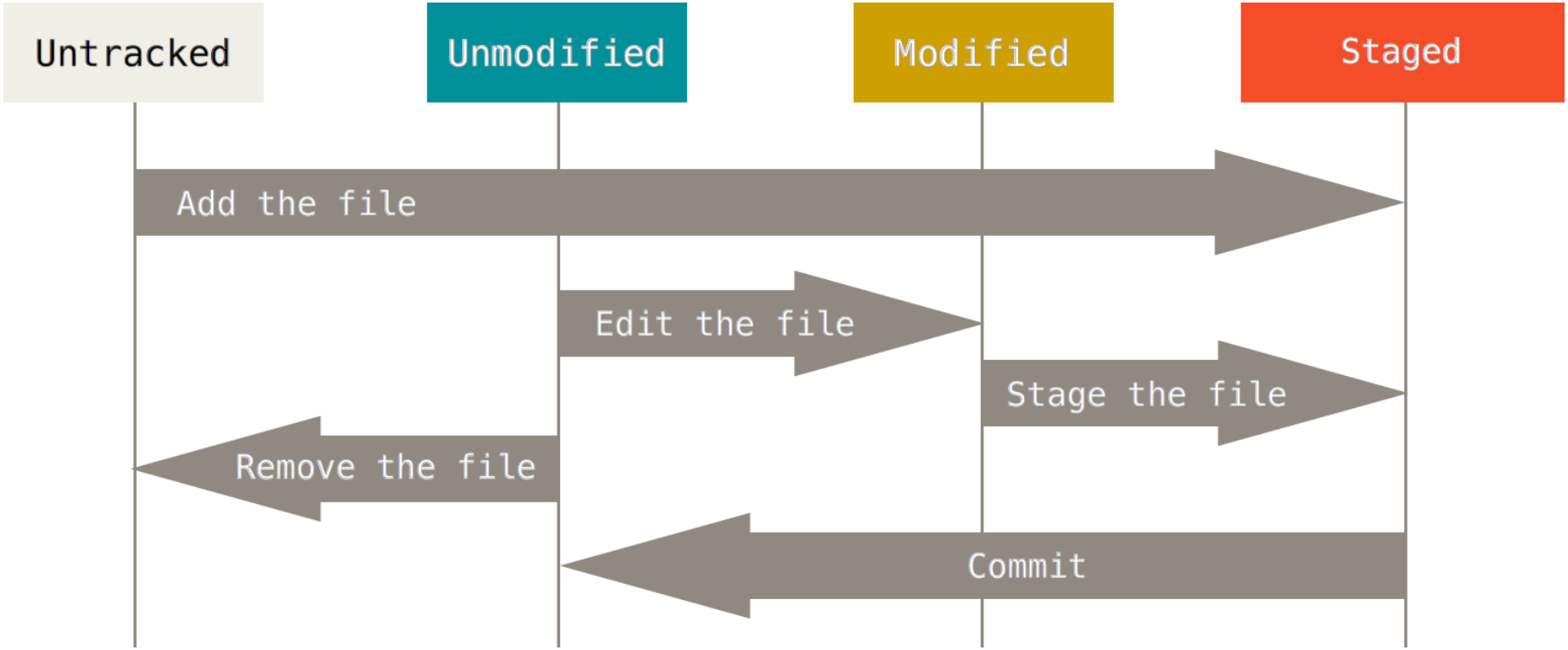
- To move or rename use:

```
> git mv <source> <destination>
```

Life cicle

- Each file can have one of four different states:
  - Untracked: It's not listed in the last commit
  - Unmodified: It hasn't changed since the last commit
  - Modified: It has changed since the last commit
  - Staged: The changes will be recorded in the next commit made

Life cicle



http://adamwilson.us/RDataScience/11_Git.html

The commit history

- To display the commit history use:

```
> git log

commit 5c9d9f3a8c997d006c197609a4355b979e784d53 (HEAD -> master)
Author: Paulo Proença <prp@upskil.pt>
Date:   Sat Nov 7 11:35:39 2020 +0000

    Domain classes file and fname added

commit 1d2f05f80b4291a68c4c6b53ca57e87006f8f987
Author: Paulo Proença < prp@upskil.pt >
Date:   Sat Nov 7 10:55:16 2020 +0000

    Inicial commit
```

The commit history

- There are a whole of host of flags to change the information and how it appears.

```
> git log --pretty=format:"%h %s <%an>" --graph

* 5c9d9f3 Domain classes file and fname added <Paulo Proença>
* 1d2f05f Inicial commit <Paulo Proença>
```

- To investigate all the different options use:

```
> git help log
```

The commit history

- Another useful way to visualize the history to is to look at a single file and see in which commit each line was last changed.

```
> git blame file.js

^1d2f05f (Paulo Proença     2020-11-07 10:55:16 +0000 1)
var express    = require('express');
5c9d9f3a (Paulo Proença     2020-11-07 11:35:39 +0000 2)
var bodyParser = require('body-parser');
00000000 (Not Committed Yet 2020-11-07 13:46:49 +0000 3)
d=4;
```

Comparing commits

- To compare commits to see how things have changed use *git diff* command.
- This command compares last unstaged changes with the commit referenced as argument.

```
> git diff <commit hash>
```

- If you don’t supply a commit hash it use the last commit

```
> git diff
```

Comparing commits

```
> git diff 5c9d9f3

diff --git a/file.js b/file.js
index 8ae5425..ad7e0a0 100644
--- a/file.js
+++ b/file.js
@@ -1,3 +1,3 @@
 var express    = require('express');
-a=1
\ No newline at end of file
+b=0;
\ No newline at end of file
diff --git a/fname.js b/fname.js
new file mode 100644
index 0000000..fd3c1fb
--- /dev/null
+++ b/fname.js
@@ -0,0 +1 @@
+var express    = require('express');
\ No newline at end of file
```

What is a branch?

- Branches allow to diverge from your current development and try something new without altering the history of your main work.
- For example, you could implement a new code feature whilst leaving the fully functional (hopefully working and tested) code intact for others to checkout.
- It is a fantastic way to test ideas, try new things and safely develop your repository.

Creating branches

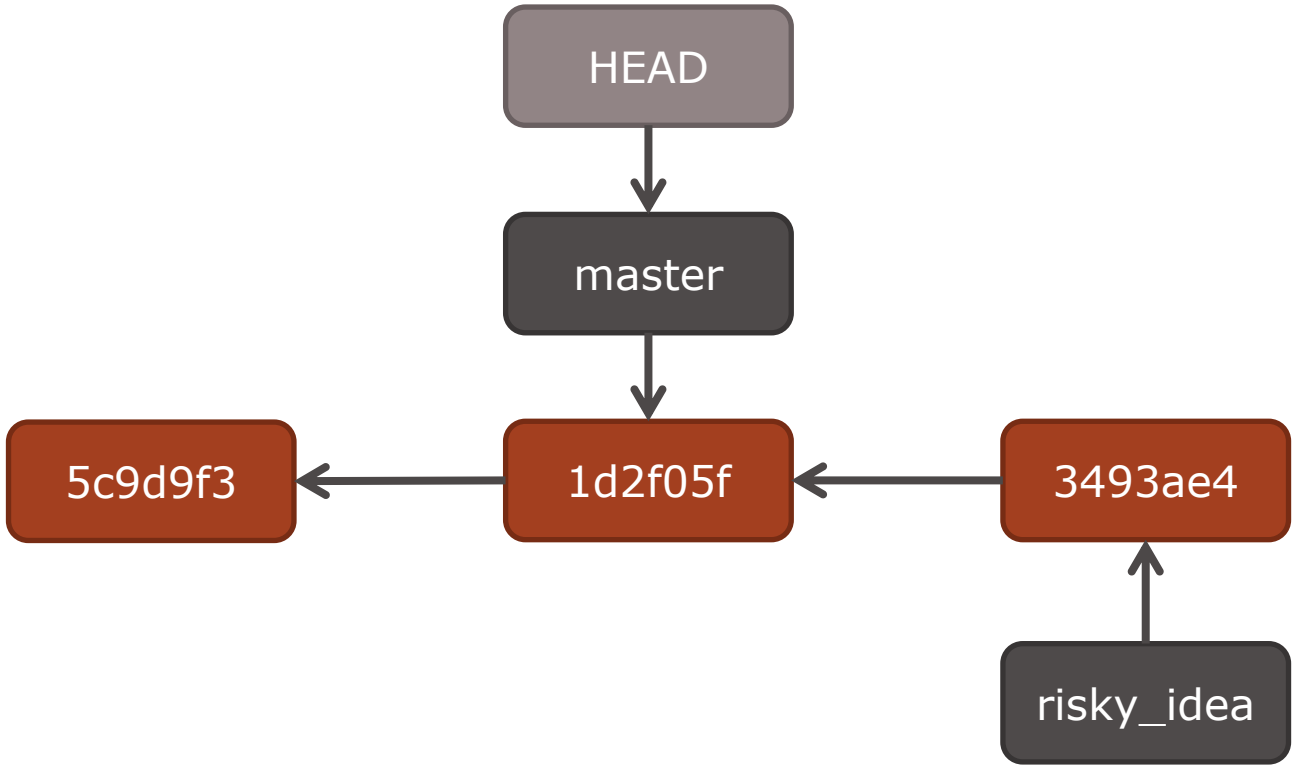
- A new repositories, by default, start on a branch called *master*
- To create a branch colled *risky_idea* use:

```
> git branch risky_idea
```

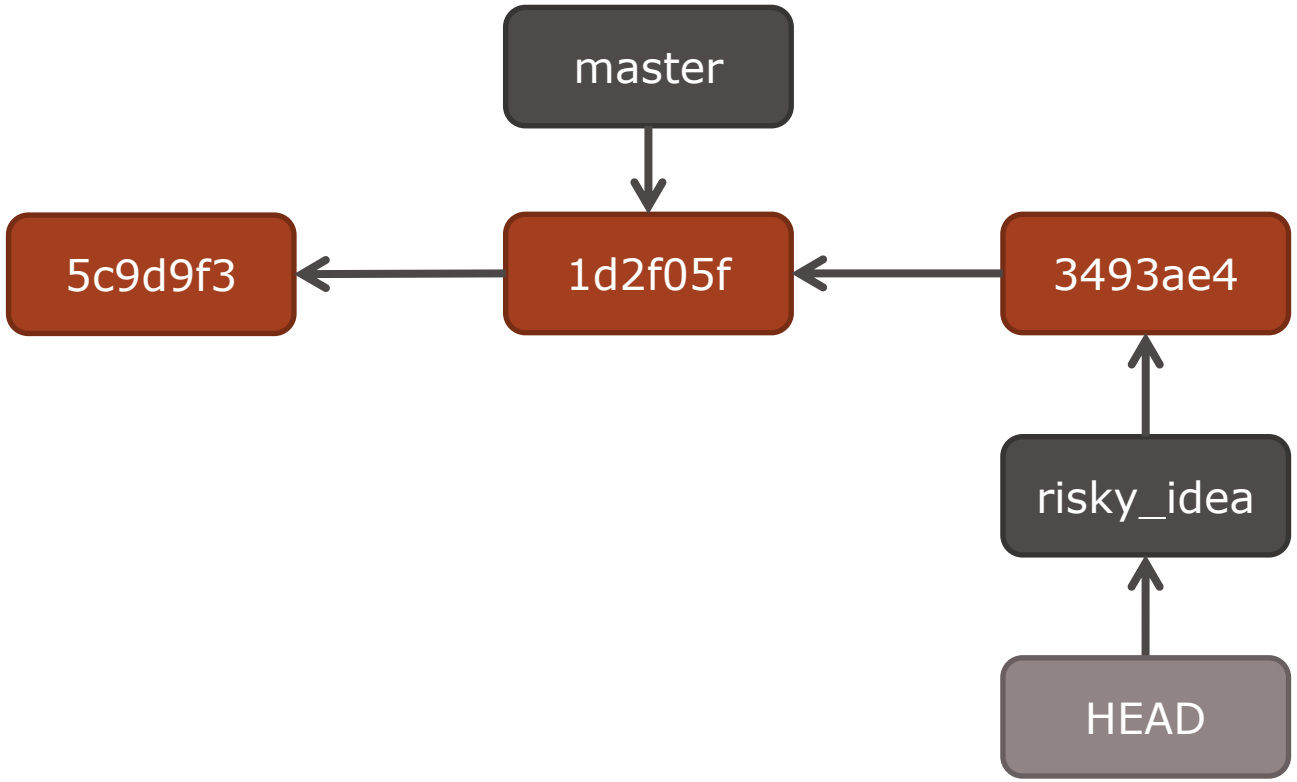
- To start working with new branch:

```
> git checkout risky_idea
Switched to branch 'Risky_idea'
```

How git checkout works



How git checkout works



Merging

- At some stage we will want fold our changes in the *risky_idea* branch back into the *master* branch.
- We do this by merging the *risky_idea* branch into *master*.

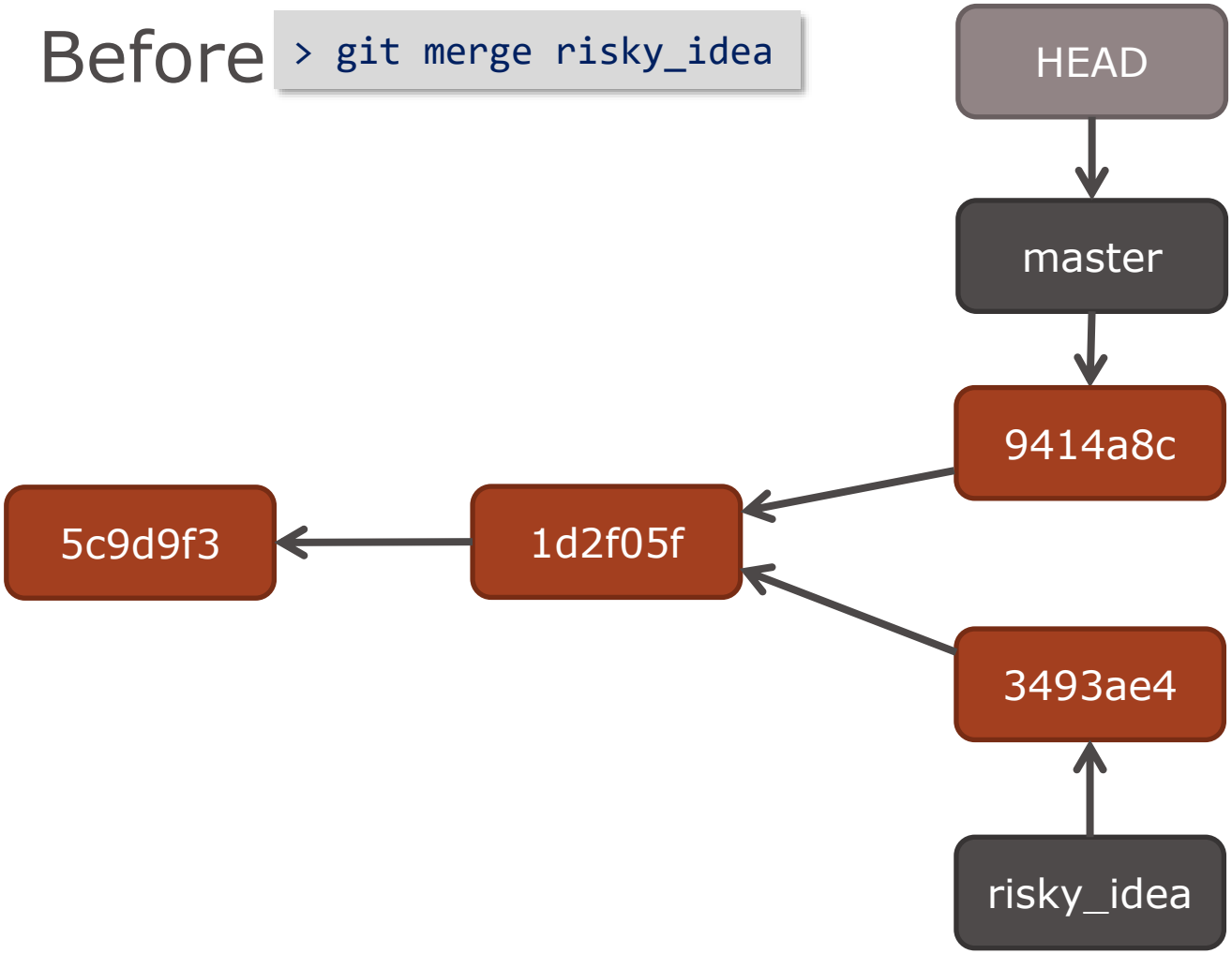
```
> git checkout master
> git merge risky_idea
```

- To delete a no longer needed branch use:

```
> git branch -d risky_idea
```

How git merge works

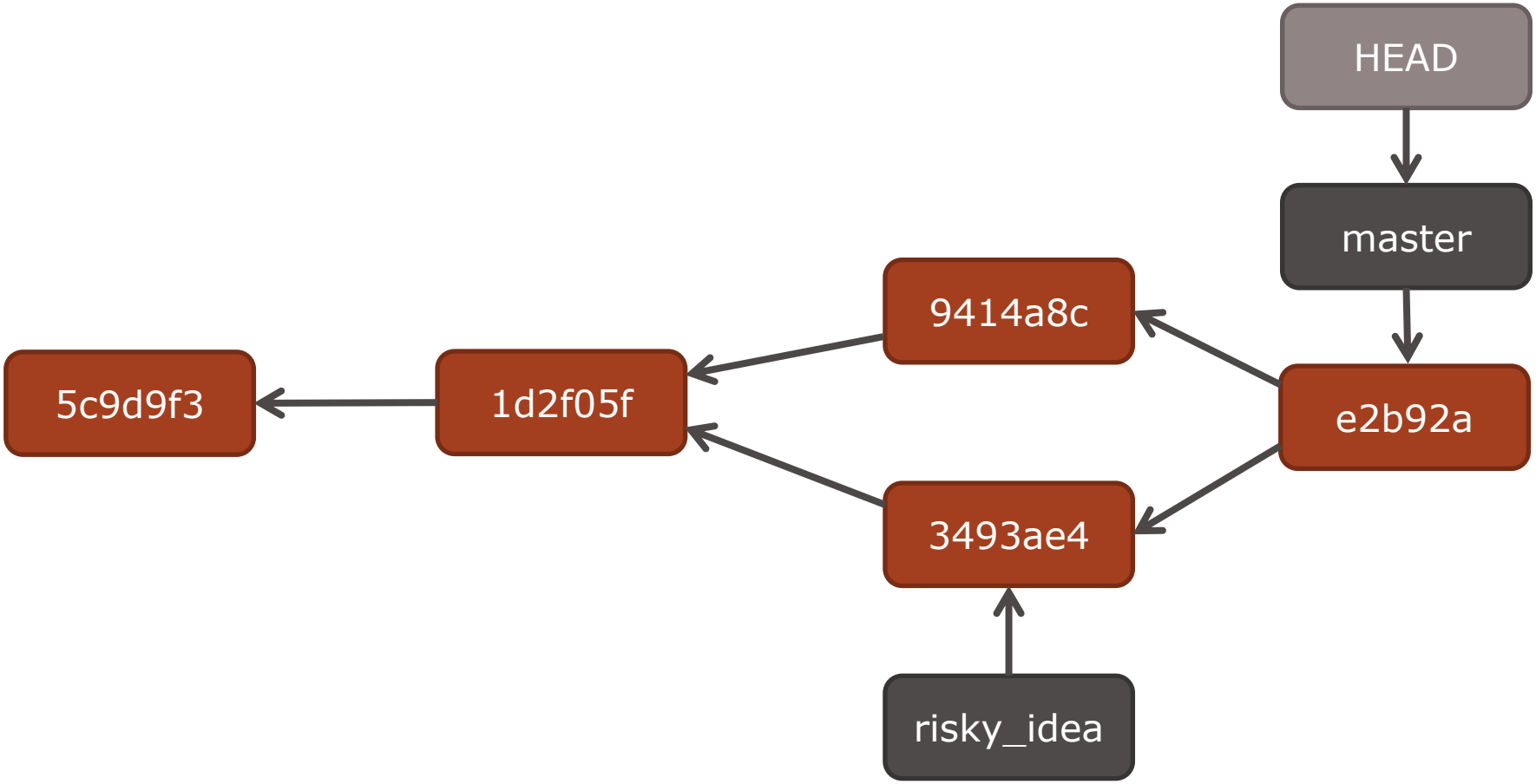
Before `> git merge risky_idea`



How git merge works

After

```
> git merge risky_idea
```



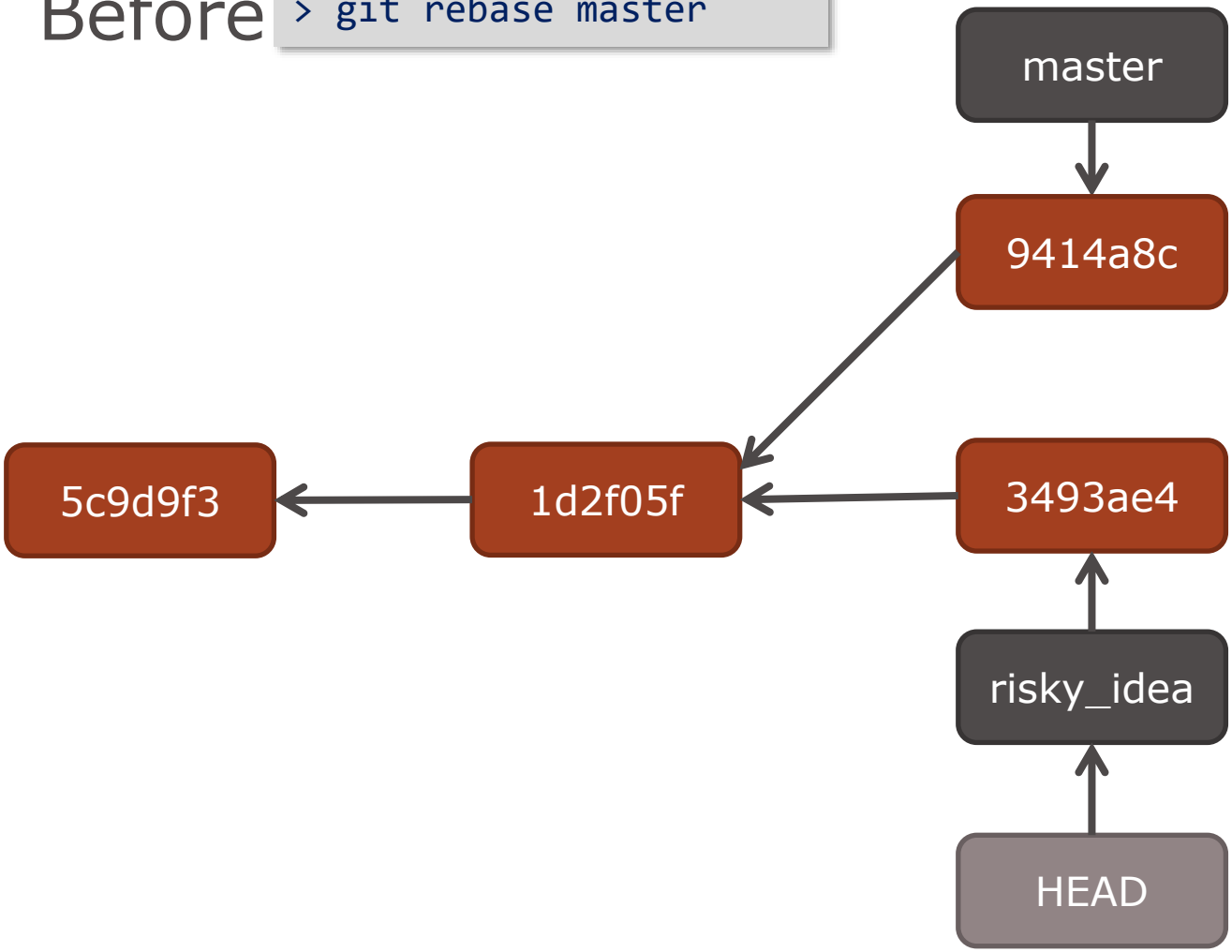
Rebasing

- In Git, there are two main ways to integrate changes from one branch into another: the merge and the rebase
- With the rebase command, you can take all the changes that were committed on one branch and replay them on a different branch

```
> git checkout risky_idea
Switched to branch 'master'
> git rebase master
Successfully rebased and updated refs/heads/master.
```

How git rebase works

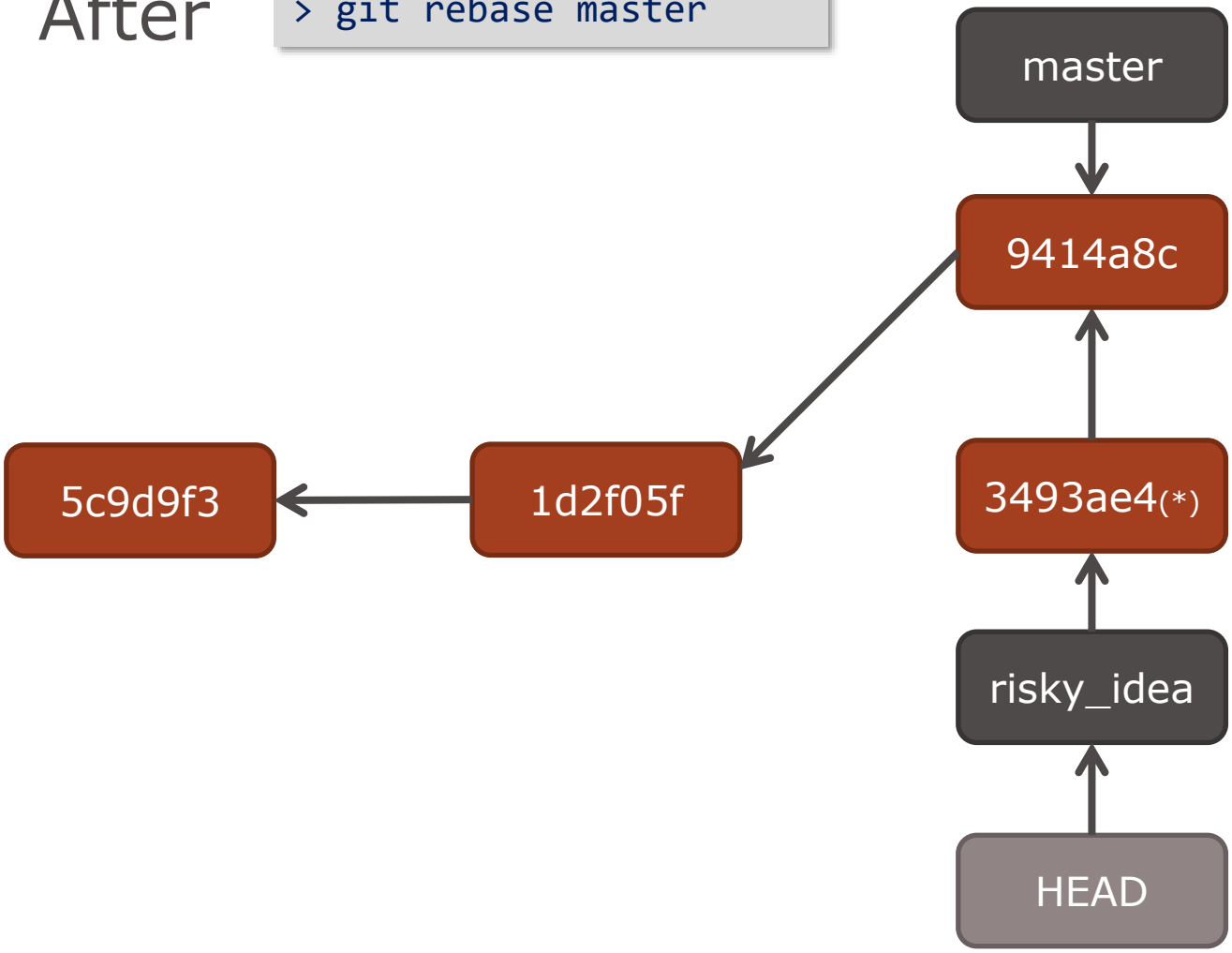
Before `> git rebase master`



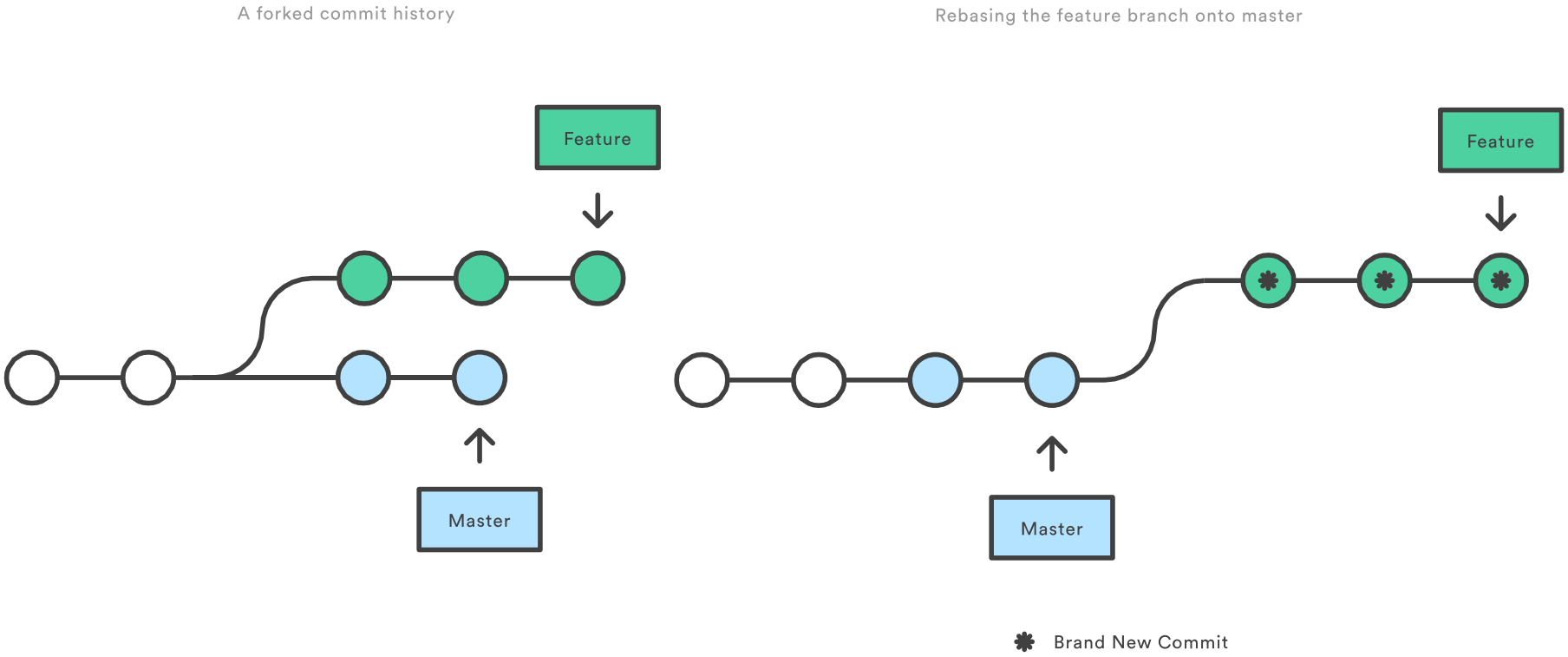
How git rebase works

After

```
> git rebase master
```



How git rebase works




```
> git rebase master
```

Online hosting

- Git uses a "distributed" model that allows everyone working on a project to have their own independent copy of the entire repository.
- To collaborate effectively though we need a central version of the code base which is used to unify everyones' efforts.
- Typically the best place for such a central repository is online

Online hosting

- There are several options for hosting git repositories online. However, there are two which stand out:
  - Bitbucket (https://bitbucket.org)
    - Unlimited free public repositories for small teams (5 users)
    - 1 GB storage
  - Github (https://github.com)
    - Unlimited free public or private repositories
    - Unlimited collaborators
    - 500 MB storage

Adding a remote

- To add a remote use *git remote add* command in the directory your repository is stored at.
- This command takes two arguments:
  - A remote name, for example, *origin*
  - A remote URL

```
> git remote add origin https://github.com/prpUpSkill/MyUPskillRepo.git

> git remote -v
origin  https://github.com/prpUpSkill/MyUPskillRepo.git (fetch)
origin  https://github.com/prpUpSkill/MyUPskillRepo.git (push)
```

Cloning a repository

- If you have the address (and correct permissions) for an online repository then you can grab your own copy using the *clone* command.

```
> git clone https://github.com/pproenca/MyProject.git
```

- Now you have your own copy of the repository and can do whatever you want with it.

Collaboration basic work-flow

- Make your changes in your own personal copy of the repo, ideally in a new branch.
- "Pull" the most recent version of the remote repo into your *master* branch.

```
> git pull
```

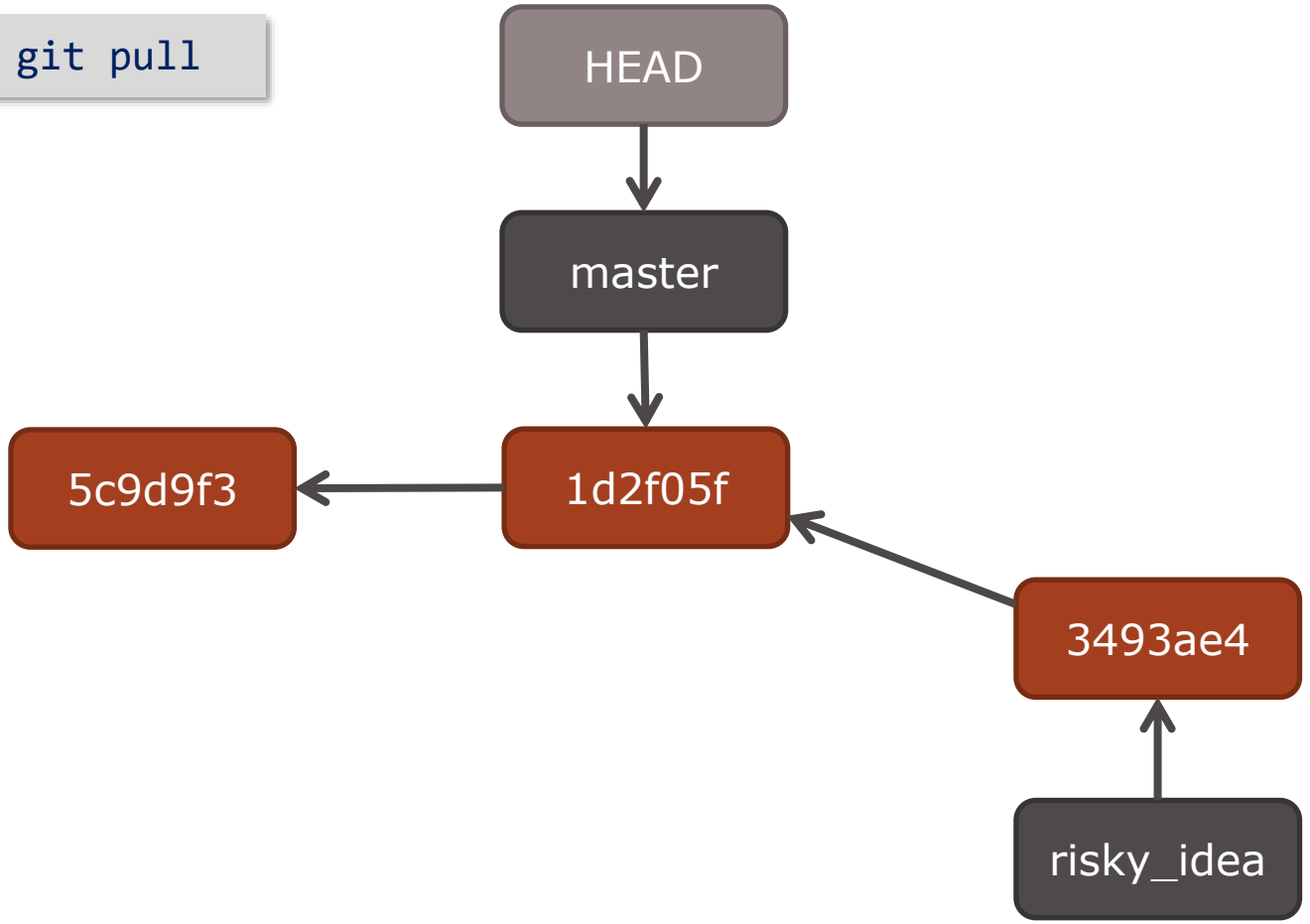
- Merge your changes from your new branch into *master*.
- Once any conflicts are resolved you can update the remote repo with your code.

```
> git push
```

How git pull works

Someone else pushed.

Before `> git pull`



How git pull works

Someone else pushed.

After `> git pull`

git-fetch command

- In Git there are two versions of the local repository:
  - One contains your code with its changes
  - The other mirrors the remote repository
- These two versions of the local repository support diff and merge commands
- *`git fetch`* can be used to download all commits from the remote repository without affecting the local code.
- Basically, *`git pull`* does a *`git fetch`* first followed by a *`git merge origin / master`*.

Git Clients

- There are several third-party tools for users looking for platform-specific experience
  - SourceTree (https://www.sourcetreeapp.com/)
  - TortoiseGit (https://tortoisegit.org/)
  - GitKraken (https://www.gitkraken.com/)
- Most IDE has Git support
  - Netbeans (https://netbeans.org/kb/docs/ide/git.html)
  - Visual Studio (https://devblogs.microsoft.com/visualstudio/improved-git-experience-in-visual-studio-2019/)
  - VS Code (https://code.visualstudio.com/docs/editor/versioncontrol)

Example

You

GitHub

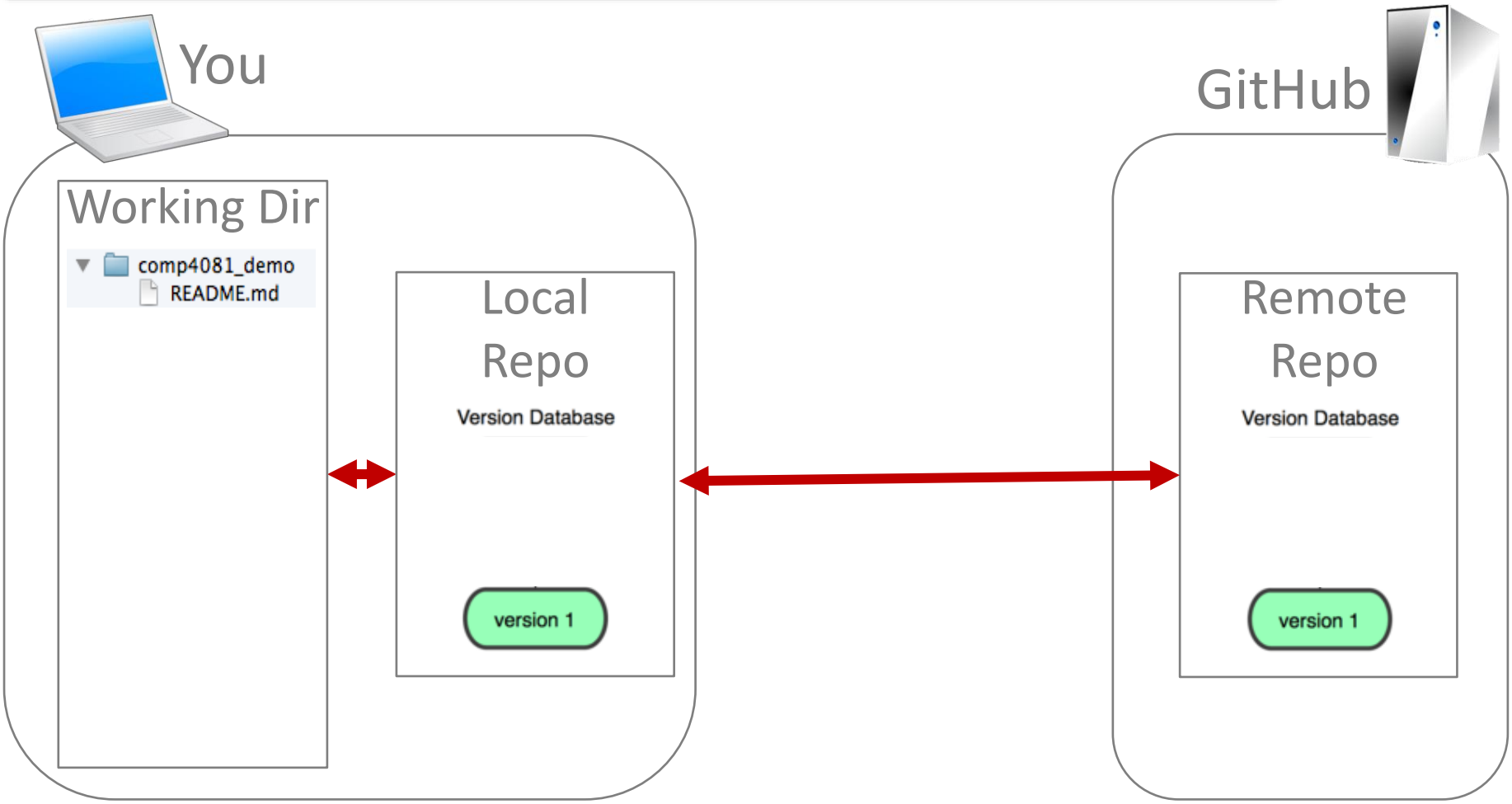
Remote Repo

Version Database

version 1

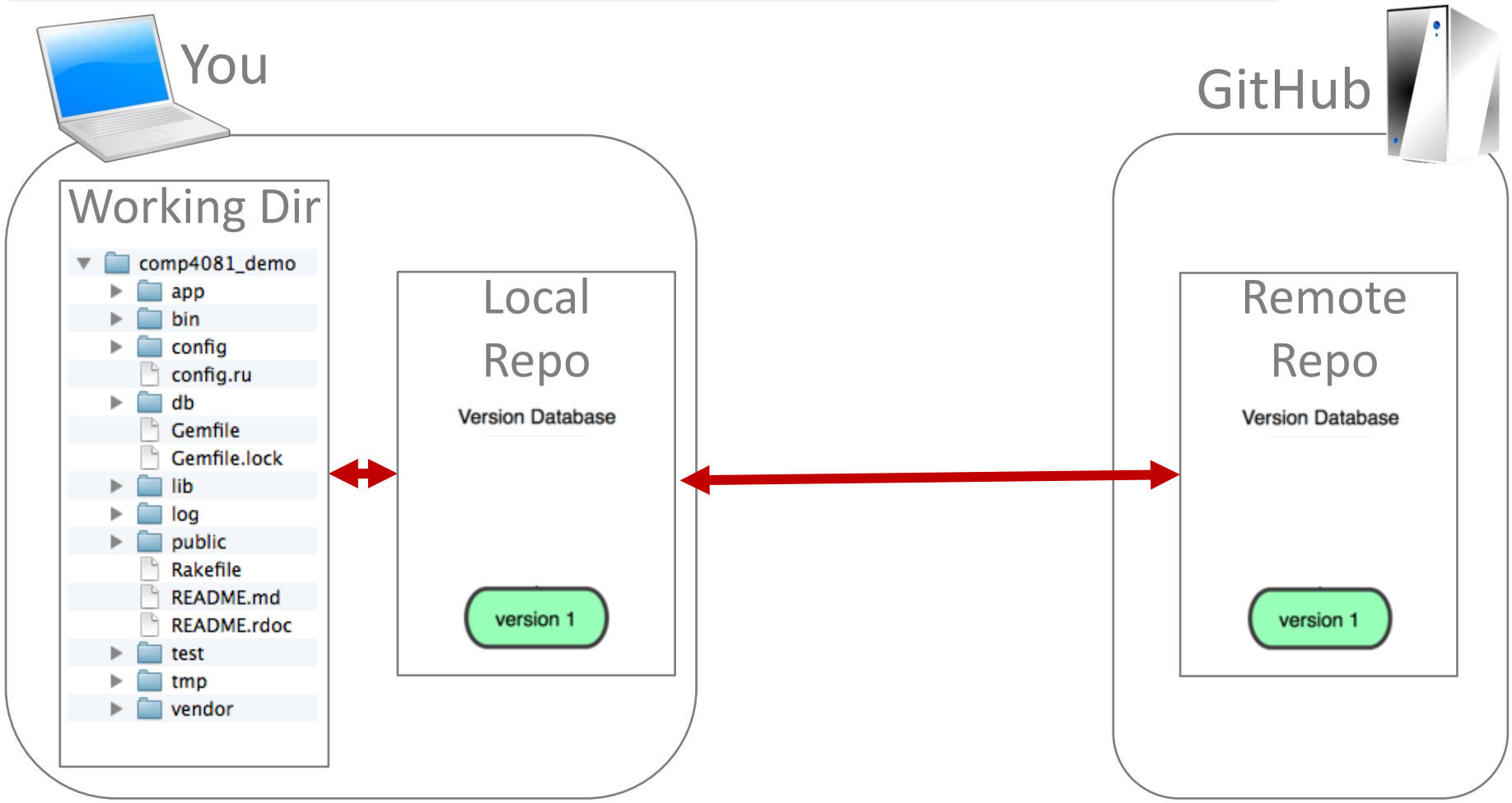
Example

```
> git clone https://github.com/pproenca/comp4081_demo.git
```



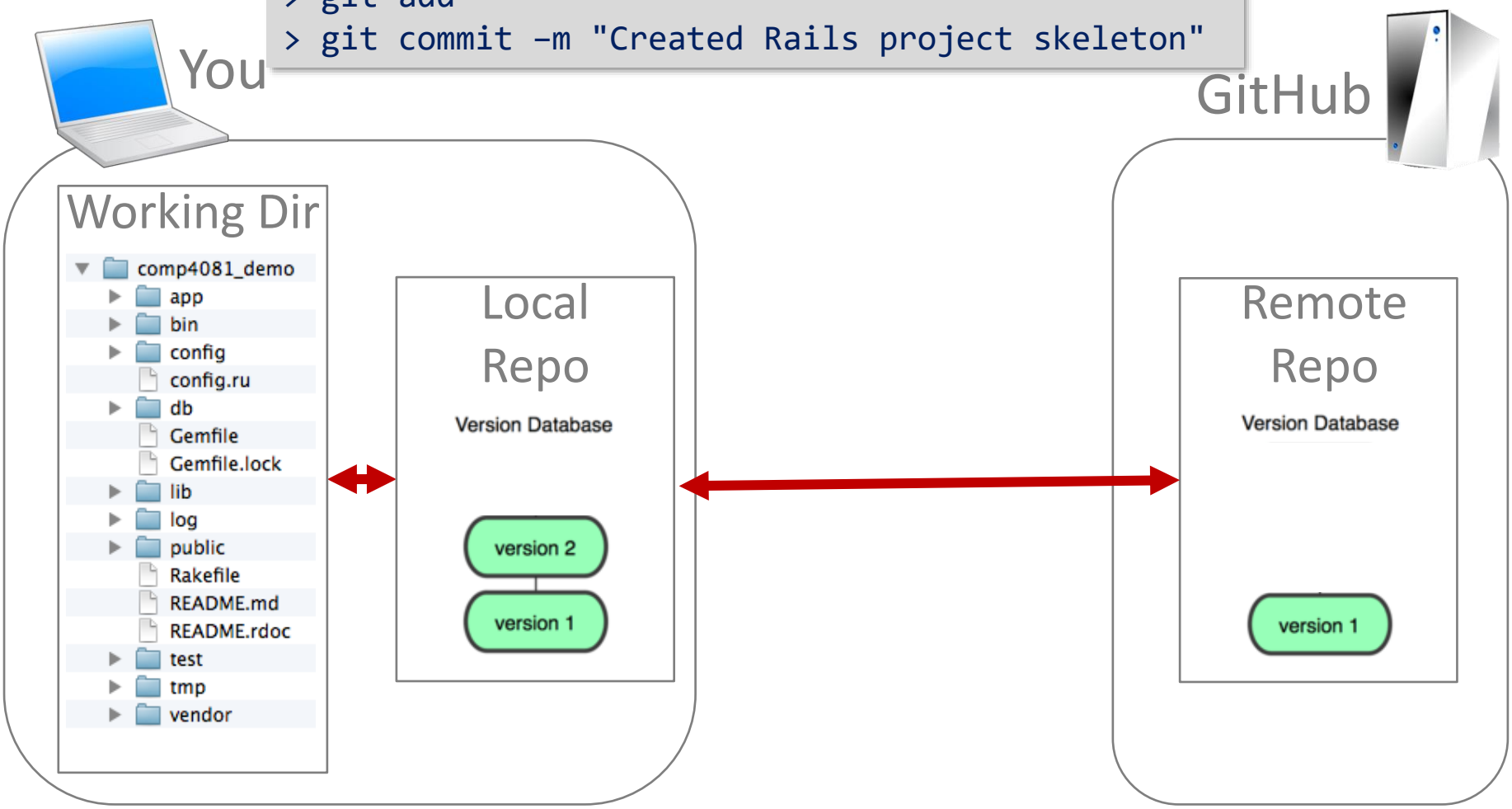
Example

```
> rails new comp4081_demo
```



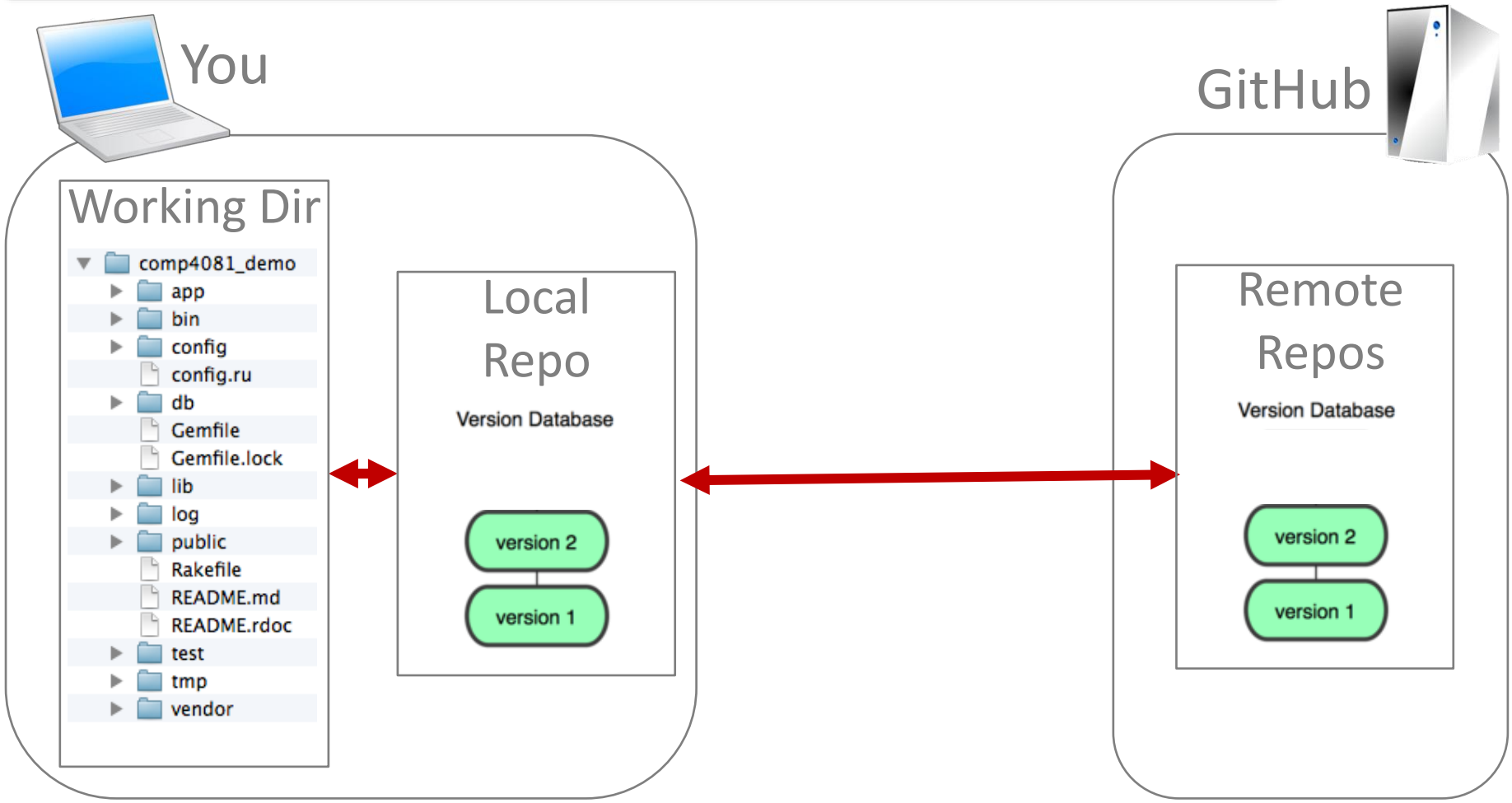
Example

```
> cd comp4081_demo
> git add *
> git commit –m "Created Rails project skeleton"
```



Example

```
> git push
```



Git conflicts

- Merge the branch

Hello.html

```
<html>
  <head>
<<<<<<< HEAD
     <link type="text/css" rel="stylesheet" media="all" href="style.css" />
=======
     <!-- no style -->
>>>>>>> master
  </head>
  <body>
     <h1>Hello,World! Life is great!</h1>
  </body>
</html>
```

Git conflicts

- Resolution of the conflict

Hello.html

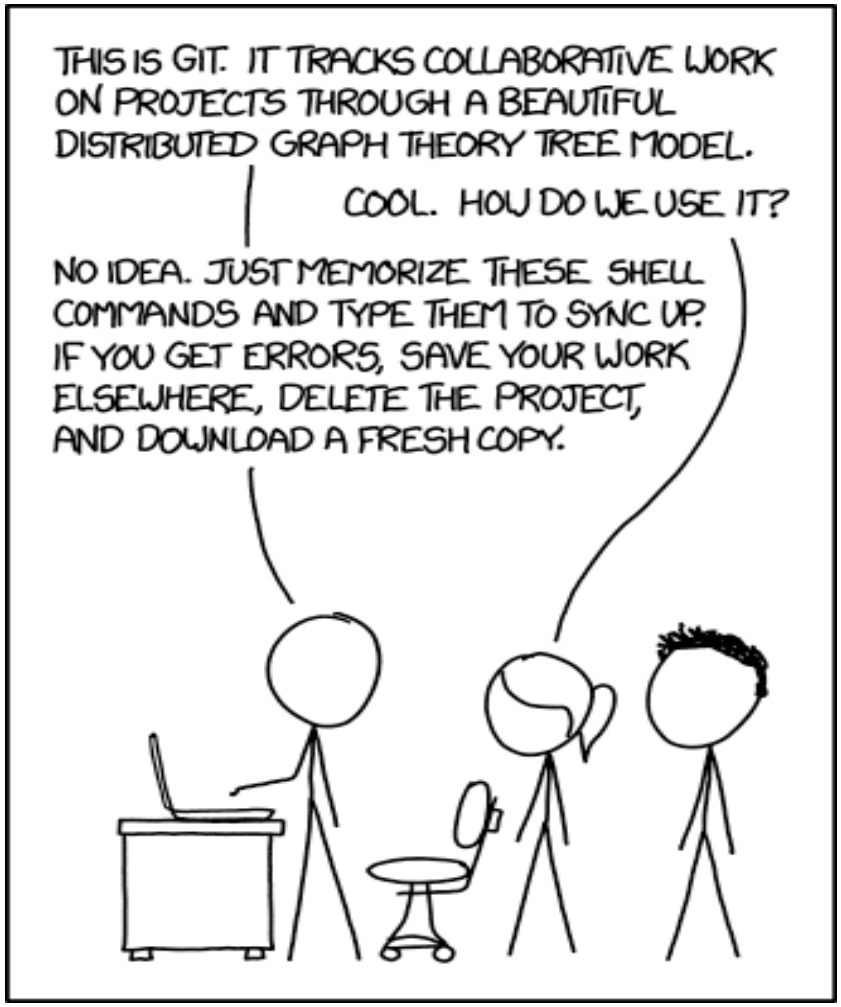
```
<html>
  <head>
     <link type="text/css" rel="stylesheet" media="all" href="style.css" />
</head>
  <body>
     <h1>Hello,World! Life is great!</h1>
  </body>
</html>
```

Git conflicts

- Make a commit of conflict resolution

```
> git add Hello.html
> git commit -m "Merged master fixed conflict."
Recorded resolution for 'Hello.html'.
[style 645c4e6] Merged master fixed conflict.
```

That's all … about Git!



GitHub

Principios de Desenvolvimento de Software

Introducing … GitHub!

"GitHub is a development platform inspired by the way you work. From open source to business, you can host and review code, manage projects, and build software alongside 50 million developers."

https://github.com/

GitHub– First steps

- Access https://github.com/
- Create an account
  - Email:@upskill.pt
- Verify email address
  - An email containing verification instructions will be sent to you registered email

GitHub– First steps

What do you want to do first?

Every developer needs to configure their environment, so let's get your GitHub experience optimized for you.

Start a new project

Start a new repository or bring over an existing repository to keep contributing to it.

Create a repository

Collaborate with your team

Improve the way your team works together and get access to more features with an organization.

Create an organization

Learn how to use GitHub

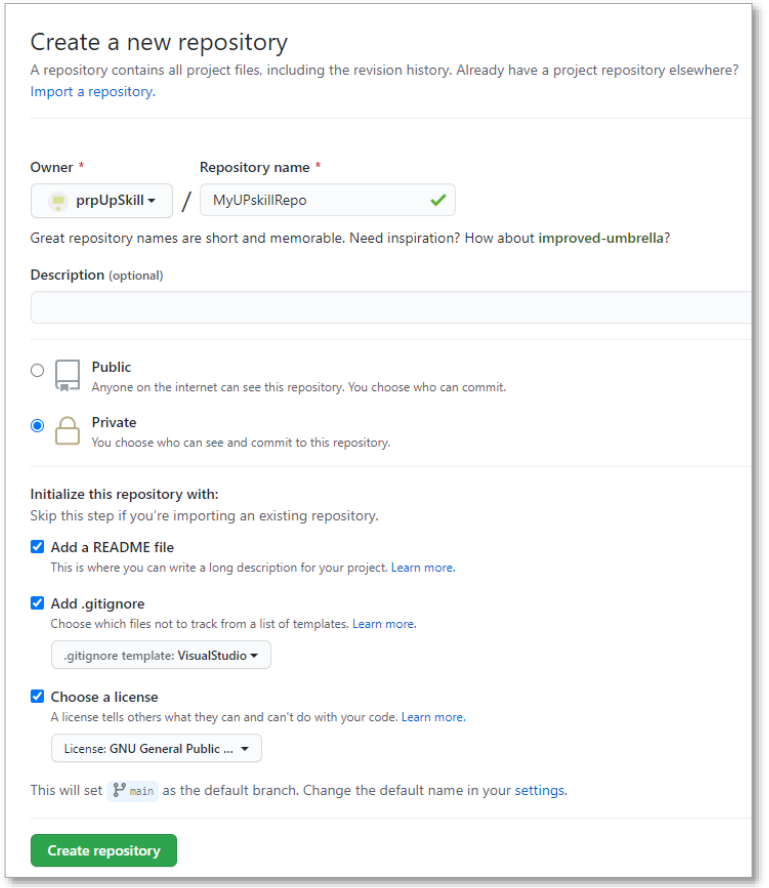
Get started with an "Introduction to GitHub" course in our Learning Lab.

Start Learning

Skip this for now >

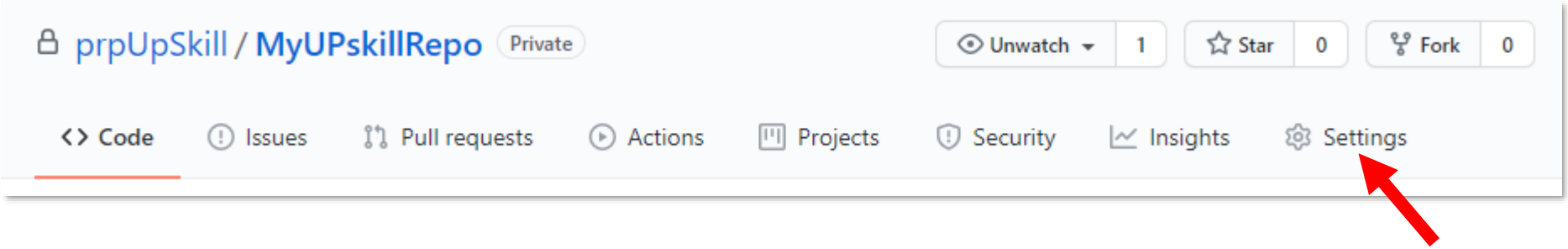
Create a new repository

- Choose a repository name
- Choose Private
- Initialize repository
  - Check “Add a README file”
  - Check “Add a .gitignore”
    - Template VisualStudio
  - Check “Choose a license”
    - GNU General Public …
- Click “Create repository”

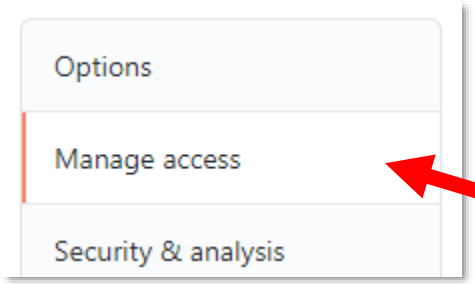


Add collaborators to the project

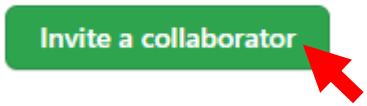
- Select Settings



- Choose Manage access

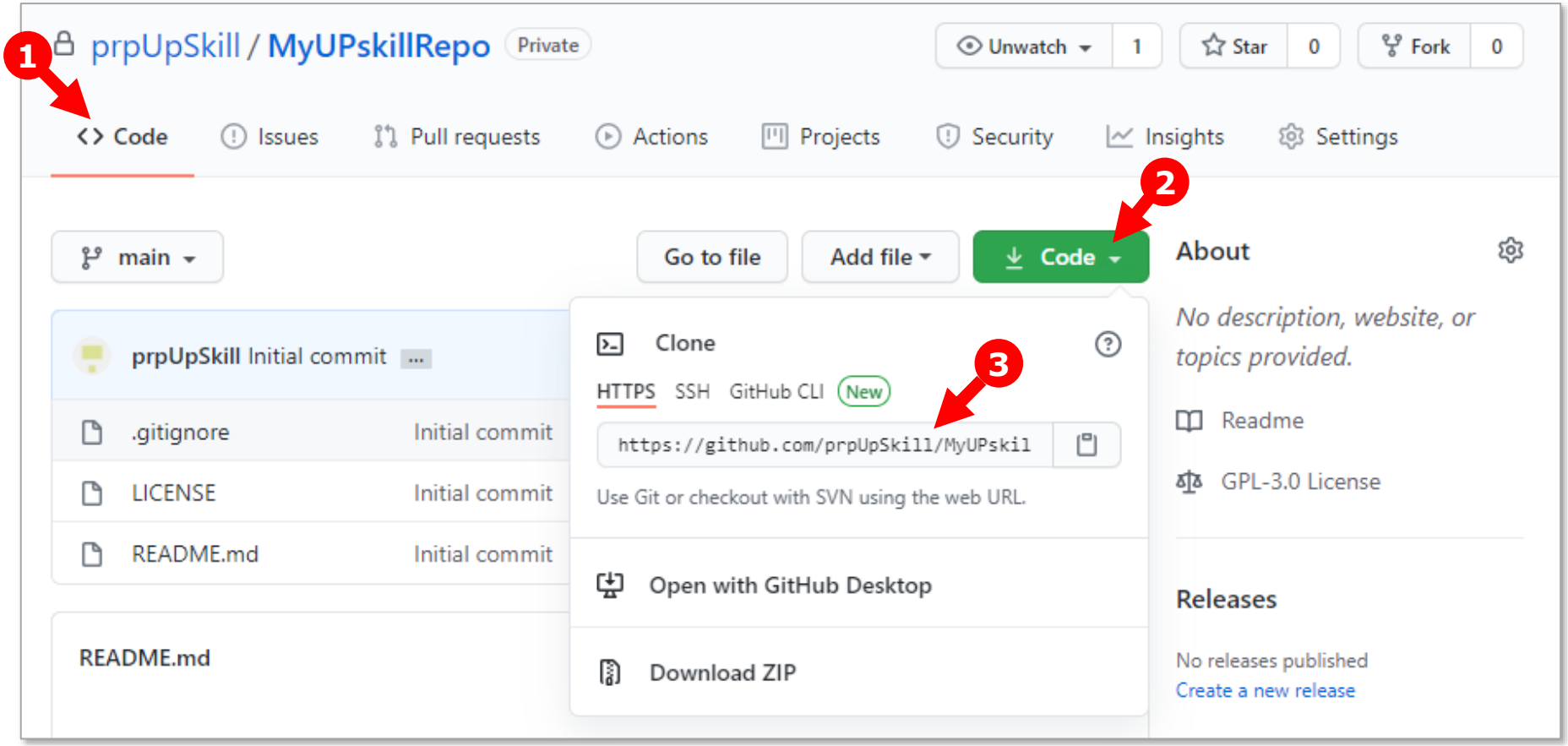


- Invite a collaborator

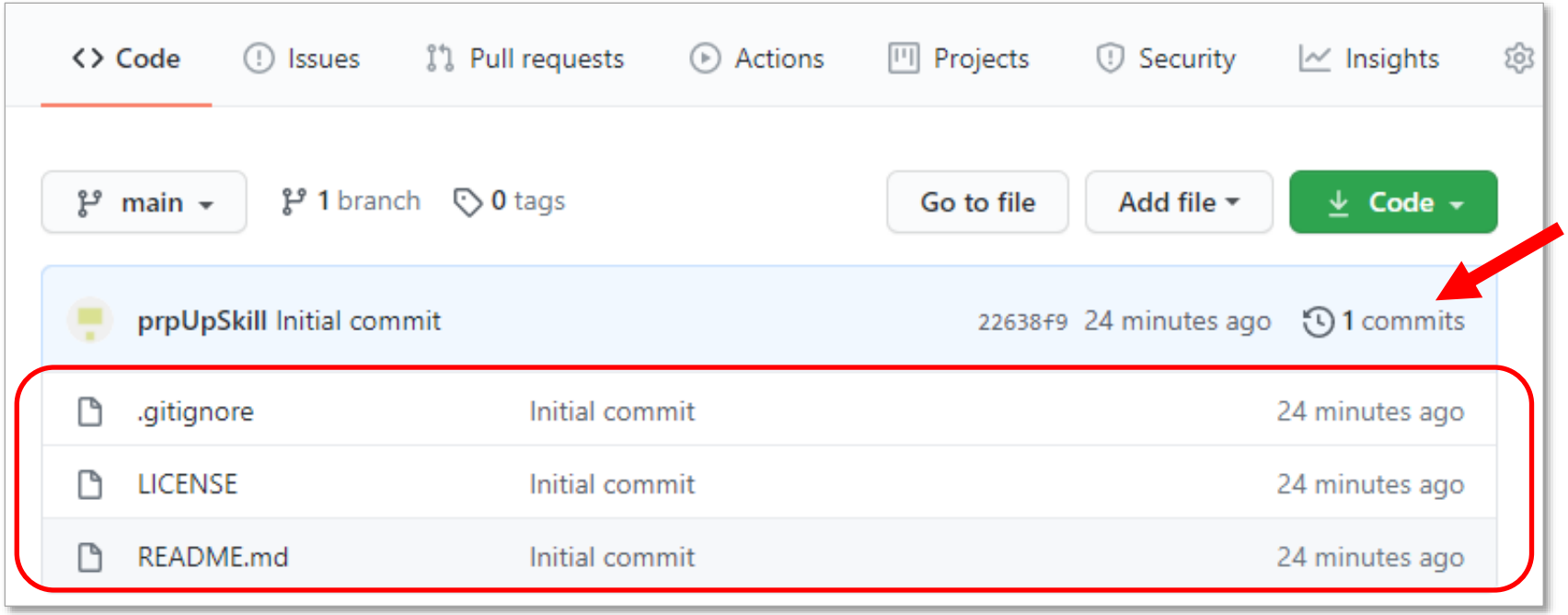


  - Search a collaborator by email (@upskill.pt)

Clone the repository



Code & commits



Exercise

Principios de Desenvolvimento de Software

Exercise

- Install and configure Git
- Register in GitHub with your @upskill.pt email
- Clone the following repository:

  https://github.com/prpUpSkill/MyUPskillRepo.git
- Make a simple web page about you
- Send it to the repository
- Change *index.html* to add a link to your page