Managing source code with git

Janis Intoy

But mostly stolen from a presentation by

Louis-Emmanuel Martinet

What is version control?

- Tracks changes to files over time so that you can recall specific versions later (+ other more advanced tools)
- Bad idea #1: quick and dumb
  - my_awesome_function_v1.m
  - my_awesome_function_v2.m
  - my_awesome_function_v3.m
- Why?
  - After a few days/weeks, your project is a mess
  - What did you change between v1 and v3?
  - Really hard to collaborate with other people

What is version control?

- Bad idea #2: using only dropbox (better but still very limited)
- Why?
    - Doesn’t save all the versions (or save versions that you don’t really want to save)
    - What did you change between versions?
    - No advanced tools (very limited collaboration)

What is version control?

- Options!

git + GitLab

software

website

Subversion,…

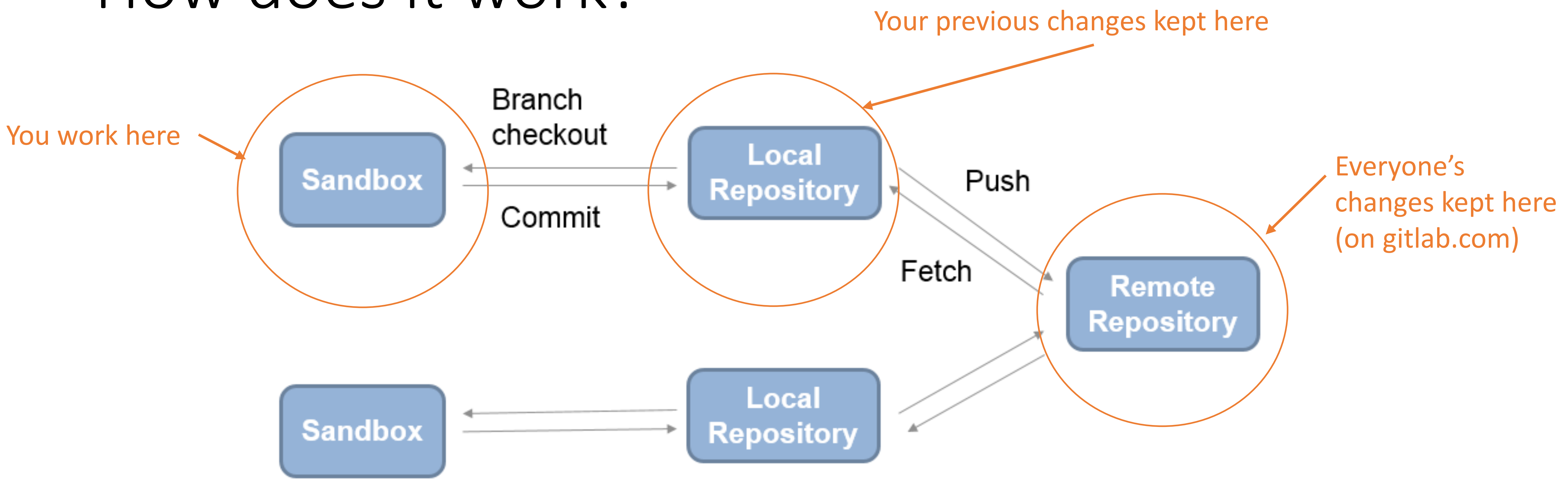
GitHub

How does it work?

- Gitlab.com
  - Serves as a central repository to store our group's code
  - At least one online backup of our code
  - Nice visual interface to navigate your code and its history
  - Easily invite people to download your code and collaborate
  - And much more!

  - **<u>Not</u>** for storing data or images!!!

How does it work?



- https://www.mathworks.com/help/matlab/matlab_prog/set-up-git-source-control.html

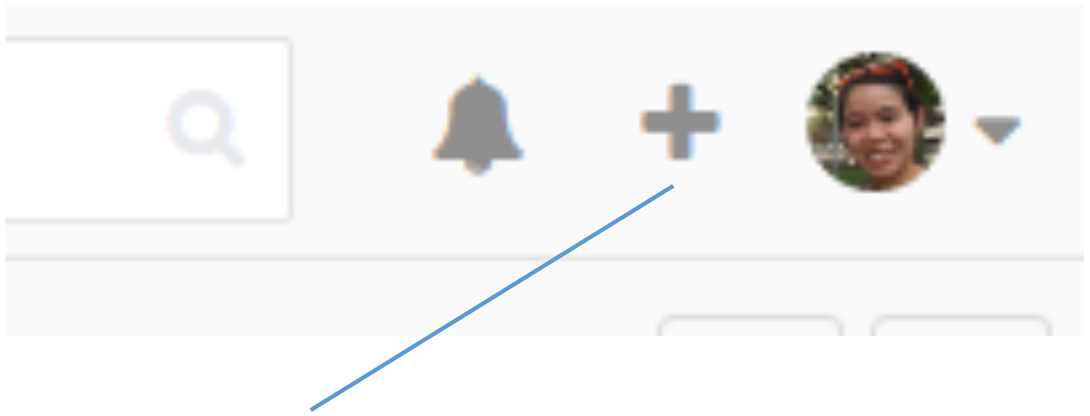
I love it. How do I start?

- Create an account on gitlab.com if you don’t have one
- Tell a group administrator (i.e. Janis – jintoy@bu.edu) what e-mail you signed up with so she can add you to the @aplabBU group
- Download and install git on your computer (google “git download”)

Step 0: Configure git on your computer

- Open up a terminal:
  - Microsoft: (using gitbash (preferred) or cmd)
  - Unix systems: your preferred terminal
  - Matlab command line: start every line with “!git”
- *git config --global user.name "Your Name”*
  git config --global user.email "your_email@whatever.com"
- Add SSH keys (follow instructions at https://gitlab.com/help/ssh/README)

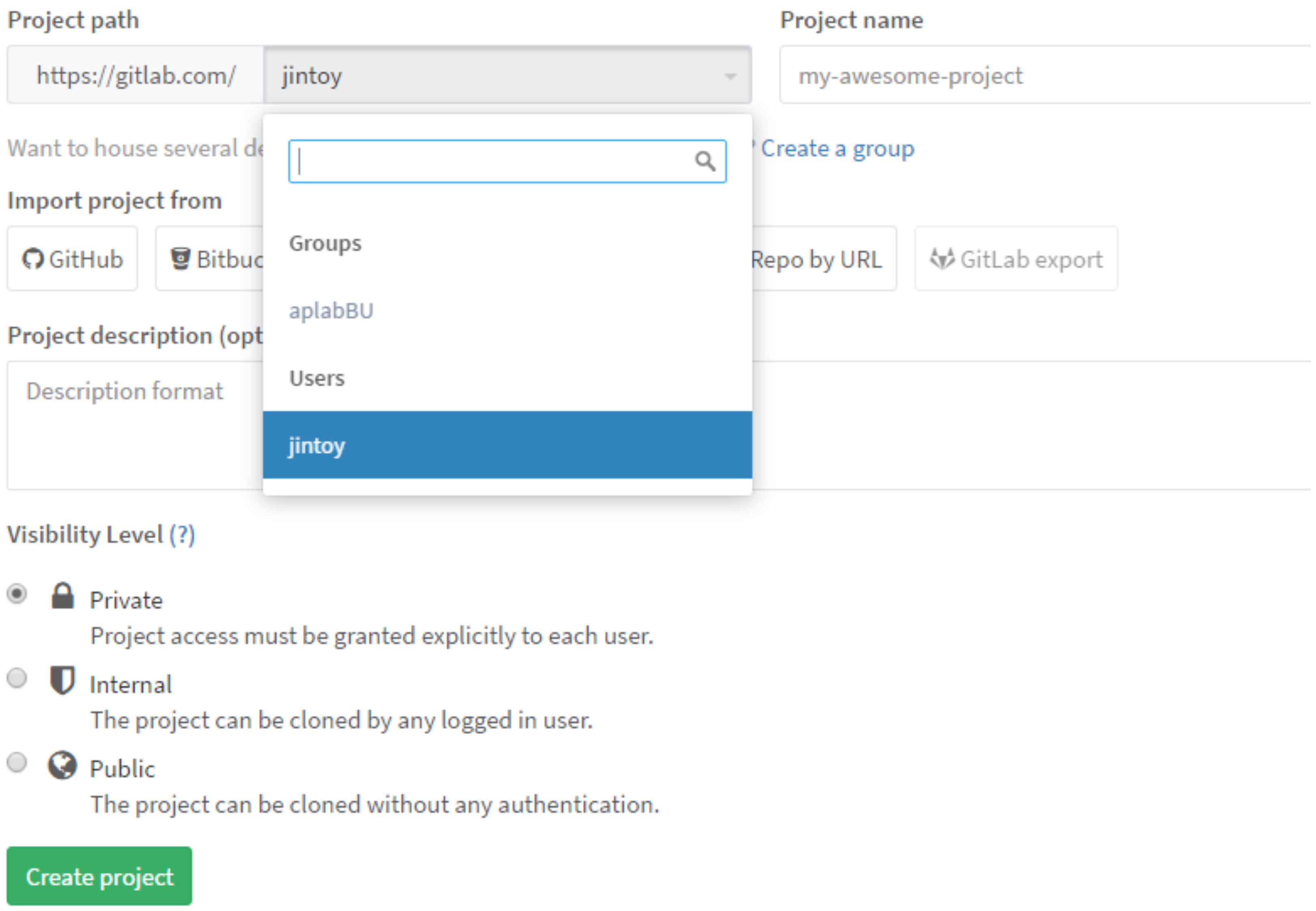
Step 1: Start a new project on gitlab.com



- New Project

Step 1: Start a new project on gitlab.com

- For now, start a personal project (select your username)
- When you're ready you can add a project to the aplabBU group!



Project path

https://gitlab.com/ jintoy

Project name

my-awesome-project

Want to house several de

Create a group

Import project from

Groups

aplabBU

Users

jintoy

Project description (opt

Description format

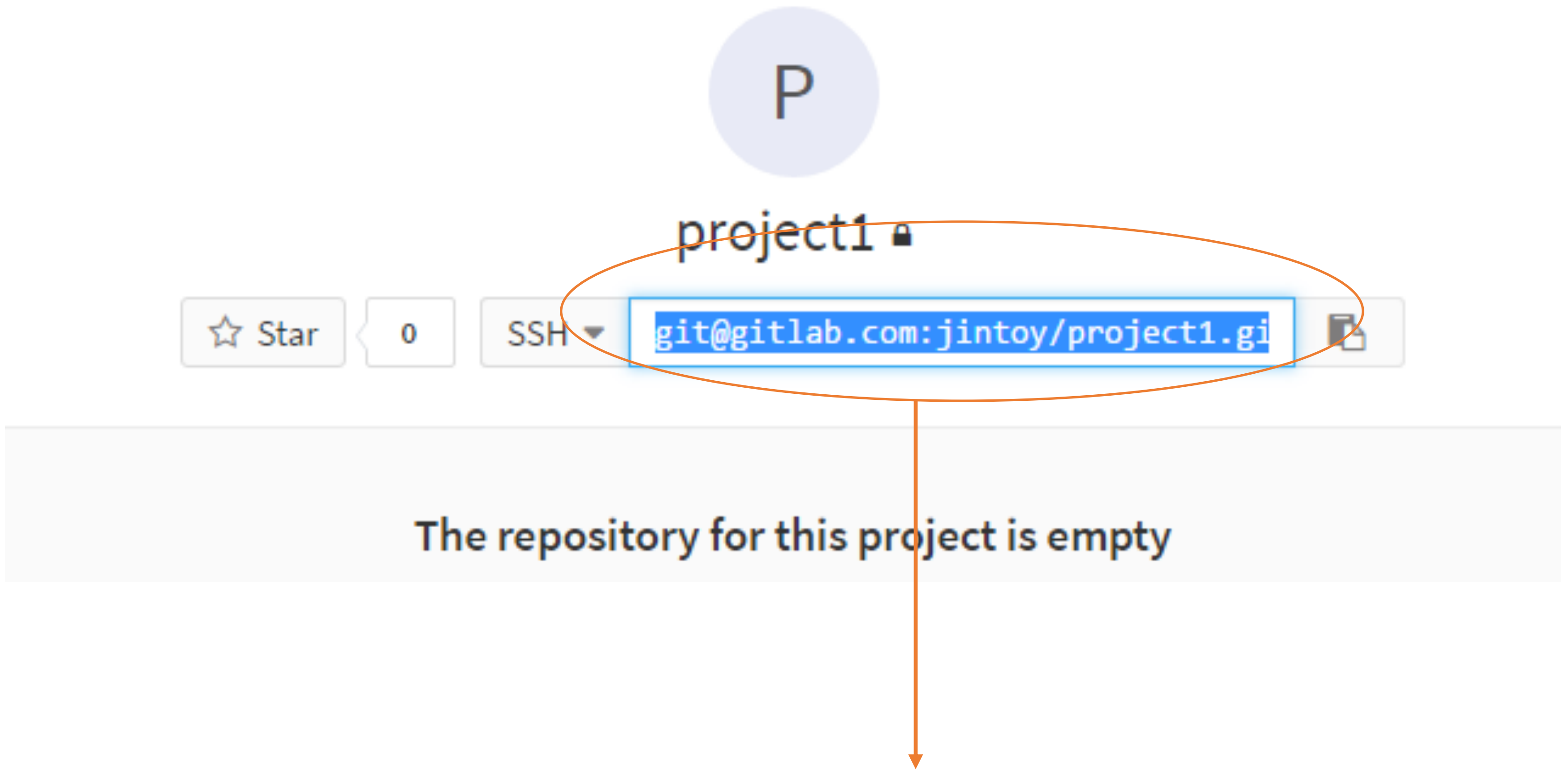
Visibility Level (?)

[✓ Private]
Project access must be granted explicitly to each user.

Internal
The project can be cloned by any logged in user.

Public
The project can be cloned without any authentication.

Step 1: Start a new project on gitlab.com



```
git@gitlab.com:jintoy/project1.gi
```

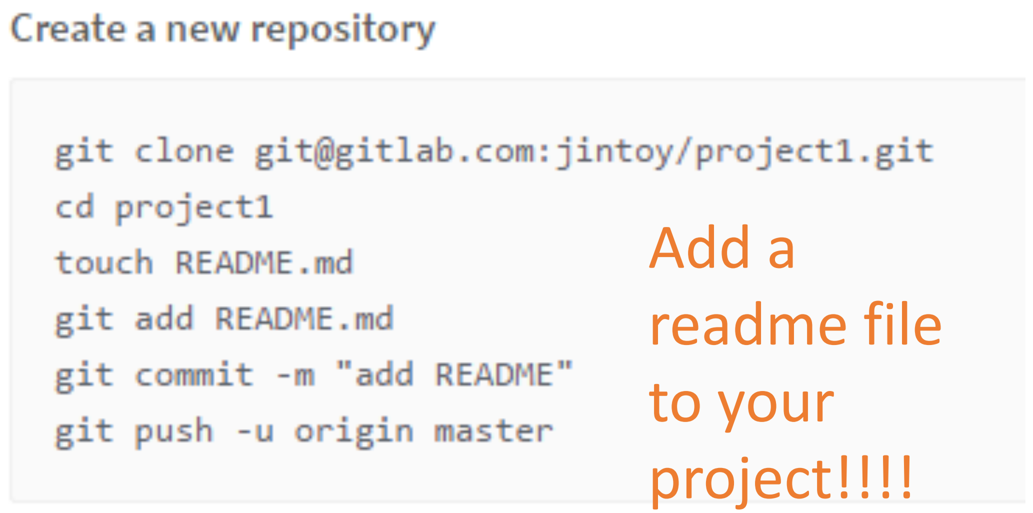
The repository for this project is empty

- The link to your new project!

Step 1: Start a new project on gitlab.com

- In your terminal
  - *cd your_code_folder*
  - *git clone git@gitlab.com:USERNAME/PROJECTNAME.git*
- Creates a folder PROJECTNAME with the repository files inside (empty if it's new)
- Run
  - cd PROJECTNAME
  - Ls
- The folder .git contains all the information about your code history

Later:



Create a new repository

```
git clone git@gitlab.com:jintoy/project1.git
cd project1
touch README.md
git add README.md
git commit -m "add README"
git push -u origin master
```

Add a readme file to your project!!!!

Step 2: add a file to the repository

- Create a new file inside the project folder, for example file1.m (if touch doesn't work create a file the way you usually would)
  - touch file1.m
- Check the status of the repository
  - git status

```
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        file1.m

nothing added to commit but untracked files present (use "git add" to track)
```

Step 2: add a file to the repository

- *git add file1.m*
  git status

```
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   file1.m
```

- Note: you can add multiple files using for example
  *git add *.m*

Step 3: Store the changes locally

- Ask git to store the changes in the database
  git commit -m "New file added"
- Look at the history of the code
  git log

```
commit 40cb55276c08756418e3cf16c81d78b31e4e5223
Author: Janis <jintoy@bu.edu>
Date:   Wed Oct 26 17:18:00 2016 -0400

    New file added
```

Write useful commit messages! This will make it easier to know what changed with each commit.

Step 3: Store the changes locally

Identifier (hash)

```
commit 2f77391614c61be5f3dc74a67131c3ca13e91242
Author: Louis-Emmanuel Martinet <louis.emmanuel.martinet@gmail.com>
Date:   Tue Oct 18 01:26:40 2016 -0400

    New file added

commit 1d13264d6d21fb3f8f81af121c893603b6fc1198
Author: lemartinet <lemartinet@users.noreply.github.com>
Date:   Tue Oct 18 00:59:31 2016 -0400

    Initial commit
```

Step 4: Send the changes to gitlab

```
git status
```

```
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
nothing to commit, working directory clean
```

```
git push
```

- Check your repository on github.com
- When you work with other people, you need to run
  git pull
  before you push to get the last code updates (more later)

Step 5: edit file1, commit and push

- Add something within file1.m, e.g.:

```
function out = file1(in)
out = in;
end
```

- What does git think about it?

```
git status
```

```
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   file1.m

no changes added to commit (use "git add" and/or "git commit -a")
```

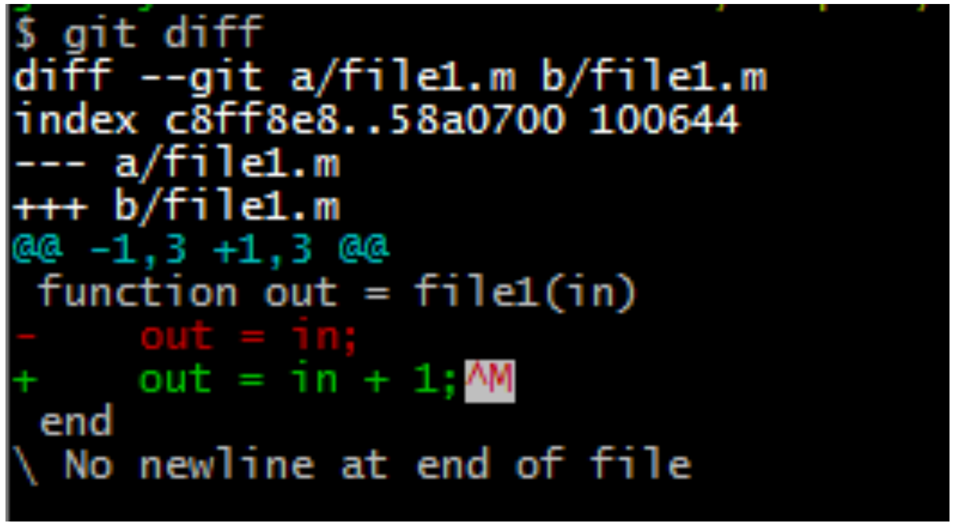
Step 5: edit file1.m, commit and push

- Ask git to store those edits
  - With two commands
    git add file1.m
    git commit -m "Added some code"
  - Using only one command
    git commit -am "Added some code"

Step 6: more edits, difference between versions

- Edit file1.m, for example change line 2:
  out = in + 1;
- Ask git to show the difference with the last committed version
  git diff

```
diff --git a/file1.m b/file1.m
index 94c4e4a..d1f9ba0 100644
--- a/file1.m
+++ b/file1.m
@@ -1,4 +1,4 @@
 function out = file1(in)
-out = in;
+out = in + 1;
 end
```



```
$ git diff
diff --git a/file1.m b/file1.m
index c8ff8e8..58a0700 100644
--- a/file1.m
+++ b/file1.m
@@ -1,3 +1,3 @@
 function out = file1(in)
-      out = in;
+      out = in + 1;^M
 end
\ No newline at end of file
```

- Note: you can compare any commits using their identifiers (found in the log), e.g.:
  git diff 1d13 9da5

Step 7: cancel the edits

- Try
  git checkout -- file1.m
- File1.m is back to its previous state stored in the last commit

Step 8: creating a new branch

- What is a branch?
  - You can picture your code in git as a tree
  - It starts from the trunk called the master branch
  - From there you can create a different version of the code in a different branch

(commits)

$$\begin{tikzcd}
\textit{Master} & C1 \arrow[r] & C2 \arrow[r] & C3 \arrow[dr] & & \\
\textit{versionA} & & & & C4 \arrow[r] & C5
\end{tikzcd}$$

Step 8: creating a new branch

- Create a new branch in git
  git branch versionA
  git branch
- Work in the new branch (i.e. HEAD points to versionA)
  git checkout versionA
- Make some edits (change line 2 again for example)
- Commit the changes
  git commit -am 'Edited file1'
- Return back to the master branch
  git checkout master
- What happened to file1.m?
  - During the checkout, git replaced file1.m with the last version stored in master!
- Compare master and versionA
  git diff versionA

Step 9: Merging branches

- When you’re happy with your code developed in another branch, you can merge it in the main branch, aka master
  git checkout master
  git merge versionA

(you can check that master and versionA are identical using *git diff versionA*)

$$\begin{tikzcd}
\textit{Master} & C1 \arrow[r] & C2 \arrow[r] & C3 \arrow[r] \arrow[dr] & C4 \arrow[r] & C5 \\
\textit{versionA} & & & & C4 \arrow[r] & C5
\end{tikzcd}$$

- You can delete the branch versionA now, since its changes are part of master
  git branch -d versionA

$$\begin{tikzcd}
\textit{Master} & C1 \arrow[r] & C2 \arrow[r] & C3 \arrow[r] & C4 \arrow[r] & C5
\end{tikzcd}$$

Step 10: dealing with conflicts while merging

- If you make changes to the same line of a given file in two different branches and try to merge them, you’ll get a conflict
- That can happen also when you work with other people on the same repository and make edits to the same file that are not compatible
- You need to select the part of the code you want to keep

Step 10: dealing with conflicts while merging

- If you make changes to the same line of a given file in two different branches and try to merge them, you’ll get a conflict
- That can happen also when you work with other people on the same repository and make edits to the same file that are not compatible
- You need to select the part of the code you want to keep

Step 10: dealing with conflicts while merging

- Quick exercise: try to generate a conflict and then solve it
  - Open file1.m, what do you see?

```
function out = file1(in)
<<<<<<< HEAD
out = in + 2;
=======
out = in + 3;
>>>>>>> versionA
End
```

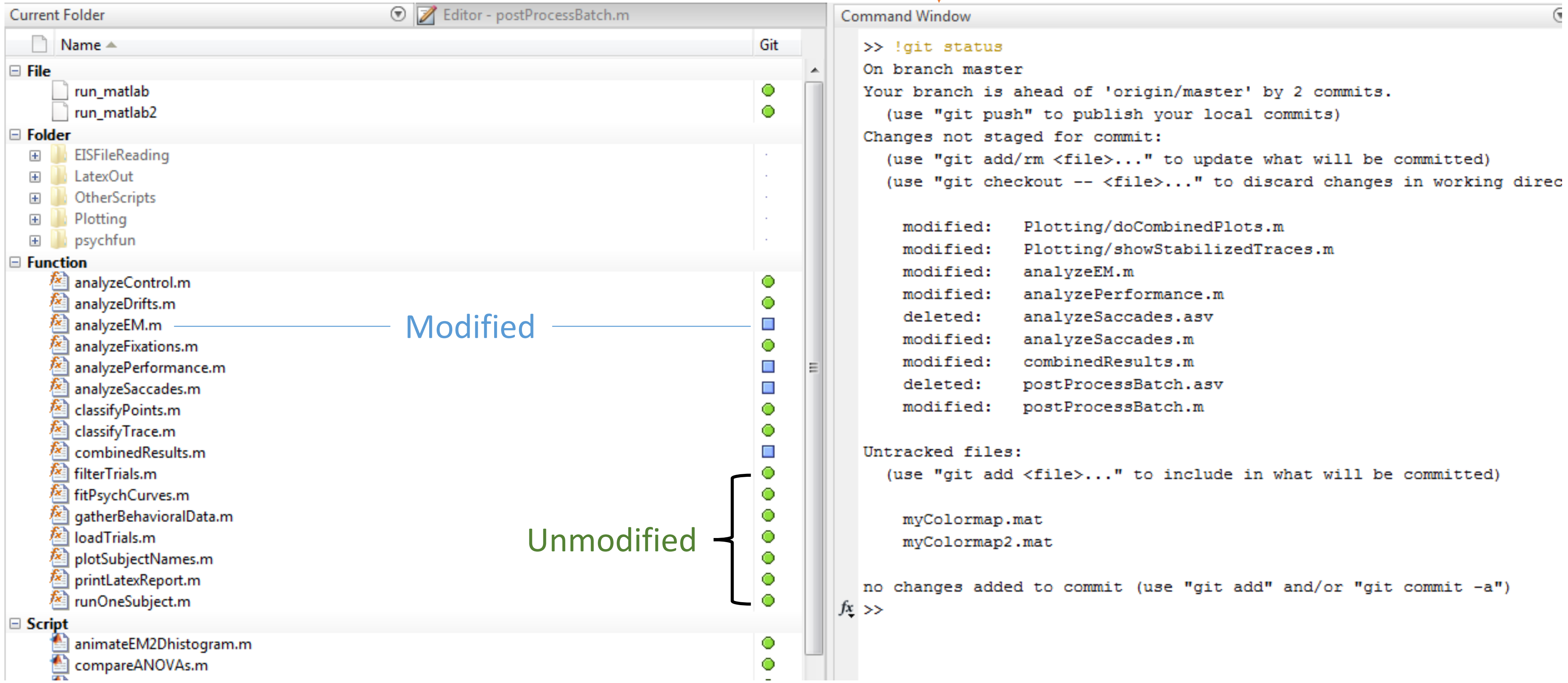
  - You need to edit the code to keep only what you want, and then commit: the conflict is solved!

Step 10: dealing with conflicts while merging

- Quick exercise #2: try to simulate a conflict between two people
  - Clone your github repository into 2 different folders
  - Edit the same line of the same file in each local repository and commit
  - Try to push both of them to github
  - What happens?
    - CONFLICT
  - What to do?
    - You need to pull first to download the conflicting update, merge it and then push again. Try it!

Using git with Matlab

You can use git commands in matlab!



```
>> !git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working direc

    modified:   Plotting/doCombinedPlots.m
    modified:   Plotting/showStabilizedTraces.m
    modified:   analyzeEM.m
    modified:   analyzePerformance.m
    deleted:    analyzeSaccades.asv
    modified:   analyzeSaccades.m
    modified:   combinedResults.m
    deleted:    postProcessBatch.asv
    modified:   postProcessBatch.m

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    myColormap.mat
    myColormap2.mat

no changes added to commit (use "git add" and/or "git commit -a")
>>
```

Other useful commands (1/2)

- To delete a file from the repository, two options
  - Delete both from git and from your computer (the file still exists in previous commits)
    git rm FILE
  - Delete only from git (i.e. the file is not tracked anymore)
    git rm --cached FILE
- To move/rename files within the repository while keeping their history
  git mv FILE1 FILE2
- Create a .gitignore file containing file names, folder names or type of file to be ignored by git (one by line): for example
  file_to_ignore.txt
  folder_to_ignore/
  **.m~*

Other useful commands (1/2)

- You can give a name to your last commit using a tag
  git tag NAME
  or to any commits using their identifier
  git tag NAME IDENTIFIER
  Examples of use: tag the commit that you used to produce some results for a conference, tag releases of a software you develop (v1.0, v1.1, v1.2)
- If you realize you made a mistake in your code just after committing, you can edit your file and amend your commit as if it was right the first time:
  git commit --amend -m 'Message' (it won't work if you've already pushed to github)
- When you push to github, only the master branch is sent. If you want to also send a particular branch to github, use:
  git push -u origin BRANCH (works also for tags)

Final comments (1/2)

- Recommended to have one repository by project, not a huge repository with all the code you’ve ever created.
- Write meaningful comments for commits, your future you will be thankful
- Don’t commit your code every 2 months! Try to commit as soon as you have a significant addition
- It is possible to have a git repository inside your dropbox
  - Provides another backup
  - Gives you access to your last version of your files on another computer if you forgot to commit/push to github

Final comments (2/2)

- Some GUI are available (from github or other ones)
- The most recent versions of Matlab integrate well with git
- Git and github are made for source code (i.e. text files) but you can also add binary files like an image file from a figure or a matlab .mat file. However, it is not optimized for that and space is limited on github.com
- If you want to collaborate but are not invited to a github repository, you can fork it and then send a pull request (more advanced topic, see resources).

More resources

- Many, many resources on the internet
- Github cheat sheet to print
  - https://services.github.com/kit/downloads/github-git-cheat-sheet.pdf
- Fun tutorial
  - https://try.github.io/levels/1/challenges/1
- Play with branching to better understand
  - http://learngitbranching.js.org/
- FAQ on stackoverflow
  - http://stackoverflow.com/questions/315911/git-for-beginners-the-definitive-practical-guide
- More advanced tutorials, for example
  - https://www.atlassian.com/git/tutorials/advanced-overview/
  - http://gitimmersion.com/