Introduction to version control with Git

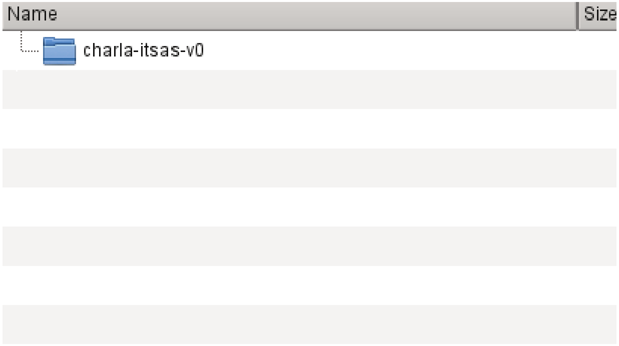
Iñigo Aldazabal Mensa – Materials Physics Center / CFM (CSIC-UPV/EHU)

Software Carpentry Workshop

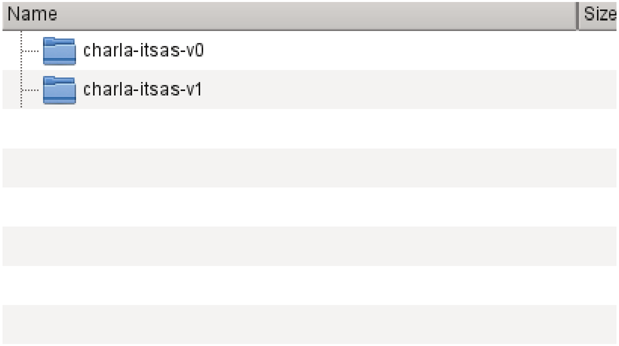
San Sebastian, 27th June 2018

Version Control

Version Control

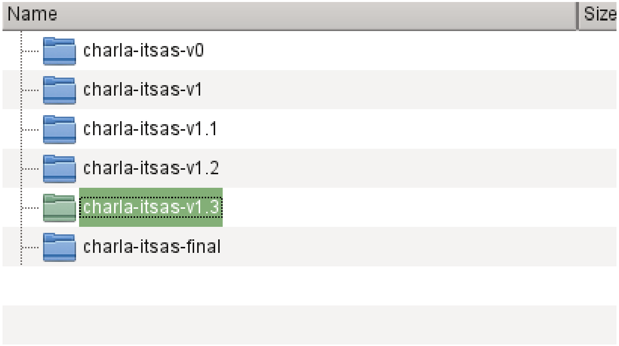


Version Control



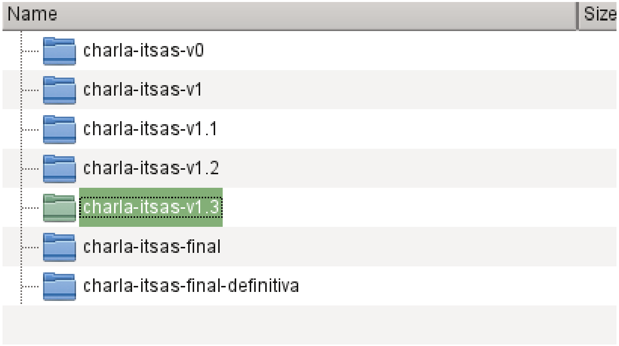
Name	Size
charla-itsas-v0	
charla-itsas-v1	

Version Control



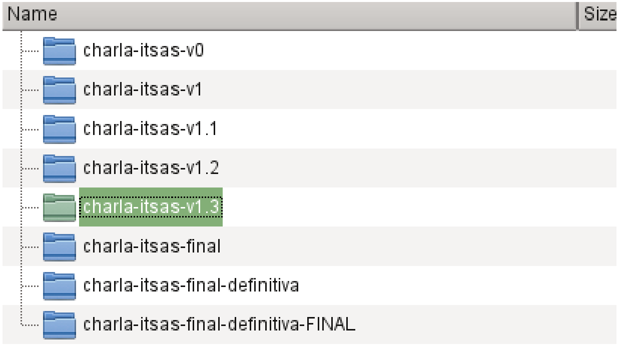
Name | Size
charla-itsas-v0
charla-itsas-v1
charla-itsas-v1.1
charla-itsas-v1.2
charla-itsas-v1.3
charla-itsas-final

Version Control



Name	Size
charla-itsas-v0	
charla-itsas-v1	
charla-itsas-v1.1	
charla-itsas-v1.2	
charla-itsas-v1.3	
charla-itsas-final	
charla-itsas-final-definitiva	

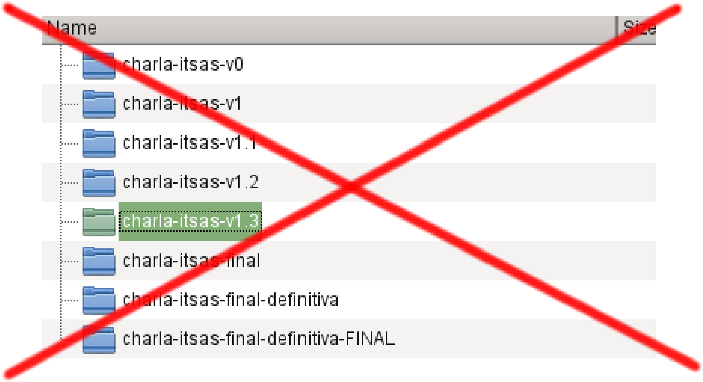
Version Control



Name | Size

- charla-itsas-v0
- charla-itsas-v1
- charla-itsas-v1.1
- charla-itsas-v1.2
- charla-itsas-v1.3
- charla-itsas-final
- charla-itsas-final-definitiva
- charla-itsas-final-definitiva-FINAL

Version Control

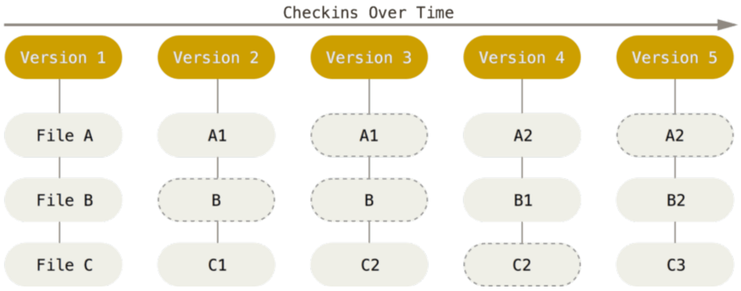


Name	Size
charla-itsas-v0	
charla-itsas-v1	
charla-itsas-v1.1	
charla-itsas-v1.2	
charla-itsas-v1.3	
charla-itsas-final	
charla-itsas-final-definitiva	
charla-itsas-final-definitiva-FINAL	

Version Control System

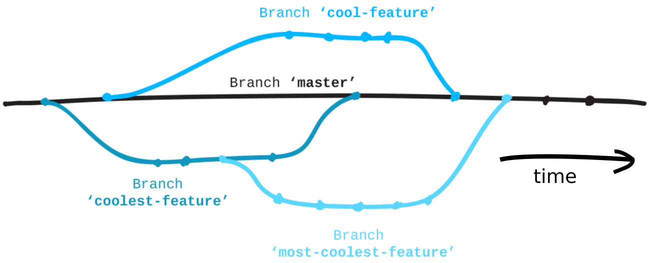
Version control is a system for recording and managing changes made to files and folders.

Features (I)



- All changes are kept.
- You can go “back in time” to any point in a file history.
- You can inspect the differences between any two versions of the files.
- You can track the file/project evolution (what, who, when)

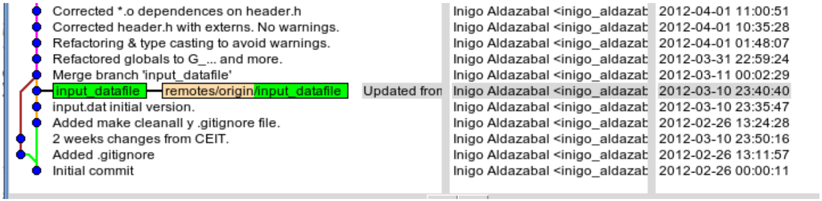
Features (II)



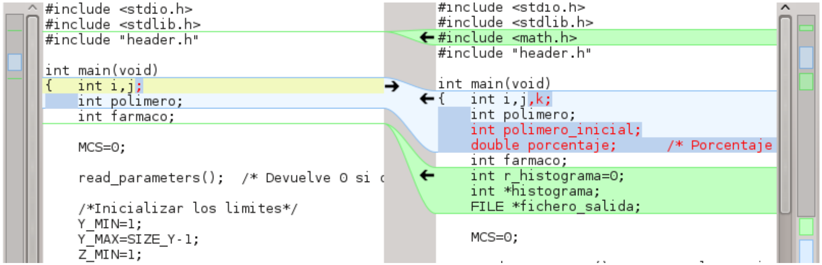
You can have different "branches" of a project.

This allows for things as:

- Easily try out new ideas.
- Create branches for developing specific features.
- Have "release", "development", etc. branches.



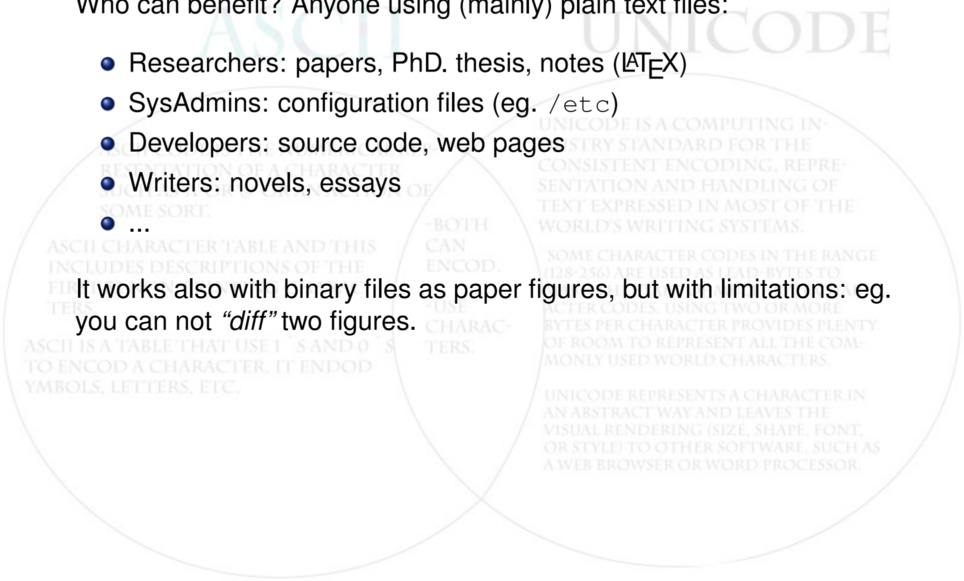
```
Corrected *.o dependences on header.h                                    Inigo Aldazabal <inigo_aldazab  2012-04-01 11:00:51
Corrected header.h with externs. No warnings.                            Inigo Aldazabal <inigo_aldazab  2012-04-01 10:35:28
Refactoring & type casting to avoid warnings.                            Inigo Aldazabal <inigo_aldazab  2012-04-01 01:48:07
Refactored globals to G_... and more.                                    Inigo Aldazabal <inigo_aldazab  2012-03-31 22:59:24
Merge branch 'input_datafile'                                            Inigo Aldazabal <inigo_aldazab  2012-03-11 00:02:29
input_datafile   remotes/origin/input_datafile   Updated fron            Inigo Aldazabal <inigo_aldazab  2012-03-10 23:40:40
input.dat initial version.                                               Inigo Aldazabal <inigo_aldazab  2012-03-10 23:35:47
Added make cleanall y .gitignore file.                                   Inigo Aldazabal <inigo_aldazab  2012-02-26 13:24:28
2 weeks changes from CEIT.                                               Inigo Aldazabal <inigo_aldazab  2012-03-10 23:50:16
Added .gitignore                                                         Inigo Aldazabal <inigo_aldazab  2012-02-26 13:11:57
Initial commit                                                           Inigo Aldazabal <inigo_aldazab  2012-02-26 00:00:11
```



```
#include <stdio.h>
#include <stdlib.h>
#include "header.h"

int main(void)
{   int i,j;
    int polimero;
    int farmaco;

    MCS=0;

    read_parameters();  /* Devuelve 0 si c

    /*Inicializar los limites*/
    Y_MIN=1;
    Y_MAX=SIZE_Y-1;
    Z_MIN=1;
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "header.h"

int main(void)
{   int i,j,k;
    int polimero;
    int polimero_inicial;
    double porcentaje;      /* Porcentaje
    int farmaco;
    int r_histograma=0;
    int *histograma;
    FILE *fichero_salida;

    MCS=0;
```

Use cases

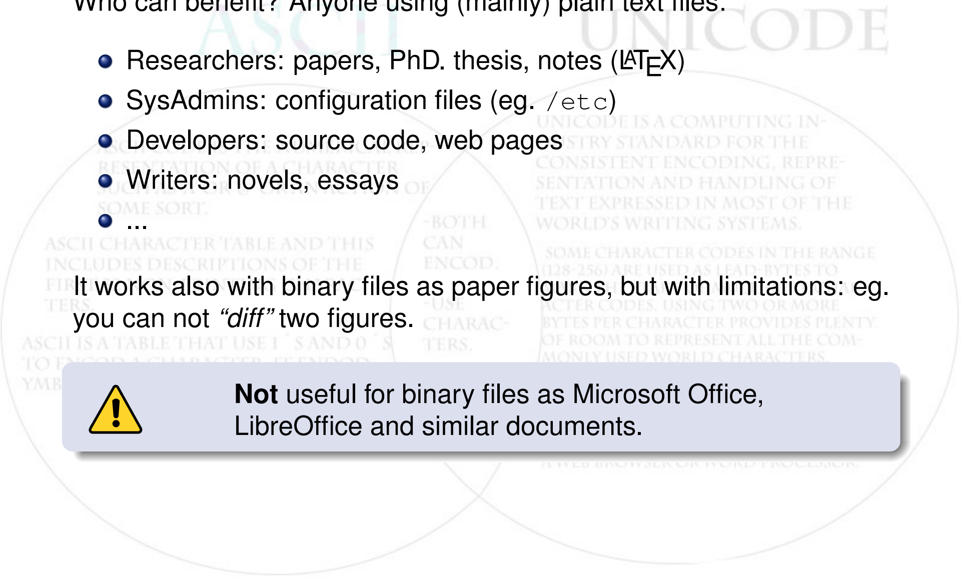


Who can benefit? Anyone using (mainly) plain text files:

- Researchers: papers, PhD. thesis, notes (LaTeX)
- SysAdmins: configuration files (eg. `/etc`)
- Developers: source code, web pages
- Writers: novels, essays
- ...

It works also with binary files as paper figures, but with limitations: eg. you can not *"diff"* two figures.

Use cases



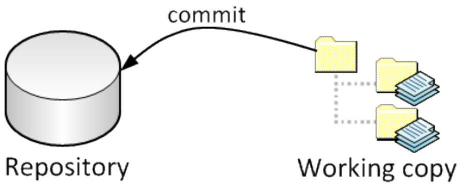
Who can benefit? Anyone using (mainly) plain text files:

- Researchers: papers, PhD. thesis, notes ($\LaTeX$)
- SysAdmins: configuration files (eg. `/etc`)
- Developers: source code, web pages
- Writers: novels, essays
- ...

It works also with binary files as paper figures, but with limitations: eg. you can not *“diff”* two figures.

⚠ **Not** useful for binary files as Microsoft Office, LibreOffice and similar documents.

Key concepts



Repository

A repository is a database of all the edits to, and/or historical versions (snapshots) of, your project.

Working copy

Your working copy is your personal copy of all the files in the project.

Commit

Once you have finished making changes to your working copy, you ***commit*** them to the repository.

“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.“

– Git website

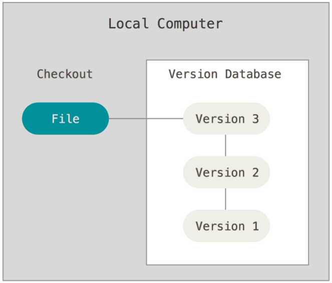
“Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

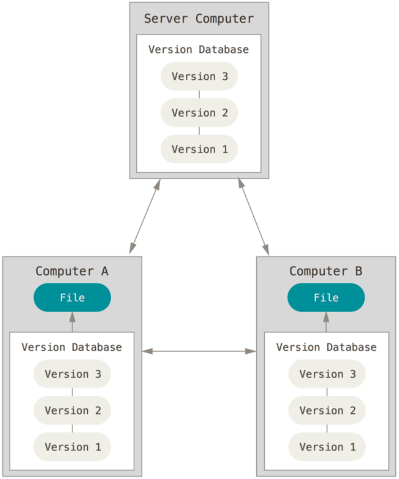
Git is easy to learn and has a tiny footprint with lightning fast performance.“

– Git website

Distributed Version Control System

- Everybody a has a *local* copy of the full repository!
- Any local copy can be synchronized with any other: none is privileged *a priori*.
- It allows for different workflows.





Let’s `git init` !

The Story

Wolfman and Dracula have been hired by Universal Missions (a space services spinoff from Euphoric State University) to investigate if it is possible to send their next planetary lander to Mars.

They want to be able to work on the plans at the same time, but they have run into problems doing this in the past. If they take turns, each one will spend a lot of time waiting for the other to finish, but if they work on their own copies and email changes back and forth things will be lost, overwritten, or duplicated.

A colleague suggests using version control to manage their work.

Setting up Git

```
git config --global user.name "User Name"
git config --global user.email "user@mail.com"
git config --global color.ui "auto"
git config --global core.editor "nano -w"

# On OSX and Linux
git config --global core.autocrlf input

# On Windows
git config --global core.autocrlf true


# check!
git config --list
```

Creating a Repository

```
git init
```

```
$ mkdir my_repo
$ cd my_repo

$ git init
```

Places to create Git repositories

```
cd                # return to home directory
mkdir planets     # make a new directory planets
cd planets        # go into planets
git init          # make the planets directory a Git
                  #  repository
mkdir moons       # make a sub-directory planets/moons
cd moons          # go into planets/moons
git init          # make the moons sub-directory a Git
                  #  repository
```

- Why is a bad idea to do this?
- How can we “undo” the last `git init`?

Tracking Changes

```
nano <file>

git status
git diff
git add <file>
git commit -m <message>

git log
```

Choosing a commit message

We added to the `mars.txt` file the line
“But the Mummy will appreciate the lack of humidity”

Which of the following commit messages would be most appropriate for the last commit made to “mars.txt”?

1.- “Changes”
2.- “Added line ‘But the Mummy will appreciate the lack of humidity’ to mars.txt”
3.- “Discuss effects of Mars’ climate on the Mummy”

Commiting changes to Git

Which command(s) below would save the changes of `myfile.txt` to my local Git repository?

```
(1) $ git commit -m "my recent changes"
```

```
(2) $ git init myfile.txt
    $ git commit -m "my recent changes"
```

```
(3) $ git add myfile.txt
    $ git commit -m "my recent changes"
```

```
(4) $ git commit -m myfile.txt "my
  recent changes"
```

`bio` Repository

Create a new Git repository on your computer called `bio`.

Write a three-line biography for yourself in a file called `me.txt`, commit your changes, then modify one line, add a fourth line, and display the differences between its updated state and its original state.

Remember the workflow!

```
  $ git init

  $ git status
  $ git add
  $ git commit

  $ git diff
  $ git log
```

Exploring History

```
  git diff
  git checkout
```

Recovering Older Versions of a File

Jennifer has made changes to the Python script that she has been working on for weeks, and the modifications she made this morning "broke" the script and it no longer runs. She has spent ∼1hr trying to fix it, with no luck...
Luckily, she has been keeping track of her project's versions using Git! Which commands below will let her recover the last committed version of the Python script called `data_cruncher.py`?

```
(1) $ git checkout HEAD
```

```
(2) $ git checkout HEAD data_cruncher.py
```

```
(3) $ git checkout HEAD~1 data_cruncher.py
```

```
(4) $ git checkout <unique ID of last commit>
    data_cruncher.py
```

```
(5) Both 2 & 4
```

Understanding Workflow and History

What is the output of `cat venus.txt` at the end of this set of commands?

```
$ cd planets
$ nano venus.txt #input the following text: Venus is beautiful and
    full of love
$ git add venus.txt
$ nano venus.txt #add the following text: Venus is too hot to be
    suitable as a base
$ git commit -m "comments on Venus as an unsuitable base"
$ git checkout HEAD venus.txt
$ cat venus.txt #this will print the contents of venus.txt to the
    screen
```

```
(1) Venus is too hot to be suitable as a base
```

```
(2) Venus is beautiful and full of love
```

```
(3) Venus is beautiful and full of love
    Venus is too hot to be suitable as a base
```

```
(4) Error because you have changed venus.txt without committing the
     changes
```

Checking Understanding of `git diff`

Consider this command: `git diff HEAD~3 mars.txt`. What do you predict this command will do if you execute it? What happens when you do execute it? Why?

Try another command, `git diff [ID] mars.txt`, where `[ID]` is replaced with the unique identifier for your most recent commit. What do you think will happen, and what does happen?

Ignoring Things

To ignore eg. `.o`, `.pdf`, `.aux`, etc. files (or folders), add the corresponding patterns to the `.gitignore` file.

```
$ cat .gitignore
*.o
*.pdf
*.aux
results/
```

Including specific files

How would you ignore all `.data` files in your root directory except for `final.data`?

Hint: Find out what “`!`” (the exclamation point operator) does.

Log-files

You wrote a script that creates many intermediate log-files of the form `log_01`, `log_02`, `log_03`, etc. You want to keep them but you do not want to track them through git.

(1) Write **one** `.gitignore` entry that excludes files of the form `log_01`, `log_02`, etc.

(2) Test your "ignore pattern" by creating some dummy files of the form `log_01`, etc.

(3) You find that the file `log_01` is very important after all, add it to the tracked files without changing the `.gitignore` again.

(4) Discuss with your neighbor what other types of files could reside in your directory that you do not want to track and thus would exclude via `.gitignore`.

Remotes in GitHub

```
$ cd my_repo

$ git remote add origin http://github.
   com/user/repo.git
$ git push origin master
$ pit pull origin master
```

GitHub GUI

Browse to your `planets` repository on GitHub.

Under the Code tab, find and click on the text that says “XX commits” (where “XX” is some number). Hover over, and click on, the three buttons to the right of each commit. What information can you gather/explore from these buttons? How would you get that same information in the shell?

push vs. commit

In this lesson, we introduced the “`git push`” command. How is “`git push`” different from “`git commit`”?

Fixing up remote settings

It happens quite often in practice that you made a typo in the remote URL. This exercise is about how to fix this kind of issues. First start by adding a remote with an invalid URL:

```
  git remote add broken https://github.com
     /this/url/is/invalid
```

Do you get an error when adding the remote? Can you think of a command that would make it obvious that your remote URL was not valid? Can you figure out how to fix the URL (tip: use "`git remote -h`")? Don't forget to clean up and remove this remote once you are done with this exercise.

Collaborating

```
  $ git clone http://github.com/user/repo.
     github
```

Review changes

The Owner push commits to the repository without giving any information to the Collaborator. How can the Collaborator find out what has changed with command line? And on GitHub?

Solving Conflicts

```
$ git push origin master
...
 ! [rejected]

$ git pull origin master
...
CONFLICT (content): Merge conflict in
   ...
```

Solving conflicts

Clone instructor's repository

```
$ git clone https://github.com/iamc/
    planets.git
```

Add a new file to it, and modify `mars.txt` file (add a line)

Wait...

When asked by your instructor, pull his changes from the repository to create a conflict, then resolve it.