

Version Control (with Git)

"FINAL".doc



FINAL.doc!



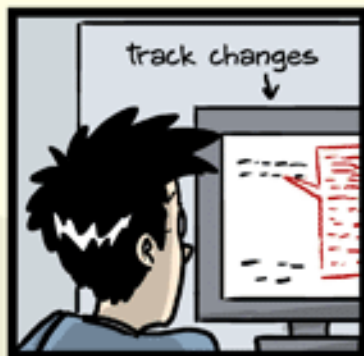
FINAL_rev.2.doc



FINAL_rev.6.COMMENTS.doc



FINAL_rev.8.comments5.
CORRECTIONS.doc



FINAL_rev.18.comments7.
corrections9.MORE.30.doc



FINAL_rev.22.comments49.
corrections.10.#@\$%WHYDID
ICOMETOGRADSCHOOL?????.doc



¿Quién hizo qué?



Originally developed for large software projects with many developers.

Also useful for single user, e.g. to:

- Keep track of history and changes to files,
- Be able to revert to previous versions,
- Keep many different versions of code well organized,
- Easily archive exactly the version used for results in publications,
- Keep work in sync on multiple computers.



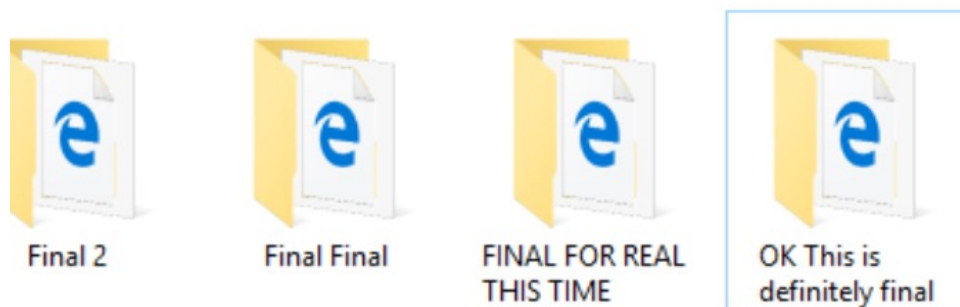
git



I prefer the real version control

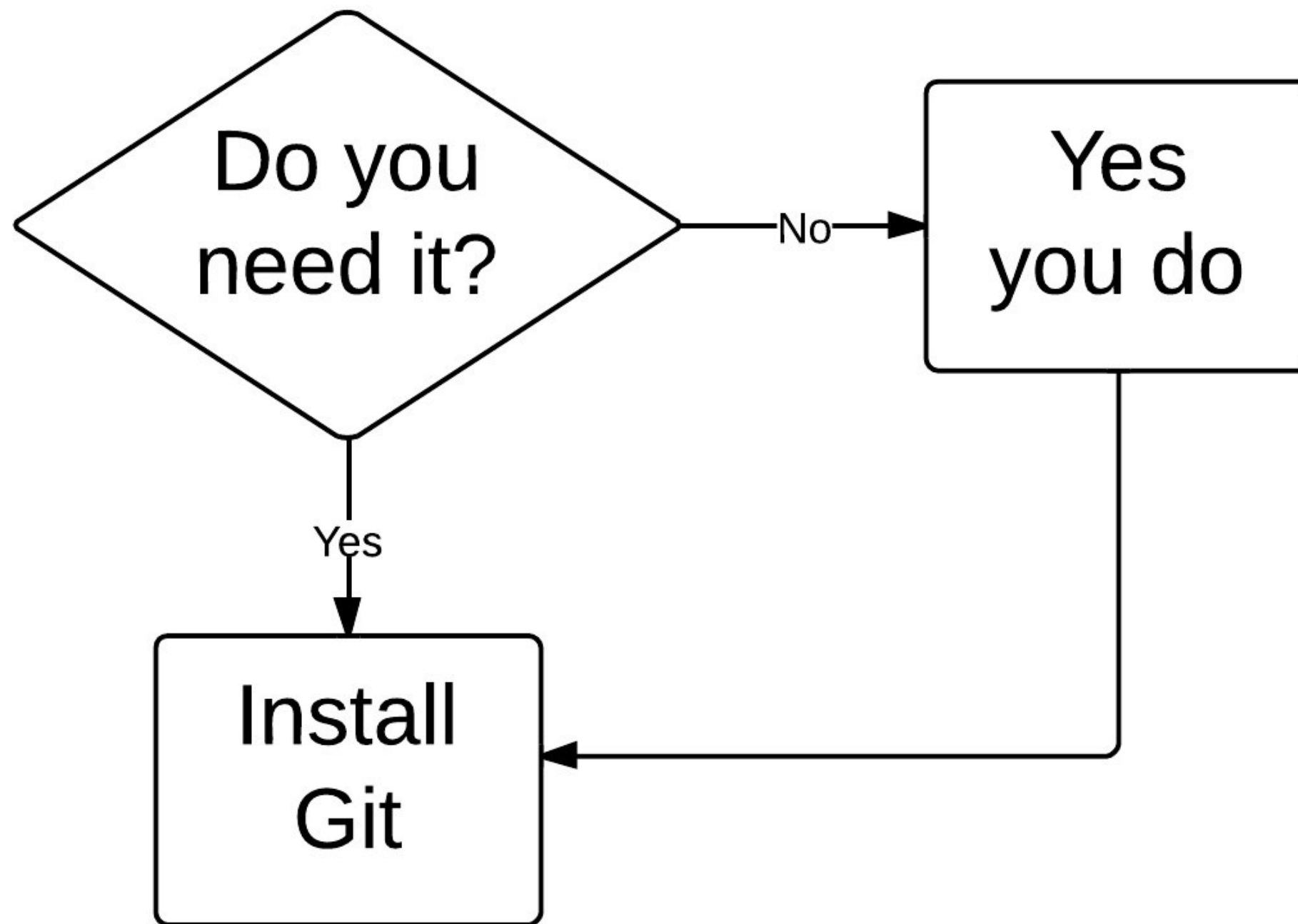


I said the *real* version control



Perfection

Version Control Flowchart



En la mayoría de distribuciones de OS viene por defecto

<https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>

Ingredientes mínimos de un Repositorio

Una carpeta donde van a ir todos los archivos del proyecto

Los archivos del proyecto

Una carpeta .git (registro de cambios)

Opcional:

Un archivo .gitignore

Hola
Mundo

Escribir un programa en python titulado “holamundo.py” en la carpeta /Descargas/HolaMundo/ que al ejecutarlo como `python holamundo.py`, muestre en la terminal el mensaje “Hola Mundo”

Hola
Mundo

```
$ cd  
$ mkdir HolaMundo  
$ cd HolaMundo  
$ echo "print(\"hola mundo\")" > holamundo.py  
$ python holamundo.py  
  
$ git init
```

```
$ git status
```

¿Qué es una branch?

¿Cómo empezamos a "track" (rastrear) los Untracked files?

git add

git add miarchivo

informo que quiero que git registre el estado actual de `miarchivo` y esté pendiente de si hay cambios

git add

```
$ git add miarchivo
```

informo que quiero que git registre el estado actual de `miarchivo` y esté pendiente de si hay cambios

```
$ git diff
```

git add y
git commit

```
$ git add miarchivo
```

informo que quiero que git registre el estado actual de `miarchivo` y esté pendiente de si hay cambios

```
$ git commit -m "mensaje personalizado, informativo"
```

informo que quiero que `miarchivo` en la versión que le hice add quede “guardado”

git add y
git commit

```
$ git add miarchivo
```

informo que quiero que git registre el estado actual de `miarchivo` y esté pendiente de si hay cambios (llevarlo a la caja registradora)

```
$ git commit -m "mensaje personalizado, informativo"
```

informo que quiero que `miarchivo` en la versión que le hice add quede "guardado" (pagarlo)

git add y
git commit

```
$ git add miarchivo
```

informo que quiero que git registre el estado actual de `miarchivo` y esté pendiente de si hay cambios (llevarlo a la caja registradora)

```
$ git commit -m "mensaje personalizado, informativo"
```

informo que quiero que `miarchivo` en la versión que le hice add quede “guardado” (pagarlo)

Importante: Los cambios que le haga al archivo entre que hice add e hice commit no quedarán guardados

```
$ git log
```


git
checkout

```
$ git log
```

¿Qué es un hash?

```
$ git checkout hash
```

```
$ git checkout master
```

Git uses a distributed model:

When you clone a repository you get all the history too,

All stored in .git subdirectory of top directory.

Usually don't want to mess with this!



Más información en:


<https://swcarpentry.github.io/git-novice/>


```
git clone https://github.com/saint-germain/astropy
```

This directory has a subdirectory `.git` with complete history.


Crear un repositorio de prueba

- Ir a su cuenta de GitHub
- Crear un nuevo repositorio
- Miramos el **QuickSetup**
 - “... or push an existing repository”
- Recargamos la página
- Revisamos el enlace a la historia del repo

 90 commits

Crear un repositorio de prueba

- Ir a su cuenta de GitHub
- Crear un nuevo repositorio
- Miramos el **QuickSetup**
 - “... or push an existing repository”
- Recargamos la página
- Revisamos el enlace a la historia del repo

 90 commits

Editar el repo en la
página

- Add file
 - Upload...
 - Create...
- git pull
- git push

Repositorios ajenos

- git clone
- Limitaciones?
- En la página de GitHub: Fork -> crea una copia propia en su cuenta
- Para contribuir al repositorio ajeno: Pull request

Original style, still widely used (e.g. CVS, Subversion)

One **central repository** on server.

Developers' workflow (simplified!):

- Check out a **working copy**,
- Make changes, test and debug,
- Check in (**commit**) changes to repository (with comments).
This creates new **version number**.
- Run an **update** on working copy to bring in others' changes.

The system keeps track of **diffs** from one version to the next (and info on who made the changes, when, etc.)

A **changeset** is a collection of **diffs** from one commit.

Only the server has the full history.

The working copy has:

- Latest version from repository (from last `checkout`, `commit`, or `update`)
- Your local changes that are not yet committed.

Note:

- You can retrieve older versions from the server.
- Can only *commit* or *update* when connected to server.
- When you *commit*, it will be seen by anyone else who does an *update* from the repository.

Often there are `trunk` and `branches` subdirectories.

Git uses a distributed model:

- `git commit` commits to your clone's `.git` directory.
- `git push` sends your recent changesets to another clone by default: the one you cloned from (e.g. bitbucket), but you can push to any other clone (with write permission).
- `git fetch` pulls changesets from another clone by default: the one you cloned from (e.g. bitbucket)
- `git merge` applies changesets to your working copy

Note: pushing, fetching, merging only needed if there are multiple clones.

Advantages of distributed model:

- You can commit changes, revert to earlier versions, examine history, etc. without being connected to server.
- Also without affecting anyone else's version if you're working collaboratively. *Can commit often while debugging.*
- No problem if server dies, every clone has full history.

For collaboration will still need to push or fetch changes eventually and [git merge](#) may become more complicated.