UNIVERSIDAD DE COSTA RICA

CONTROL DE VERSIONES

Prof. Marlon Brenes y Prof. Federico Muñoz
Escuela de Física, Universidad de Costa Rica

EFis Escuela de Física

Que es control de versiones?

- El control de versiones es una herramienta para el **manejo** de **cambios** en un **conjunto de archivos**
    - Mantiene un historial de versiones

- Es importante utilizar control de versiones para
    - Posibilitar el trabajo colaborativo
    - Proceder de manera colaborativa de forma más sencilla
    - Organizar formalmente el código
    - Mantener un registro de cambios
    - Mantener reproducibilidad
    - Mantener muchas versiones por medio de **ramas (branches)**

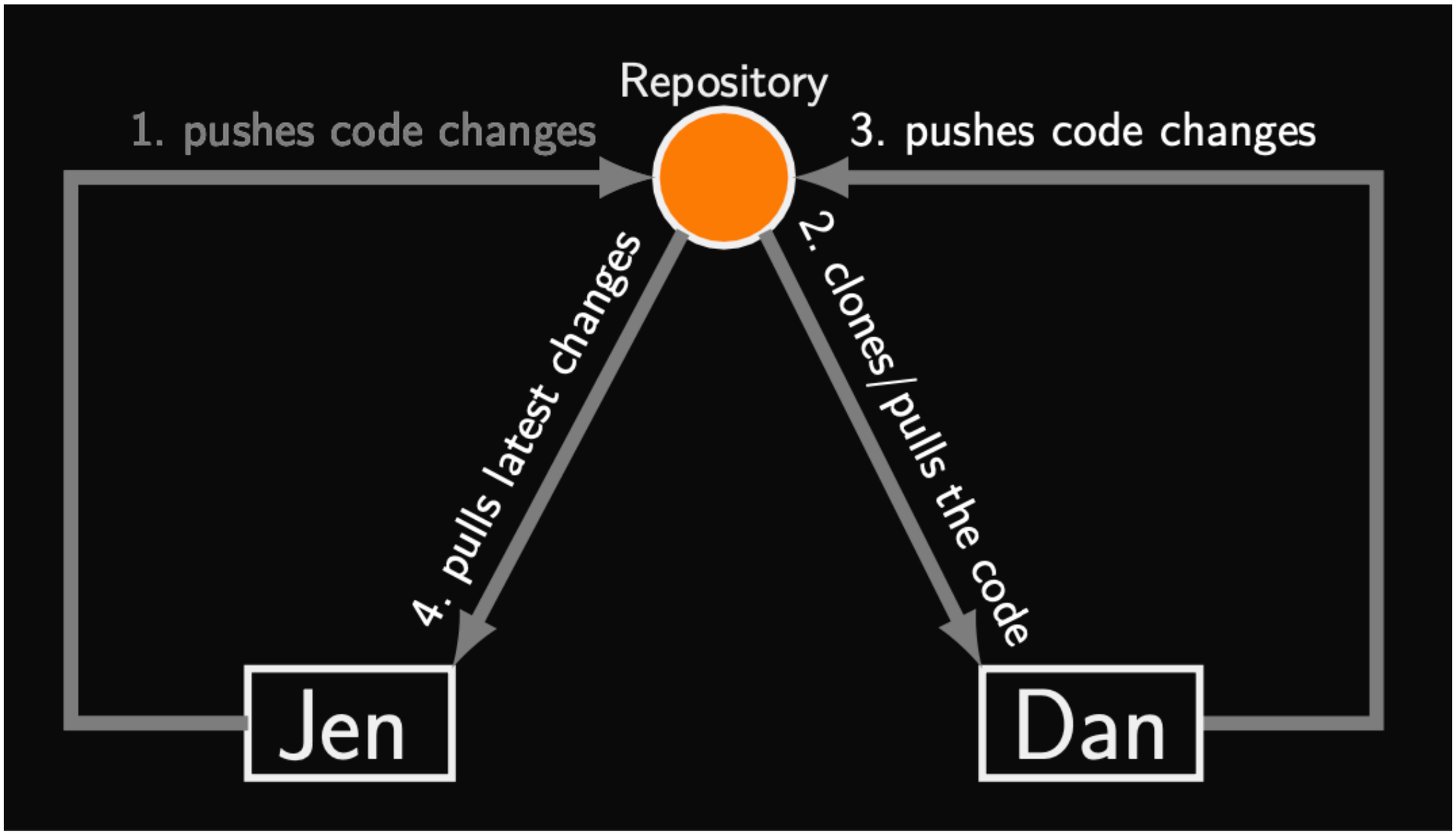
Tipos de flujos de trabajo en control de versiones

- Centralizado
  - Un **repositorio** autoritario, remoto y central
  - Clones locales pueden someter cambios
  - E.g.: CVS, SNV, Github, Gitlab

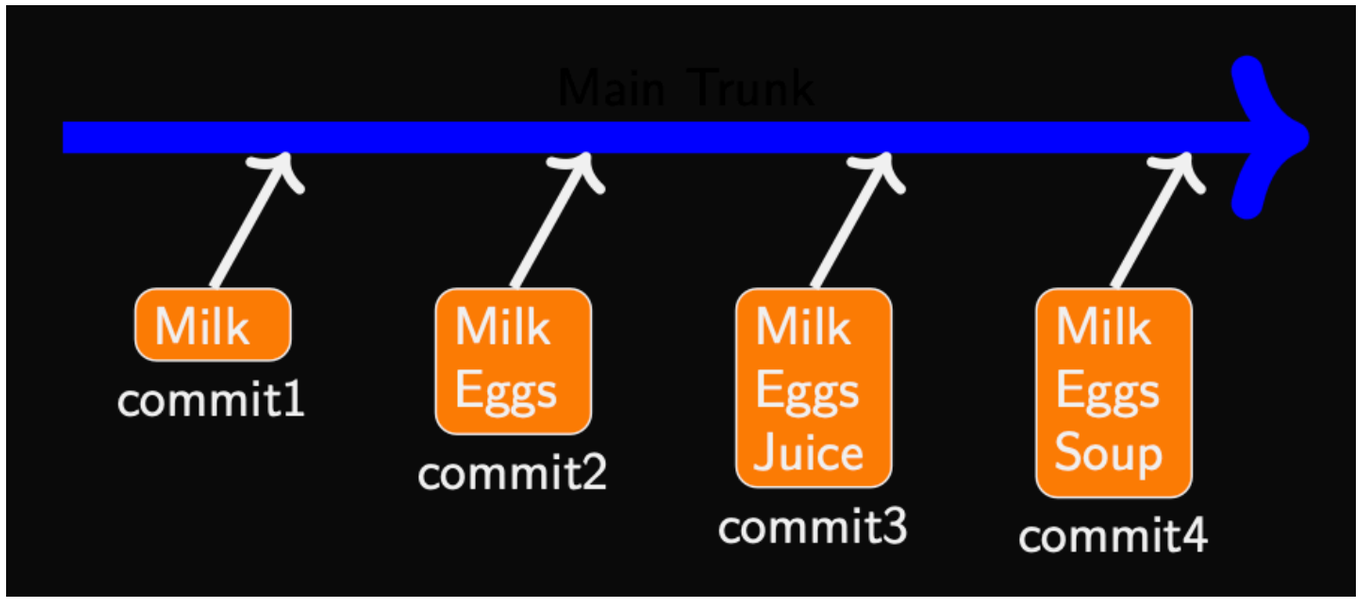
- Distribuido
  - Cada clon es válido, cada repositorio es completo
  - Cualquier repositorio puede ser clonado
  - Se puede "jalar" (*pull*) de cualquier otro y resolver diferencias
  - Se suele preferir el flujo centrali
  - E.g.: Github, Mercurial

- Similitudes:
  - Ambos paradigmas son equivalentes cuando se trabaja con un solo repositorio local
  - Se debe ser explícito con respecto a cuales cambios son seguidos (tracked) y comprometidos (commits)
  - Se puede solicitar un historial y moverse hacia atrás o adelante en el tiempo

Control de versiones centralizado

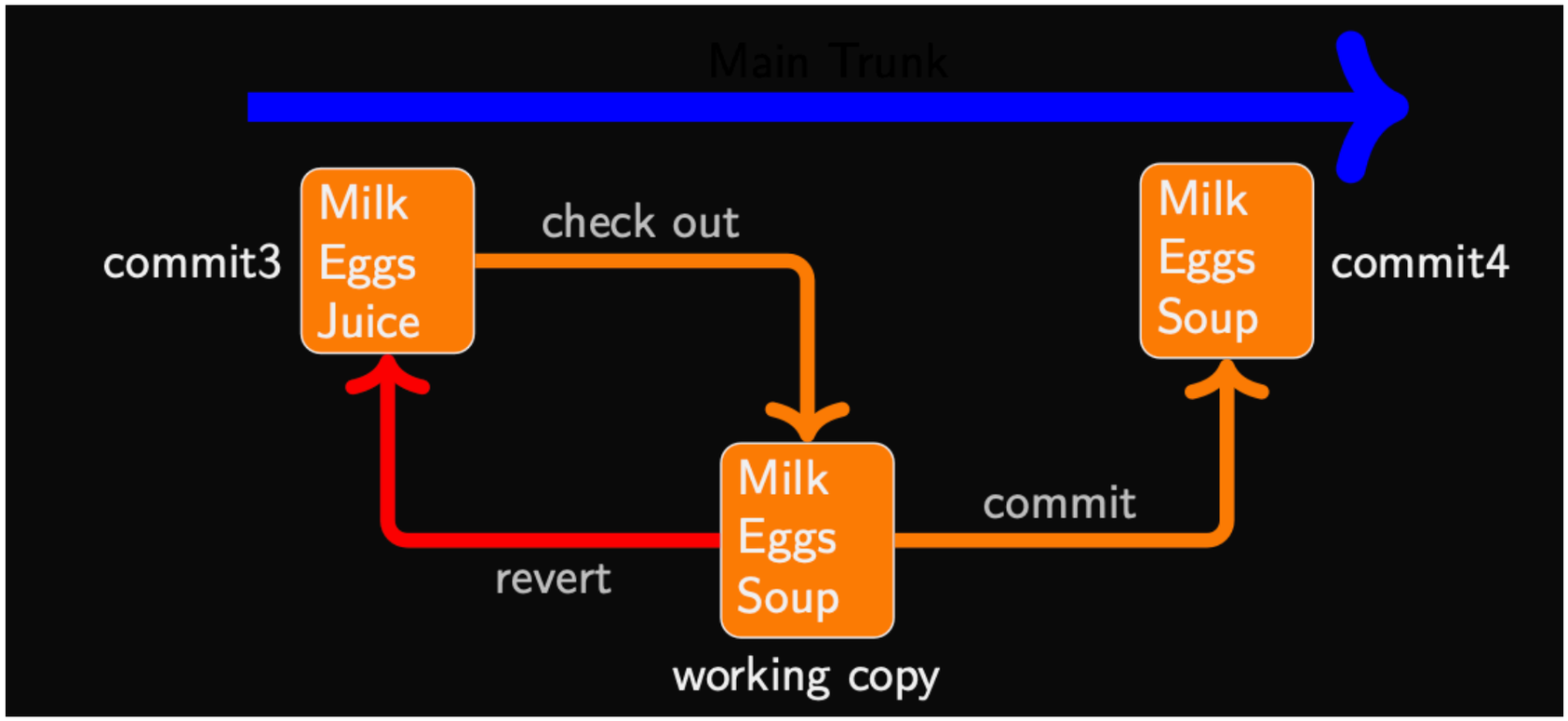


Compromiso de cambios locales (*commits*)



- Los **commits** son **diferencias** en el archivo en cuestión
- Los archivos que no cambian no se almacenan de nuevo
- Los **commits** actúan como fotografías cuando son "chequeados" (checked out)

Checkout y edición



- La idea es no dar acceso directo al repositorio
- En lugar de eso, se realiza una copia. En cierta "rama", se trabajan los cambios
- Una vez que los cambios son correctos, se compromete el cambio a la "rama"

Identidad

- La primera vez que se utiliza git, el sistema puede dar una advertencia de que la identidad no ha sido establecida
- La mejor forma es identificarse de manera global:

```
git config –global user.email “marlon.brenes@utoronto.ca”

git config –global user.name “mbrenesn”
```

- Después de esto, git marcará todos los commits a ese usuario

Creación de un repositorio local

- El primer paso es crear un repositorio para el código

```
$ mkdir code # if there is no  code yet

$ cd code

$ git init --initial-branch main  # creates a repository for this directory, in the 'main' branch
```

- Esto crea un directorio `.git` en el directorio actual
- `.git` is nuestro repositorio local
- El directorio actual contiene una copia (de momento vacía incluso si tenemos archivos en el directorio)
- El directorio `.git` está escondido pero se puede listar con `ls -a`

```
$ ls -a
. .. .git
$
```

Control de versiones: Añadir archivos al repositorio

- Primero se añaden archivos al repositorio y luego se **comprometen** (*commit*)

```
$ echo "some data" > temp.txt
$ cp temp.txt temp2.txt
$ cp temp.txt temp3.txt
$ ls
temp.txt temp2.txt temp3.txt

$ git add temp.txt text2.txt

$ git commit -m "First commit for my repository"
[main (root-commit) dd9d139] First commit for my repository
 2 files changed, 2 insertions(+)
 create mode 100644 temp.txt
 create mode 100644 temp2.txt
$
```

- Note que `temp3.txt` no aparece porque no ha sido añadido

EFis Escuela de Física

Control de versiones: Comparar versiones de archivos

- Actualicemos los datos para comparar los archivos comprometidos…

```
$ echo "some more data" >> temp.txt
```

```
$ git diff temp.txt
diff --git a/temp.txt b/temp.txt
index 4268632..fdd9353 100644
--- a/temp.txt
+++ b/temp.txt
@@ -1 +1,2 @@
 some data
+some more data
```

- Supongamos que queremos comprometer este cambio:

```
$ git add temp.txt
$ git commit -m "updating data due to ..."
```

- Los archivos presentes en el repositorio son seguidos por el control de versiones y podemos comprometer los cambios con un comando

```
$ git commit -a -m "updating data due to ..."
```

Control de versiones: Reporte de estado

- Con el comando `git log` se pueden revisar los cambios

```
$ git log
commit b0292f6e3a820856f1d29b5aee2acdc4fd9e73c9 (HEAD -> main)
Author: Ramses van Zon <rzon@scinet.utoronto.ca>
Date:   Thu Jan 27 09:50:01 2022 -0500

    updating data due to ...

commit dd9d13999ac5073089e6ea4282b0c78854256bc1
Author: Ramses van Zon <rzon@scinet.utoronto.ca>
Date:   Thu Jan 27 09:49:02 2022 -0500

    First commit for my repository
```

Control de versiones: Reporte de estado

- Con el comando `git log` se pueden revisar los cambios

```
$ git log --stat
commit b0292f6e3a820856f1d29b5aee2acdc4fd9e73c9 (HEAD -> main)
Author: Ramses van Zon <rzon@scinet.utoronto.ca>
Date:   Thu Jan 27 09:50:01 2022 -0500

    updating data due to ...

 temp.txt | 1 +
 1 file changed, 1 insertion(+)

commit dd9d13999ac5073089e6ea4282b0c78854256bc1
Author: Ramses van Zon <rzon@scinet.utoronto.ca>
Date:   Thu Jan 27 09:49:02 2022 -0500

    First commit for my repository

 temp.txt  | 1 +
 temp2.txt | 1 +
 2 files changed, 2 insertions(+)
```

Control de versiones: Eliminar archivos

- Frecuentemente deseamos eliminar archivos del repositorio
- Supongamos que queremos eliminar un archivo en la copia local y en el repositorio

```
$ git rm temp2.txt  # let git do it!
$ git commit -m "Remove temp2.txt"
[main f1af560] removed temp2.txt
 1 file changed, 1 deletion(-)
 delete mode 100644 temp2.txt
$
```

- Para establecer estos cambios en el repositorio:
    - `git push`

Control de versiones: Ramas (*branches*)

- Las ramas son extremadamente útiles para desarrollar código de prueba antes de combinarlo en la rama principal

- Para mostrar la rama actual:
  - `git branch`

- Para mostrar todas las ramas:
  - `git branch -a`

- Para mostrar todas las ramas remotas (más adelante):
  - `git branch -r`

- Para crear una rama
  - `git branch MYNEWBRANCH`

- Para cambiar de rama
  - `git checkout branchname`

Repositorios remotos

- Git es un sistema de control de versiones distribuido
- Se puede **clonar** un repositorio de cualquier lugar y copiarlo en otro lugar
- Cada clon es un repositorio como tal con una historia completa
- Se puede empujar (push) o jalar (pull) el estado de un repositorio a otro
- Este caso difiere de lo que hemos visto anteriormente, en el cual teníamos un repositorio centralizado manejado por distintas ramas. Esta descentralización puede ser conveniente en muchos casos
- Git puede interactuar con repositorios remotos mediante protocolos `ssh`

Control de versiones

- Utilizar control de versiones nos puede ayudar a evitar muchos problemas

- Comprometer cambios lo más frecuente posible

- Incluir mensajes descriptivos a la hora de comprometer cambios

- Se puede utilizar en distintas instancias: código, colaboraciones, artículos…

- Ver tutoriales:
  - https://github.com
  - https://www.vogella.com/tutorials/Git/article.html

Práctica

- Vamos a crear un repositorio con el código de Python para el cálculo de pi y realizar cambios remotos