Taller de Git

Clase 2 - Branches y Workflows

Fernando López

LINTI
Facultad de Informática
Universidad Nacional de la Plata

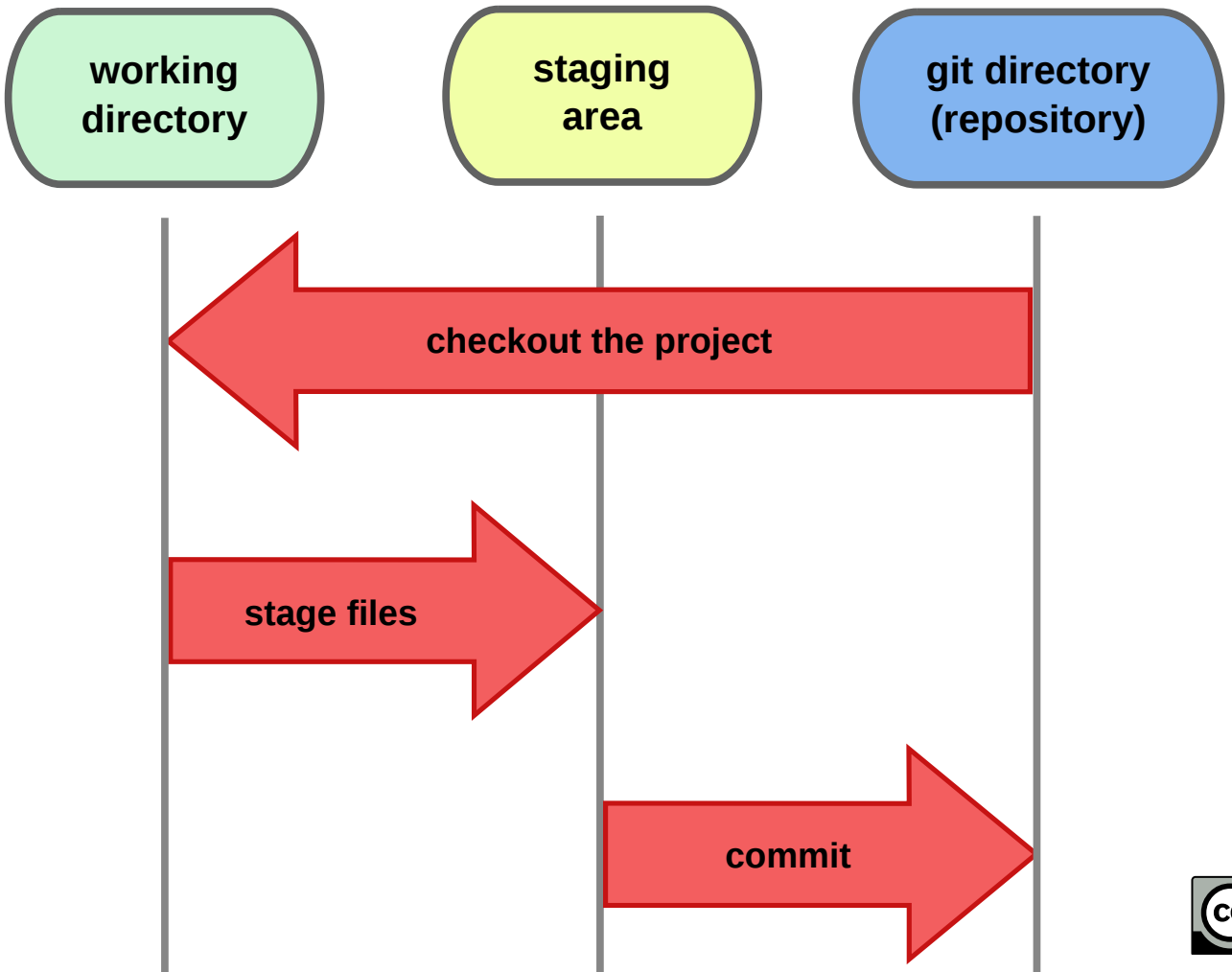
30 de Octubre de 2013

Indice

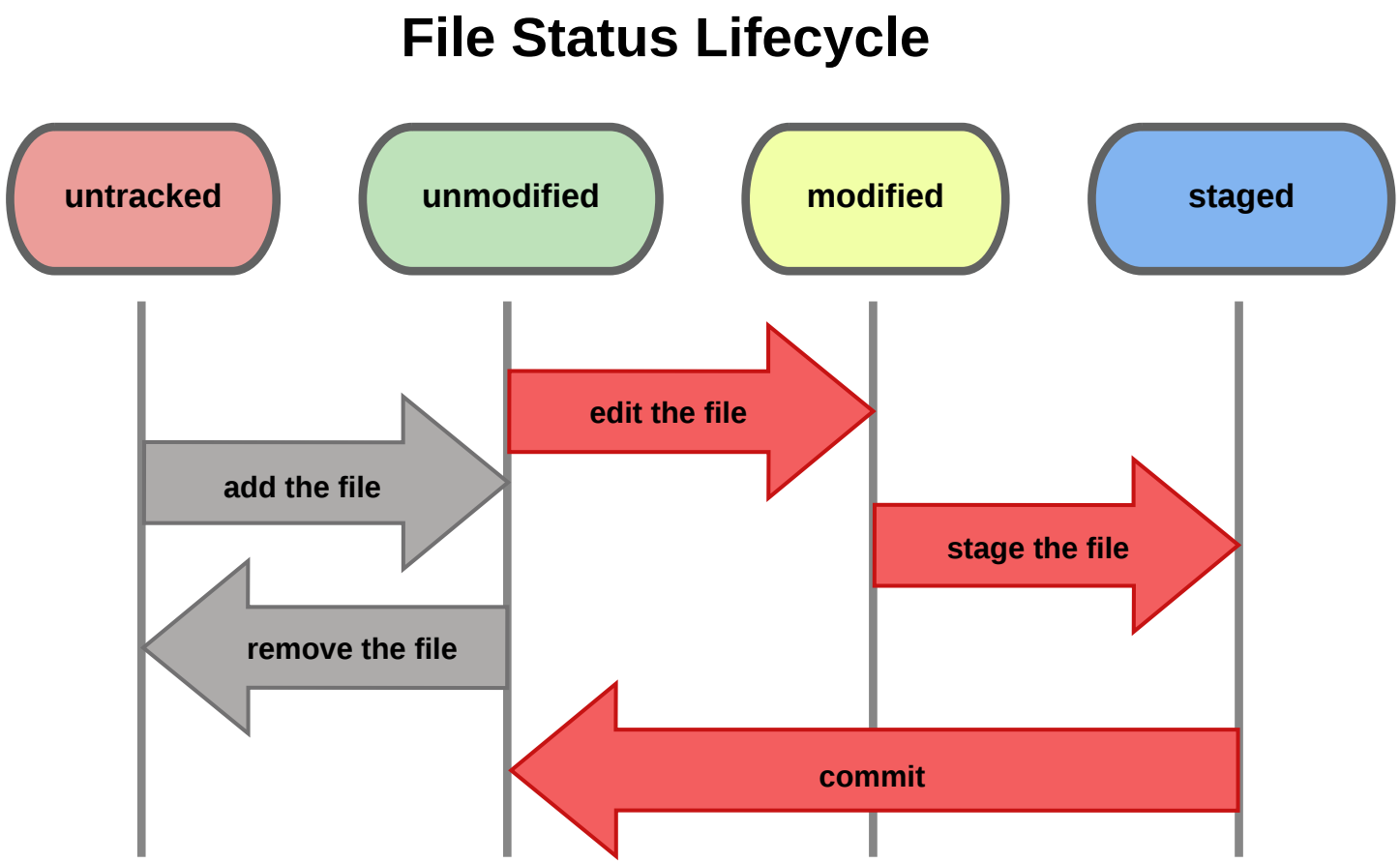
Links

- http://lihuen.linti.unlp.edu.ar/index.php?title=Workflow_Git/SVN
- http://lihuen.linti.unlp.edu.ar/index.php?title=Git

Secciones de Git

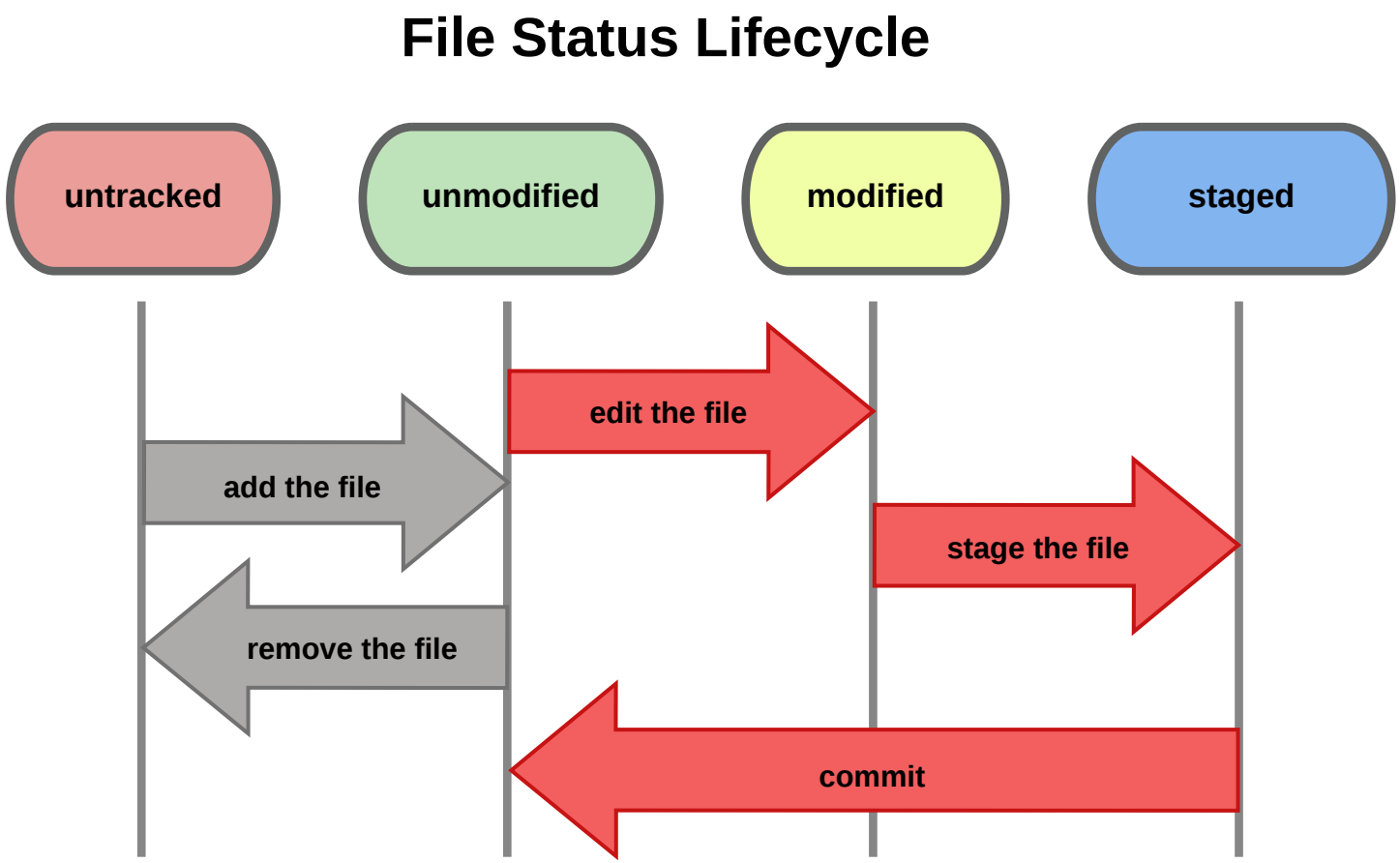


Ciclo de vida de los estados de los archivos



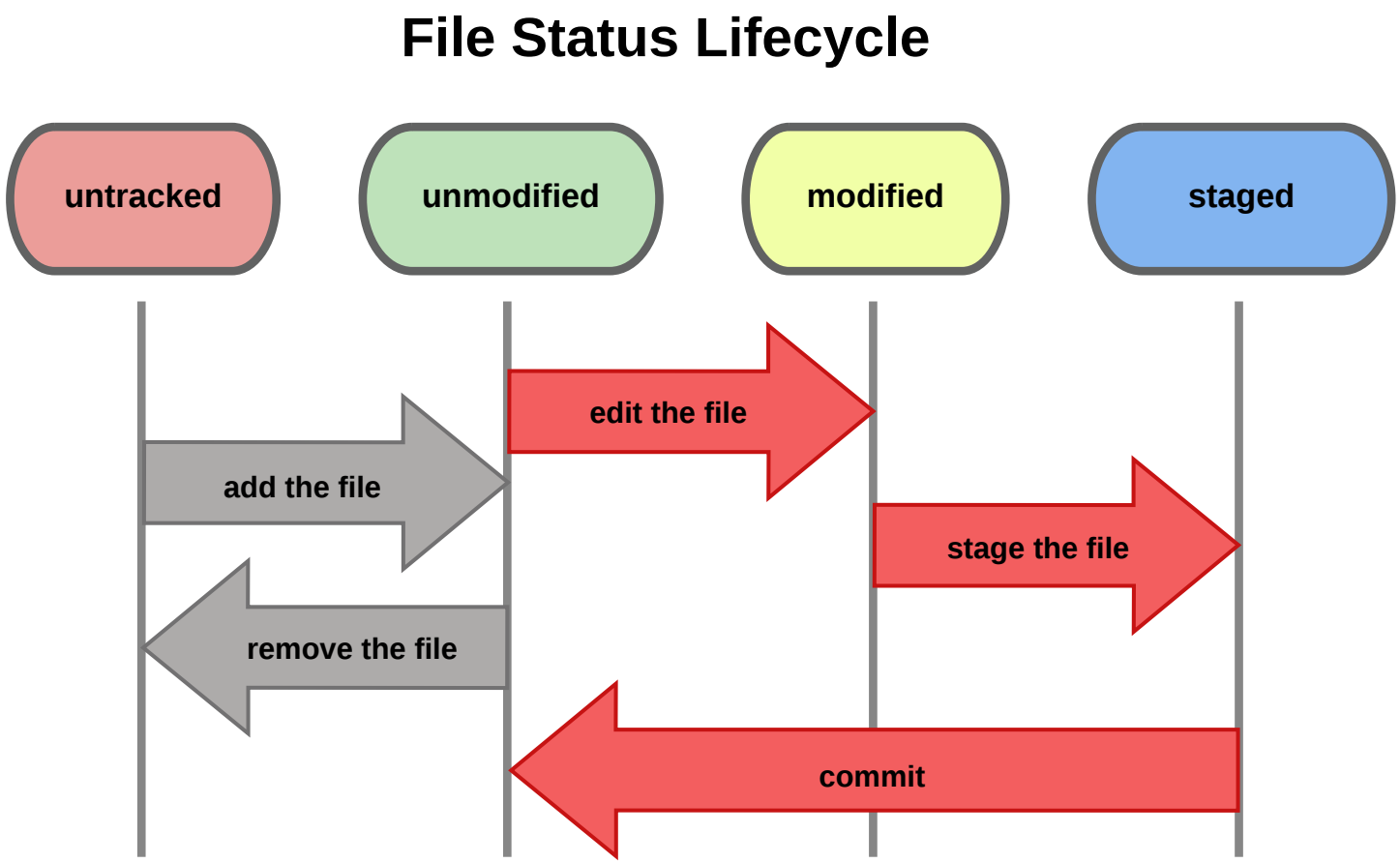
`git add` → pasa de untracked o de modified a staged

Ciclo de vida de los estados de los archivos



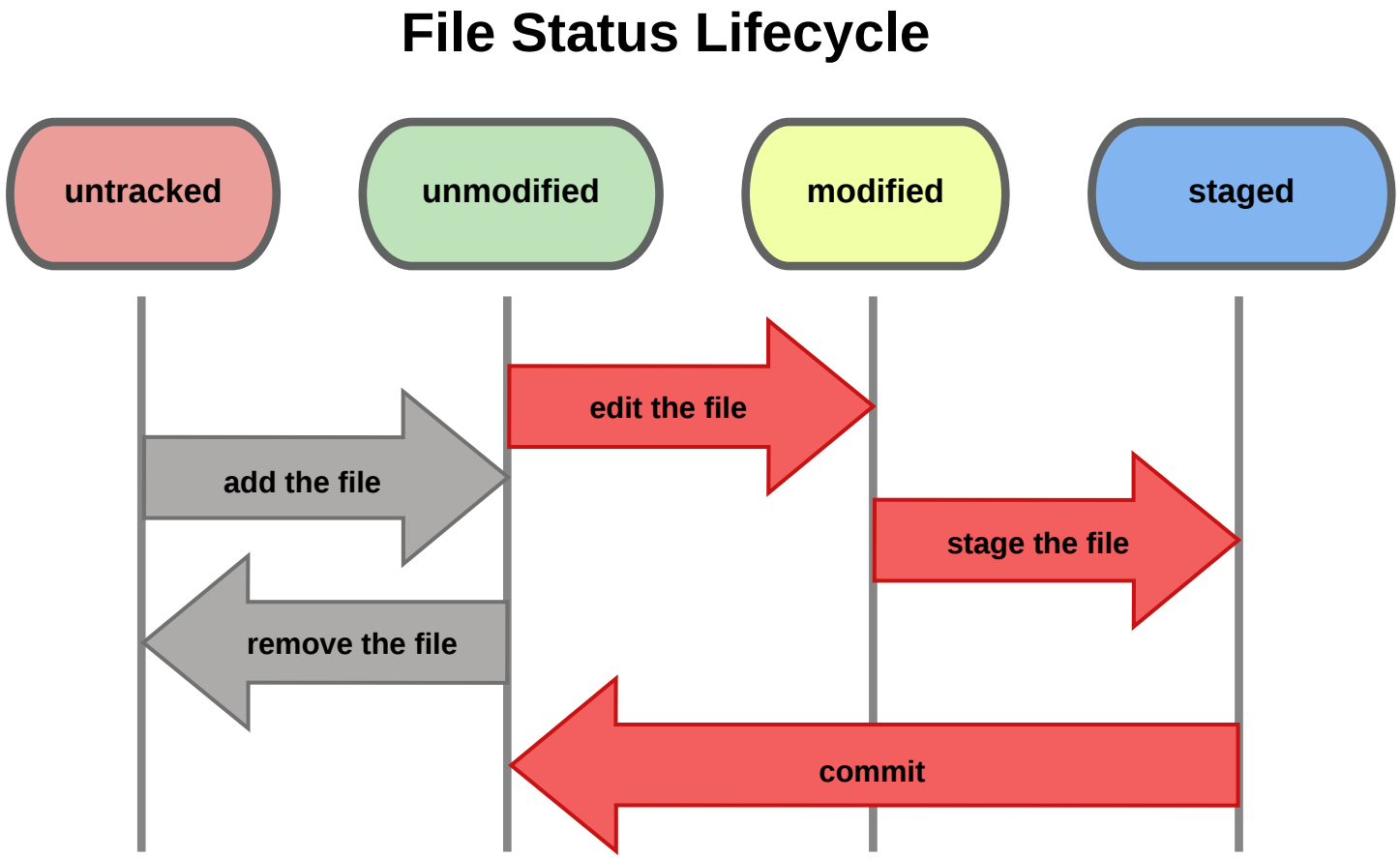
`git unstage` → pasa de staged a untracked o modified

Ciclo de vida de los estados de los archivos



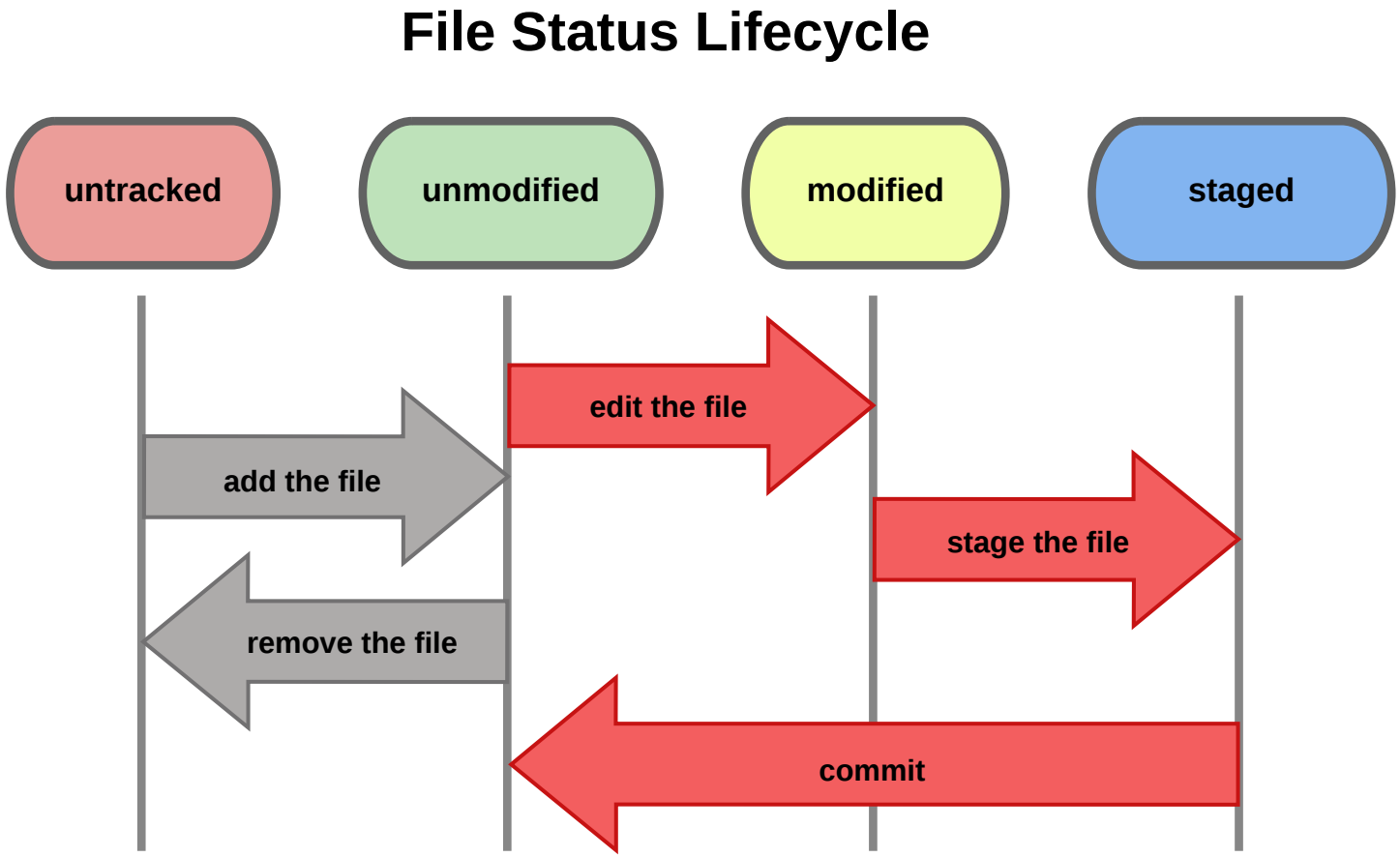
`git commit` → pasa de staged a unmodified.

Ciclo de vida de los estados de los archivos



Los archivos unmodified son los únicos correctamente "commited".

Ciclo de vida de los estados de los archivos



Recordar que `git push` solamente enviará al servidor las versiones que estén "commited".

Ejemplos de status

Local:

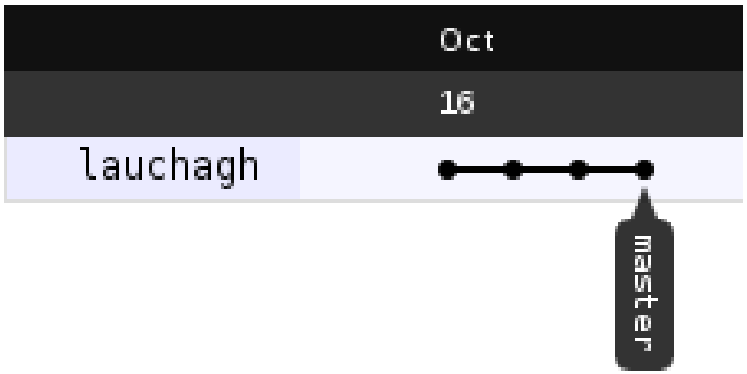
- Untracked files
- Changes not staged for commit
- Changes to be commited

Con remotos:

- Your branch is **ahead** of 'origin/master' by 1 commit.
  → `push`
- Your branch is **behind** 'origin/master' by 1 commit...
  → `pull`
- Your branch and 'origin/master' **have diverged**, and have 1 and 1 different commit(s) each, respectively. → `pull` y después `push`

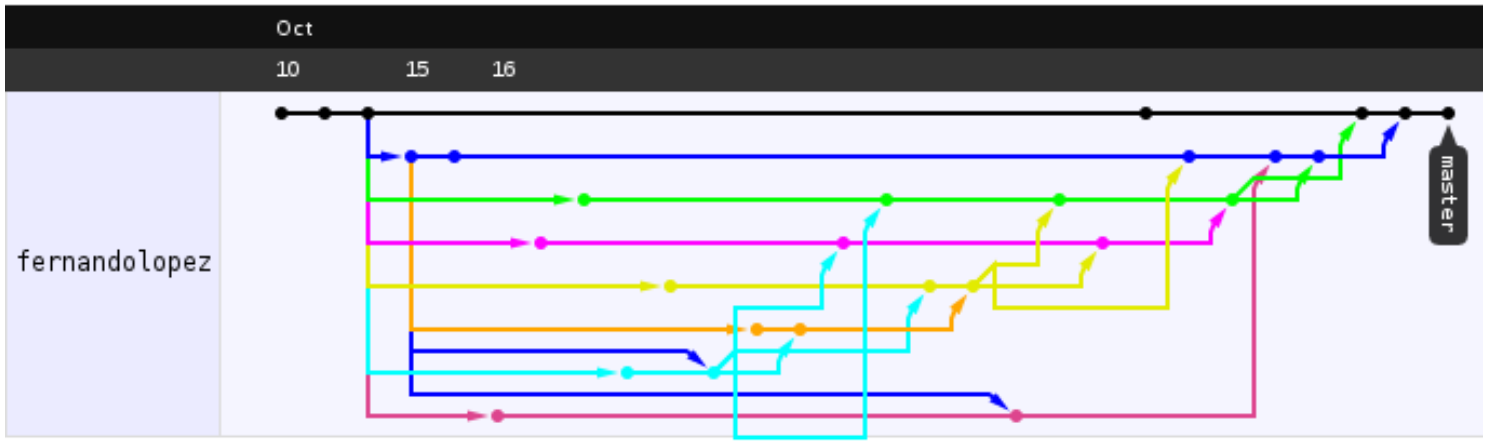
Branches
o ramas

- En el proyecto `octocats` nuestro árbol tenía una sola rama: `master`.



Branches
Infierno de merges

- Técnicamente en `octocat_viewer` también hay un solo branch: `master`.
- Pero hay distintas variantes del mismo:



- Todas esas variantes son distintas versiones de `master`
- Es díficil ver que cambio cada desarrollador y la historia está llena de merges.

Branches
ahora sí

- `git branch <nombre>` → Crear un branch.
- `git checkout <nombre>` → Cambia a un branch.
- `git diff <nombre>` → Diferencia entre el branch actual y el indicado.
- `git merge <nombre>` → Incorpora los cambios del branch indicado en el actual.
- Selección de revisión → muchos comandos reciben cómo argumento un "tree-ish".
- Tree-ish → branch, tag, hash de commit, etc...

Tree-ishes:

http://git-scm.com/book/en/Git-Tools-Revision-Selection

Branches
remotos

- Para ver todos los branches:
  `git branch -a`
- Para subir un branch a un remoto:
  `git push <remoto> <branch>`
- Para borrar un branch localmente:
  `git branch -d <branch>`
- Para borrar un branch en el servidor:
  `git branch <remoto> :<branch>`

Para investigar:

- `git push --tags`
- `git push --all`
- `git push --mirror`

Introducción

http://lihuen.info.unlp.edu.ar/index.php?title=Workflow_Git/SVN

- Metodologías de trabajo.
- Distintas variantes:
  - GitHub workflow.
  - Gitflow.
  - ¿Otras?
- Conceptos comunes:
  - `master` tiene que ser instalable.
  - Dividir el trabajo en features.
  - Un branch por cada feature.
  - Hacer merge lo más pronto posible para evitar conflictos.

GitHub workflow

- Simple.
- Feature branches.
- Commits y push a los feature branches todo el tiempo.
- Pull request para integrar con el proyecto.
- Quién acepta el pull request debe revisar el código.
- Ideal para productos con entrega continua.
- No es práctico sin testing automático ni peer review.

GitFlow

- Ligeramente más complejo.
- Branch `develop`.
- En `master` solamente commits con tags.
- Feature branches.
- Integración con `develop`.
- Preparar release: `release-1.0` (solo bugfixes).
- Versión liberada mergear `release-1.0` en `master` y crear el tag `1.0`.
- Ideal para productos orientados a release.

GitFlow simplificado

Consideremos en lo que queda de la clase la siguiente variante de GitFlow:

- Se modifica `master` solamente para crear un tag.
- Se desarrolla en los feature branches.
- Los feature branches se integran con `develop` frecuentemente.
- `develop` se integra con `master` al estabilizarse.

Mantener los branches actualizados

- Bajo cualquier workflow.
- Supongamos el siguiente caso:

```
          +-+--+ (feature-nuevo-formato)
         /  D  F
+--+---+----+----+ (develop)
A  B   C    E    G
```

- Usar rebase:

```
git checkout feature-formato-nuevo
git rebase develop
git add archivo-conflicto
git rebase --continue
```

- Entonces el grafo se transforma en:

```
                    +-+--+ (feature-nuevo-formato)
                   /  D' F'
+--+---+----+----+ (develop)
A  B   C    E    G
```

Integrando cambios a develop

- Hacerlo periódicamente.
- `develop` no tiene que ser perfecto pero si funciona mejor.
- Dado el caso anterior:

```
          +-+--+ (feature-nuevo-formato)
         /  D' F'
-+----+ (develop)
 E    G
```

```
git checkout develop
git merge feature-nuevo-formato
```

- El grafo queda:

```
-+----+---+--+ (develop, feature-nuevo-formato)
 E    G   D' F'
```

Creando una release


```
git checkout master
git merge develop
git tag 4.2
git push origin --all
git push origin --tags
```

Ejercicio GitFlow simplificado

Iteración 1

- Clonar el repositorio:
  `git@gitlab.linti.unlp.edu.ar:lihuen/remotebot4js.g`
  o
  `git@github.com:fernandolopez/RemoteBot4JS.git`
- Cada uno debe crear un feature branch a partir de develop `develop`.
- Desarrollar la feature manteniendo dentro de lo posible el branch actualizado con `git rebase develop`.
- Al terminar hacer merge con `develop` y eliminar el feature-branch.
- Cuando todos terminen un miembro debe crear una release.

Ejercicio GitFlow simplificado

Iteración 2

- Crear el segundo feature branch.
- Branch actualizado con `git rebase develop`.
- Los cambios que funcionen se pueden mergear en `develop`.
- Al terminar hacer merge con `develop` y eliminar el feature-branch.
- Cuando todos terminen un miembro debe crear una release.

Licencia

Fuentes:

- http://git-scm.com/book
- http://git-scm.com
- http://nvie.com/posts/a-successful-git-branching-model/
- https://wiki.diasporafoundation.org/Git_Workflow#Branching_model
- http://lihuen.linti.unlp.edu.ar/index.php?title=Workflow_Git/SVN