
git

# Pro Git Inglés

is.gd/progit_en

# Pro Git Español

is.gd/progit_es

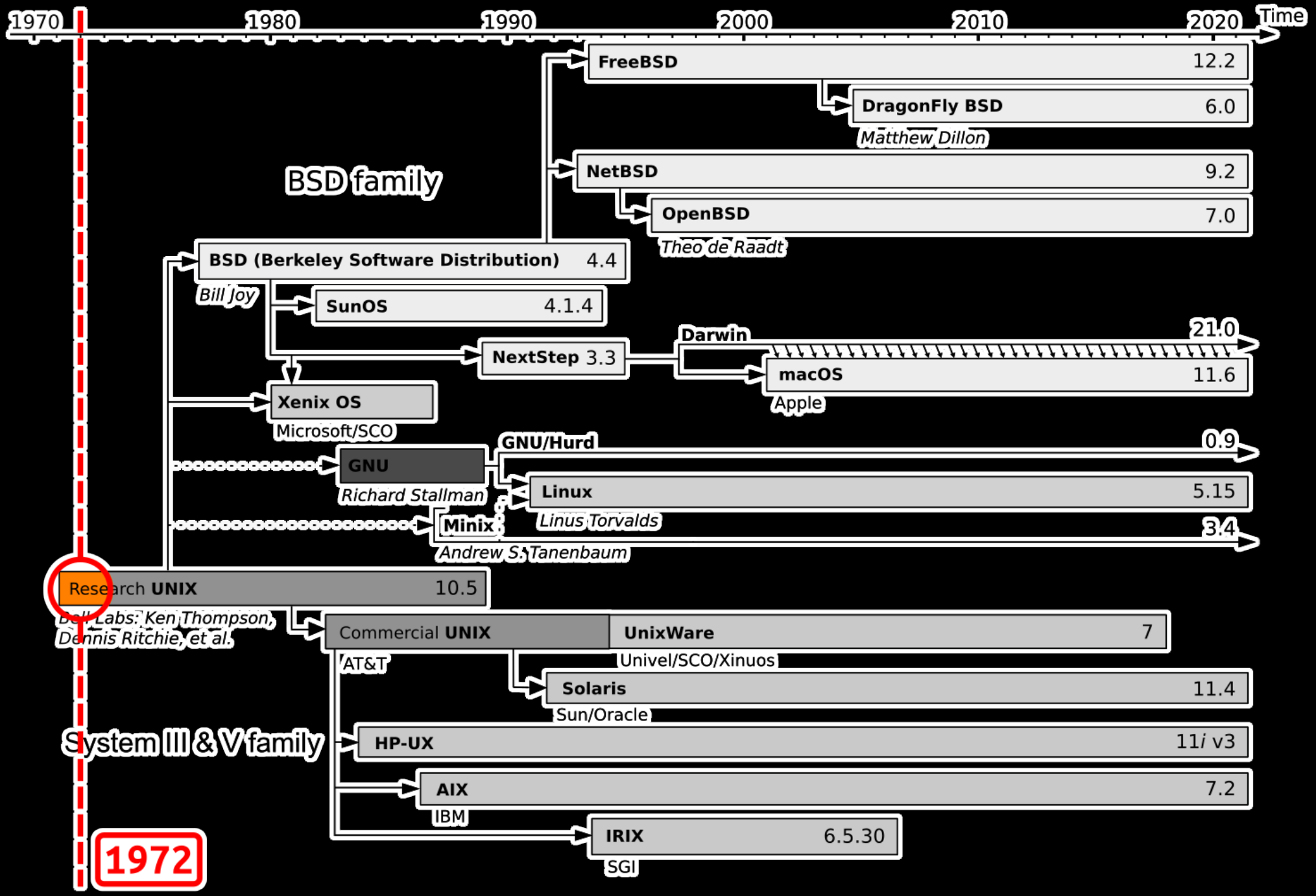
# Capítulos del Libro

Leer:

1. Sobre el Control de Versiones
2. Fundamentos de Git
3. Ramificaciones en Git
5. Git en entornos distribuidos

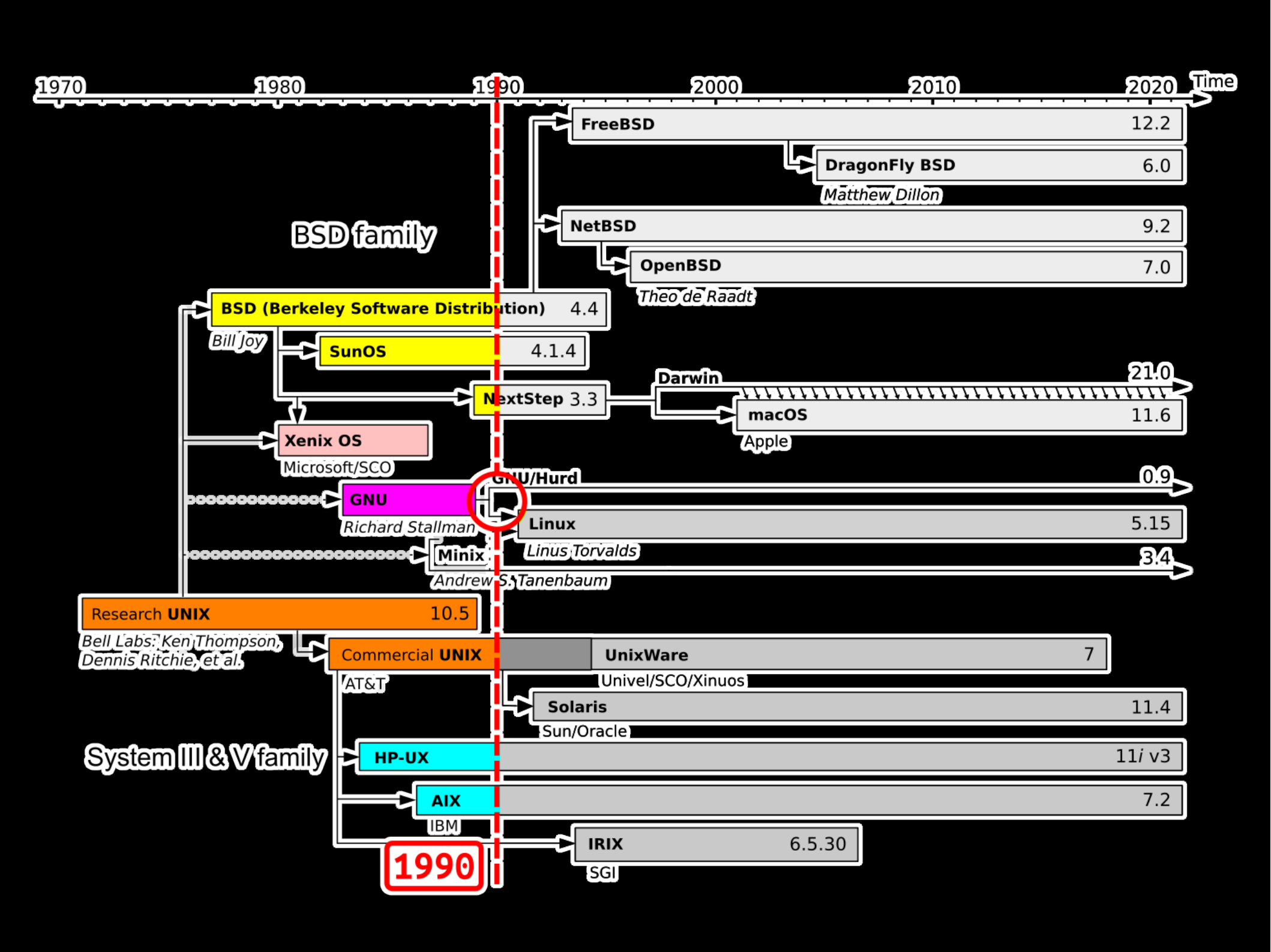
Recomendados:

6. GitHub
7. Herramientas de Git


1970
1980
1990
2000
2010
2020
Time
BSD family
FreeBSD 12.2
DragonFly BSD 6.0
Matthew Dillon
NetBSD 9.2
OpenBSD 7.0
Theo de Raadt
BSD (Berkeley Software Distribution) 4.4
Bill Joy
SunOS 4.1.4
NextStep 3.3
Darwin 21.0
macOS 11.6
Apple
Xenix OS
Microsoft/SCO
GNU/Hurd 0.9
GNU
Richard Stallman
Linux 5.15
Linus Torvalds
Minix 3.4
Andrew S. Tanenbaum
Research UNIX 10.5
Bell Labs: Ken Thompson, Dennis Ritchie, et al.
Commercial UNIX
UnixWare 7
Univel/SCO/Xinuos
AT&T
Solaris 11.4
Sun/Oracle
System III & V family
HP-UX 11i v3
AIX 7.2
IBM
IRIX 6.5.30
SGI
1972

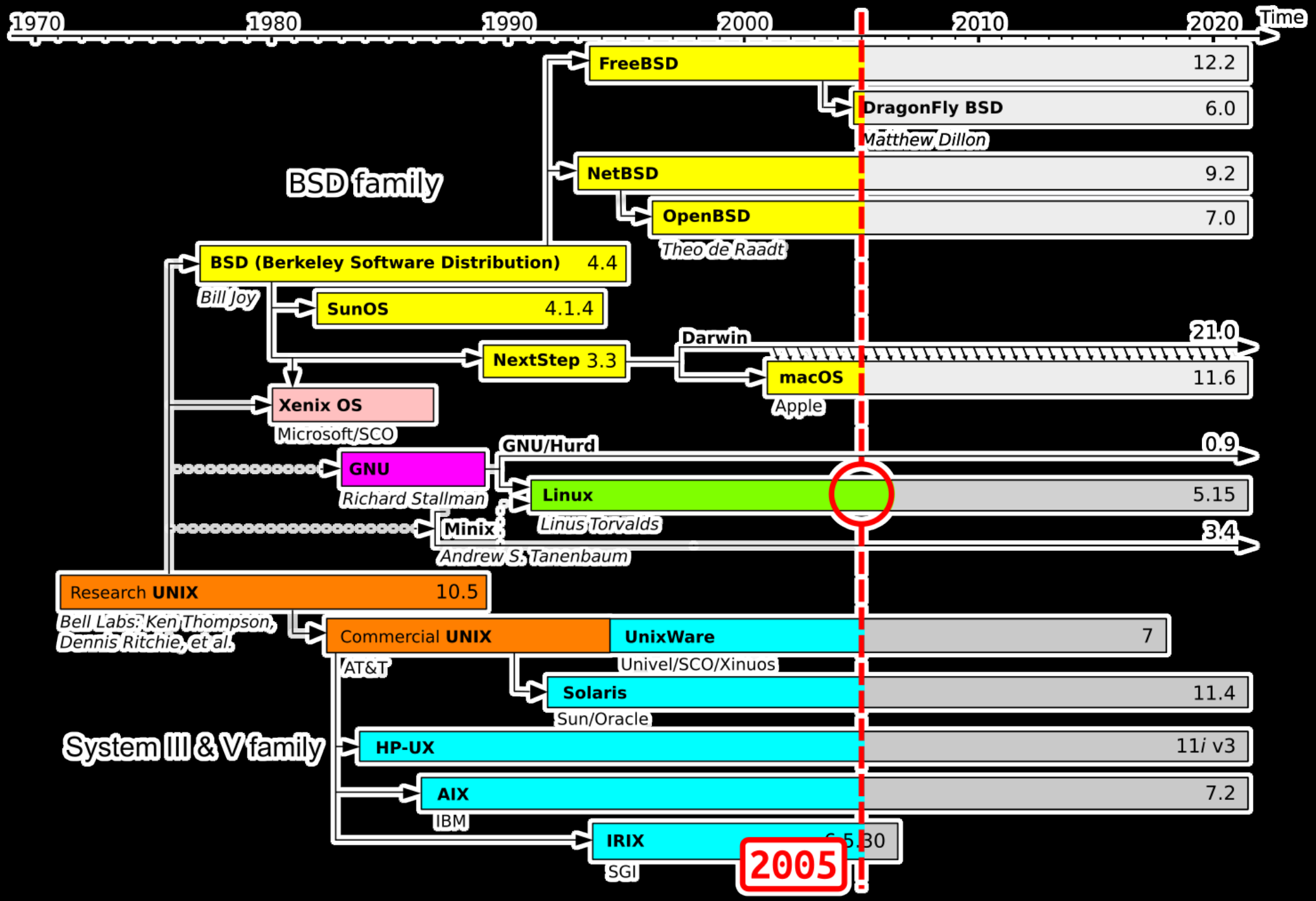
# 1972

- **1972 - SCCS** (Source Code Control System): Desarrollado por Marc Rochkind en Bell Labs para la computadora IBM System/370 que ejecutaba OS/360. En 1973 se reescribió en C para usarse en los sistemas UNIX (en ése momento una PDP-11).
- **1982 - RCS** (Revision Control System): Desarrollado por Walter F. Tichy. Una innovación de RCS fue la de los deltas inversos, que permiten regresar a una versión anterior de un archivo.


1970
1980
1990
2000
2010
2020
Time
BSD family
FreeBSD 12.2
DragonFly BSD 6.0
Matthew Dillon
NetBSD 9.2
OpenBSD 7.0
Theo de Raadt
BSD (Berkeley Software Distribution) 4.4
Bill Joy
SunOS 4.1.4
NextStep 3.3
Darwin 21.0
macOS 11.6
Apple
Xenix OS
Microsoft/SCO
GNU/Hurd 0.9
GNU
Richard Stallman
Linux 5.15
Linus Torvalds
Minix 3.4
Andrew S. Tanenbaum
Research UNIX 10.5
Bell Labs: Ken Thompson, Dennis Ritchie, et al.
Commercial UNIX
AT&T
UnixWare 7
Univel/SCO/Xinuos
Solaris 11.4
Sun/Oracle
System III & V family
HP-UX 11i v3
AIX 7.2
IBM
IRIX 6.5.30
SGI
1990

# 1990

- **1990 - CVS** (Concurrent Versions System): Desarrollado por Dick Grune. Opera como front end de RCS. Agrega soporte para registrar cambios a nivel de repositorio y el modelo cliente servidor, donde un servidor guarda las versiones actuales del proyecto y su historial.
- **2000 - SVN** (Apache Subversion): Creado por CollabNet Inc. Su objetivo fue el de ser un sucesor compatible de CVS.


1970
1980
1990
2000
2010
2020
Time
BSD family
FreeBSD
12.2
DragonFly BSD
6.0
Matthew Dillon
NetBSD
9.2
OpenBSD
7.0
Theo de Raadt
BSD (Berkeley Software Distribution)
4.4
Bill Joy
SunOS
4.1.4
NextStep
3.3
Darwin
21.0
macOS
11.6
Apple
Xenix OS
Microsoft/SCO
GNU/Hurd
0.9
GNU
Richard Stallman
Linux
5.15
Linus Torvalds
Minix
3.4
Andrew S. Tanenbaum
Research UNIX
10.5
Bell Labs: Ken Thompson, Dennis Ritchie, et al.
Commercial UNIX
AT&T
UnixWare
7
Univel/SCO/Xinuos
Solaris
11.4
Sun/Oracle
System III & V family
HP-UX
11i v3
AIX
7.2
IBM
IRIX
6.5.30
SGI
2005

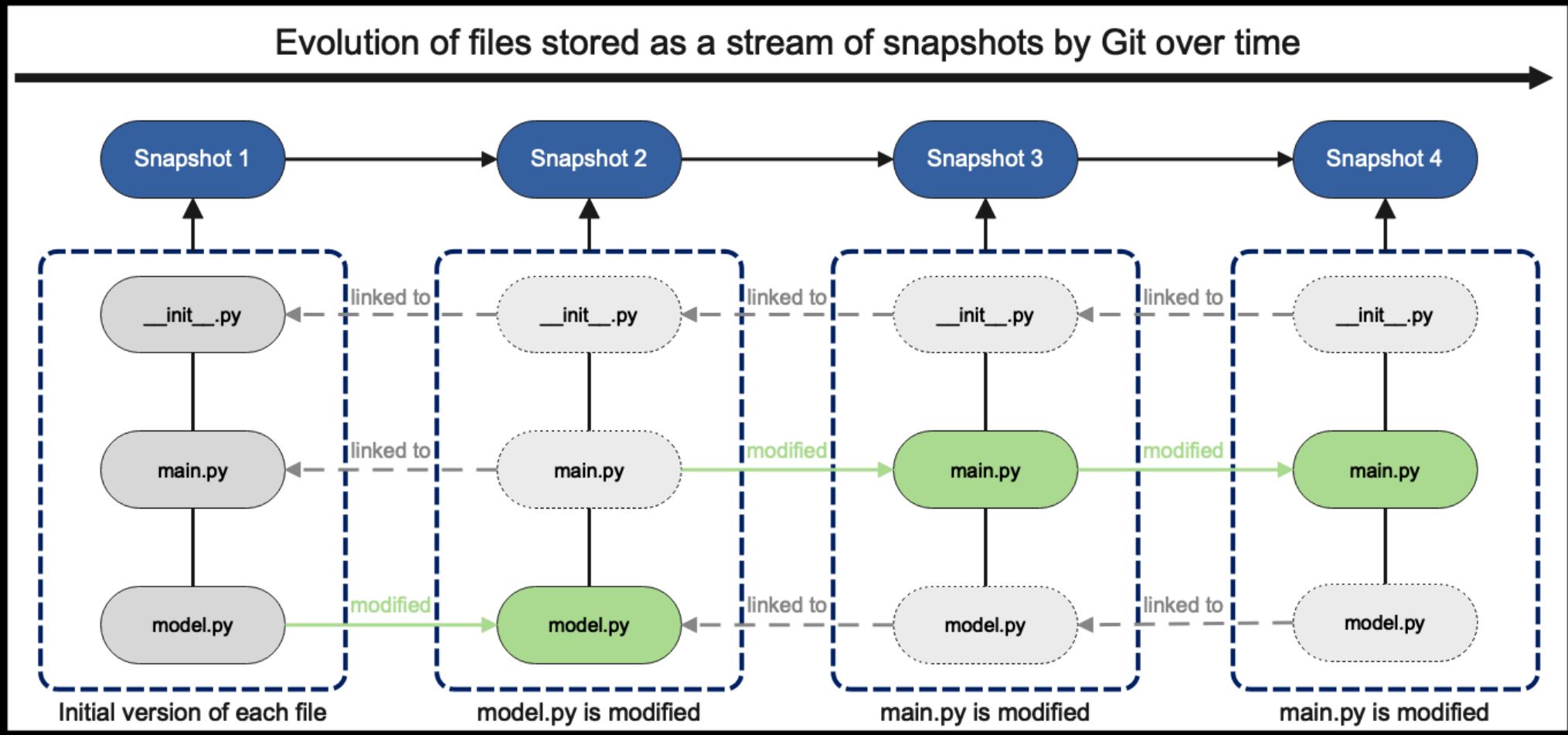
# 2005

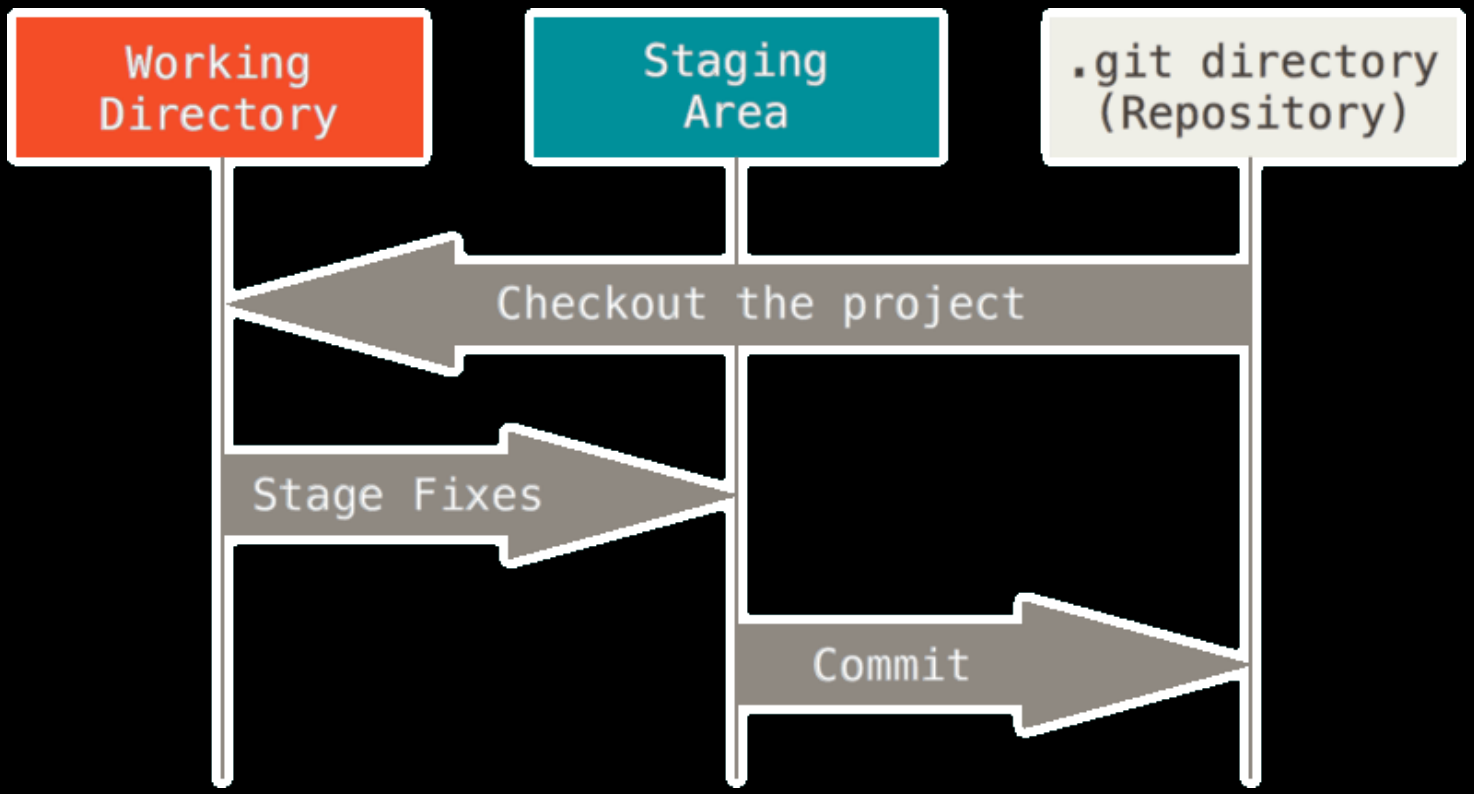
- **2005 - git**: Desarrollado por Linus Torvalds, luego de que BitKeeper revocara su licencia para el desarrollo de Linux (¡Gracias BitKeeper!). El esl sistema de control de version distribuido más popular, con cerca del 95% de los desarrolladores usandolo como su VCS principal.
- **2005 - Mercurial**: Desarrollado por Matt Mackall. Éste proyecto comenzó unos pocos días después del desarrollo de git y con el mismo objetivo en mente.

# DVCS

## Distributed Version Control Systems

(Sistemas de Control de Versiones Distribuidos)


Evolution of files stored as a stream of snapshots by Git over time
Snapshot 1
Snapshot 2
Snapshot 3
Snapshot 4
__init__.py
main.py
model.py
linked to
modified
Initial version of each file
model.py is modified
main.py is modified
main.py is modified

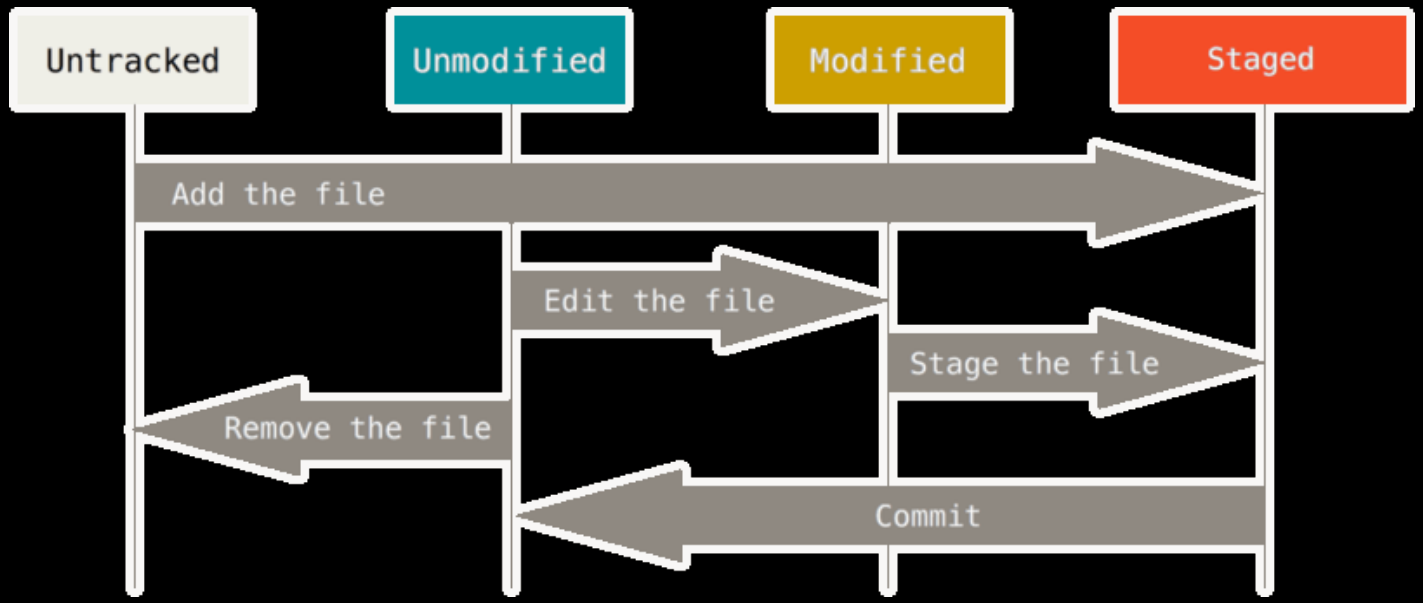

Working Directory
Staging Area
.git directory (Repository)
Checkout the project
Stage Fixes
Commit

# Configurando Git por primera vez

```
$ git config --global user.name "Natalia Natalia"
$ git config --global user.email natalia@natalia.com
$ git config --global core.editor vi
$ git config --list
```

# Inicializando un Repositorio

```
$ git status
$ git init
$ git status
$ git add FILES
$ git status
$ git commit -m 'First commit'
$ git status
```


Untracked
Unmodified
Modified
Staged
Add the file
Edit the file
Stage the file
Remove the file
Commit

# Guardando y Viendo Cambios

```
$ cat .gitignore
$ git diff [--staged|--cached] [--check]
$ git rm FILES
$ git mv ORIGIN DESTINY
$ git log
```

# Mensajes de Commits

```
Capitalized, short (50 chars or less) summary

More detailed explanatory text, if necessary.  Wrap it to about 72
characters or so.  In some contexts, the first line is treated as the
subject of an email and the rest of the text as the body.  The blank
line separating the summary from the body is critical (unless you omit
the body entirely); tools like rebase can get confused if you run the
two together.

Write your commit message in the imperative: "Fix bug" and not "Fixed bug"
or "Fixes bug."  This convention matches up with commit messages generated
by commands like git merge and git revert.

Further paragraphs come after blank lines.

- Bullet points are okay, too

- Typically a hyphen or asterisk is used for the bullet, followed by a
  single space, with blank lines in between, but conventions vary here

- Use a hanging indent
```

A Note About Git Commit Messages

# Alias de Git

```
$ git config --global alias.co checkout
$ git config --global alias.br branch
$ git config --global alias.ci commit
$ git config --global alias.st status
$ git config --global alias.unstage 'reset HEAD --'
```

# Colaborar Localmente

```
$ # 1.a - Create bare repository
$ mkdir /path/to/pp.git
$ cd /path/to/pp.git
$ git init --bare
$ # 1.b - Or clone bare repository
$ git clone --bare --local /from/pepe/.git /to/pp.git
$ # 2 - Clone (local) "bare" repository
$ cd /path/to/workspace/
$ git clone file:///absolute/path/to/pp.git pepa.git
$ git clone file:///absolute/path/to/pp.git pepi.git
$ cd pepa.git
$ git remote -v
```

# Parches

```
$ cd /path/to/project.git
$ # ... work and commit ...
$ # Create and apply the patch (option #1)
$ # Check gnu diff and patch commands
$ git diff > file.patch
$ git apply file.patch
$ # Create and apply the patch (option #2)
$ # Public Project over Email
$ git format-patch --find-renames origin/master
$ git am *.patch
```