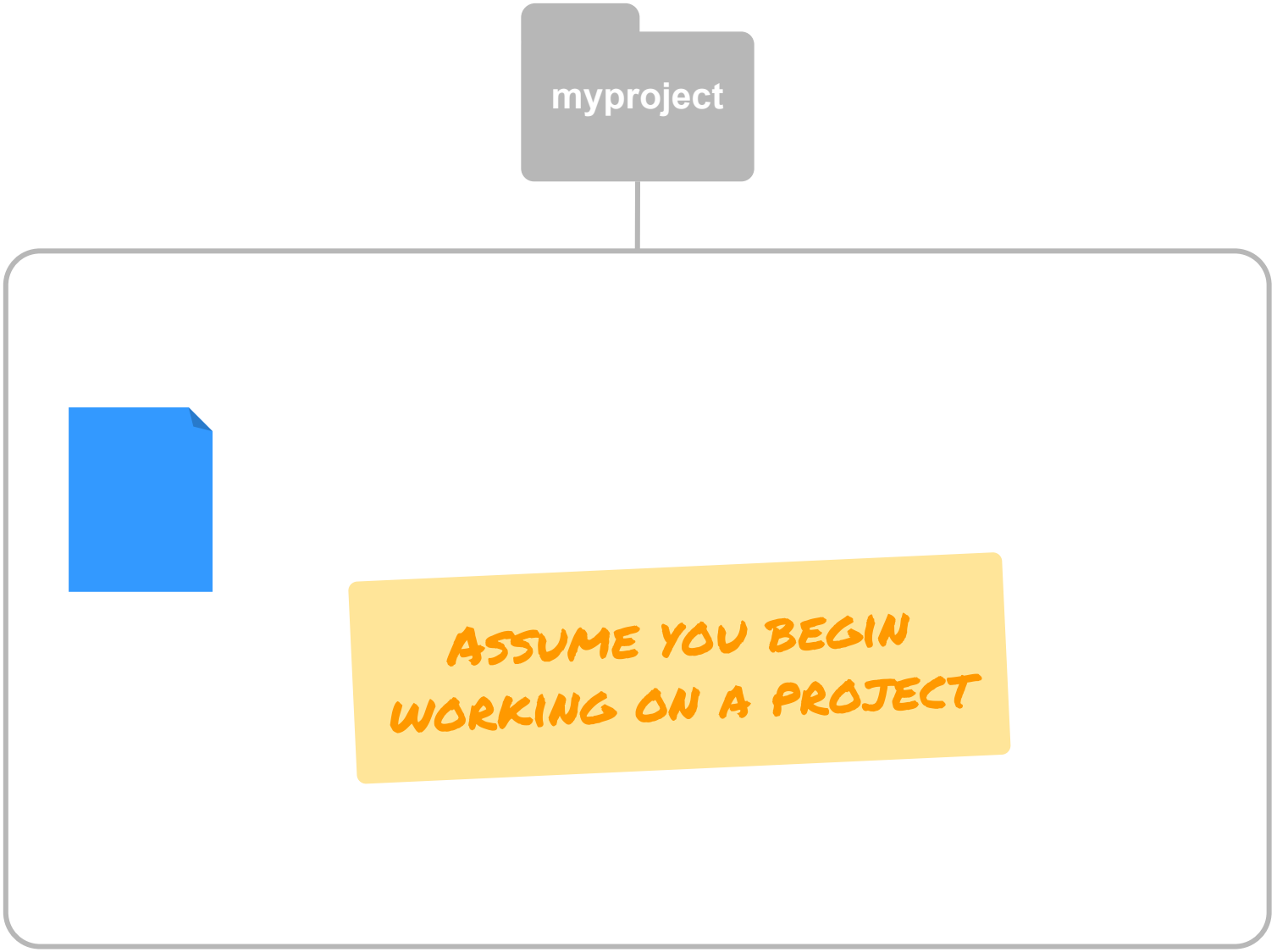
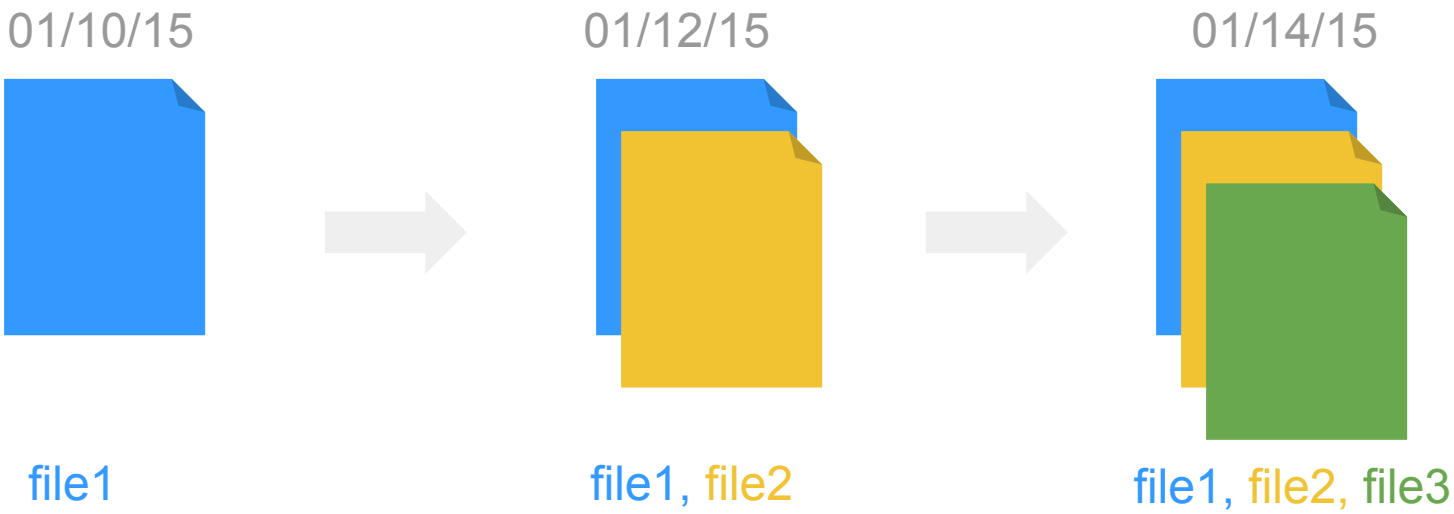
Git Basic Concepts 1

Stat 133 by Gaston Sanchez

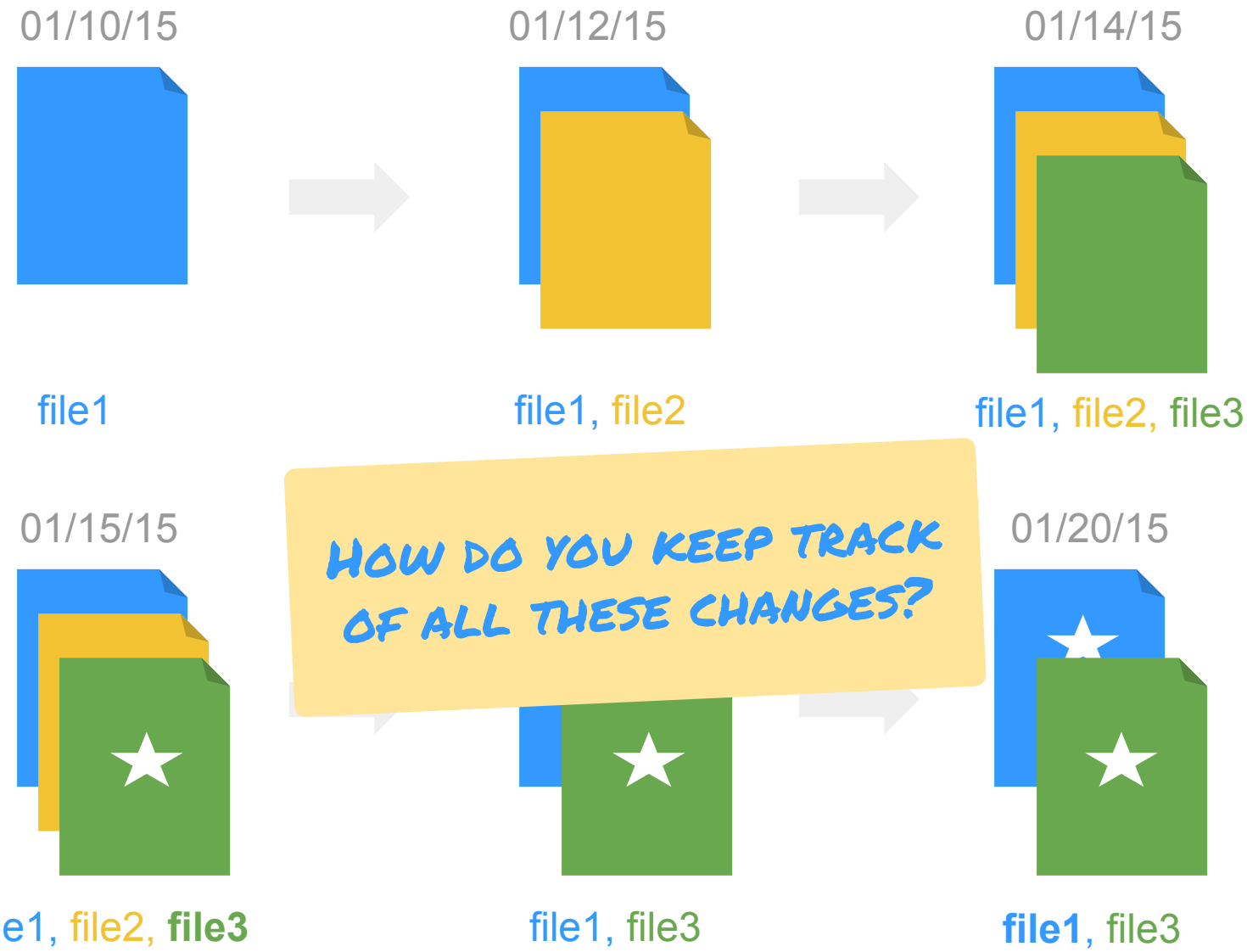
project's directory



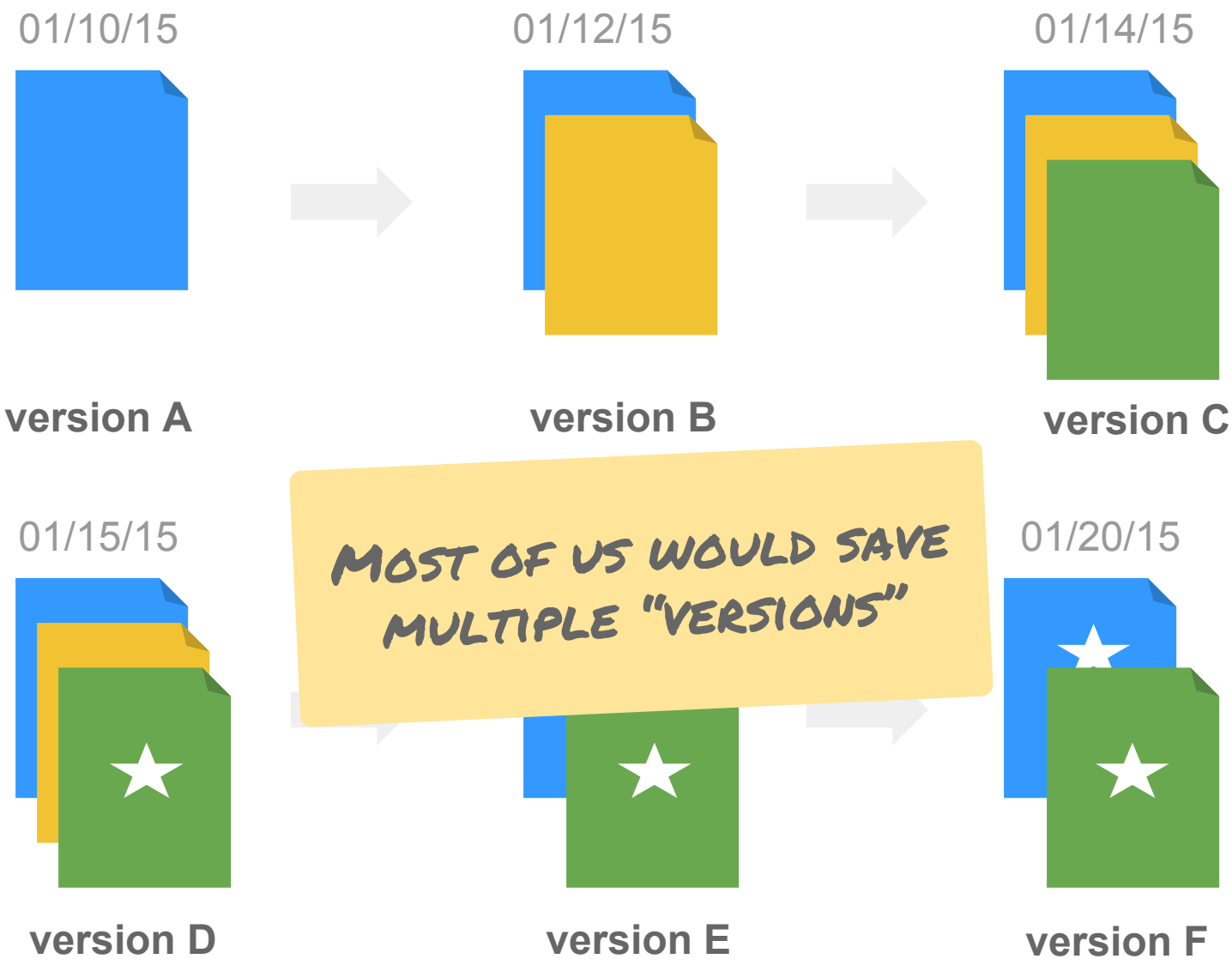
myproject


01/10/15
01/12/15
01/14/15
file1
file1, file2
file1, file2, file3
01/15/15
01/17/15
01/20/15
file1, file2, file3
file1, file3
file1, file3

myproject


01/10/15
01/12/15
01/14/15
file1
file1, file2
file1, file2, file3
01/15/15
How do you keep track of all these changes?
01/20/15
file1, file2, file3
file1, file3
file1, file3

myproject


01/10/15
01/12/15
01/14/15
version A
version B
version C
MOST OF US WOULD SAVE MULTIPLE "VERSIONS"
01/15/15
01/20/15
version D
version E
version F

Savings multiple "versions" is highly inefficient

Key Ideas

Keep a record of all the made changes

Storing changes of each version

Git is a Version Control System (VCS)

Version Control System

Keeps tracks of changes over time

Allows you track progress

Allows you to revert to earlier versions (dog can't eat your homework)

Makes it easier to collaborate with others

Consider some changes in a file

Saving a file 3 different times

-vs-

Saving snapshots of the changes

Saving a file 3 different times

file_v1

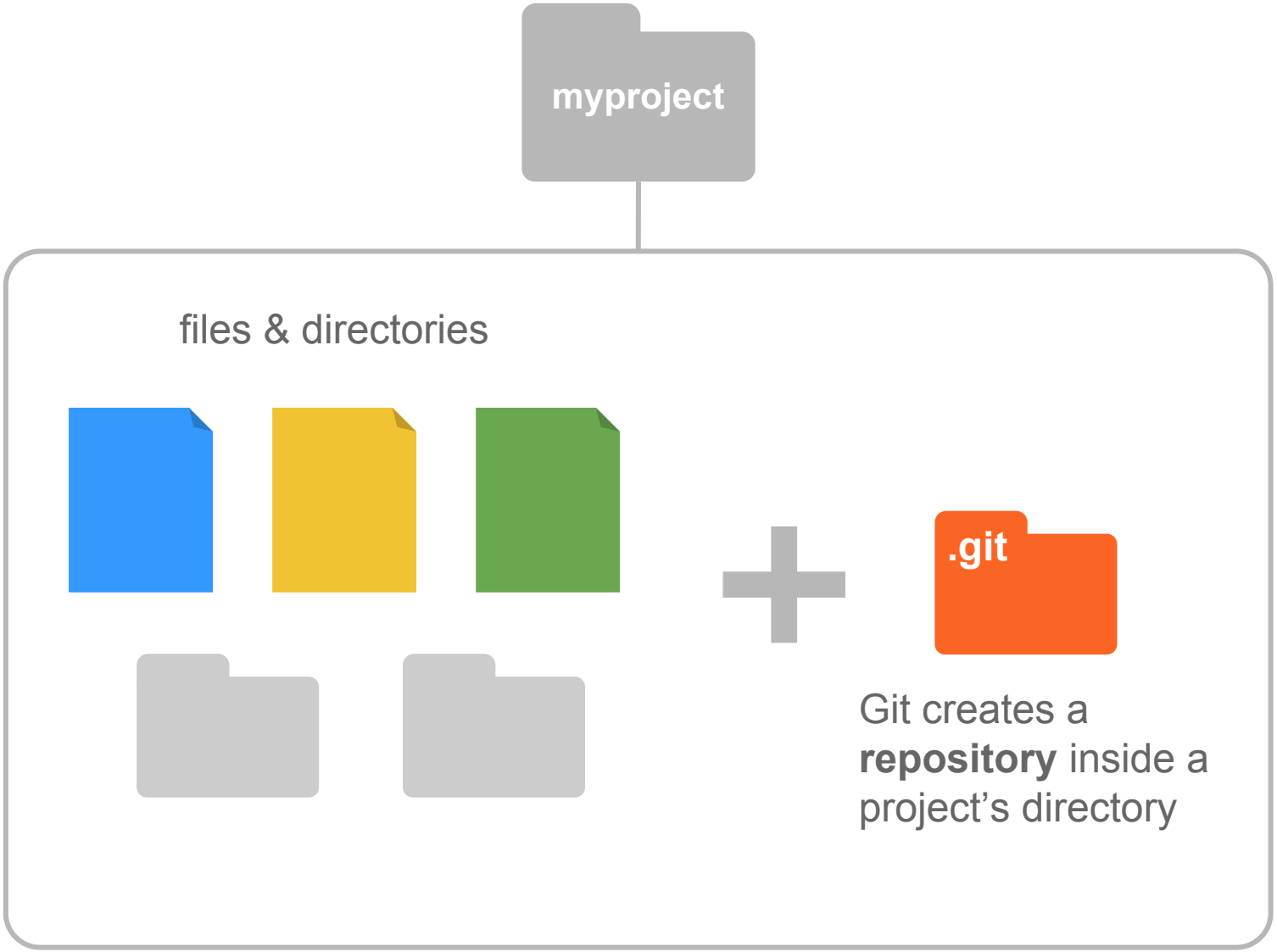
file_v2

file_v3

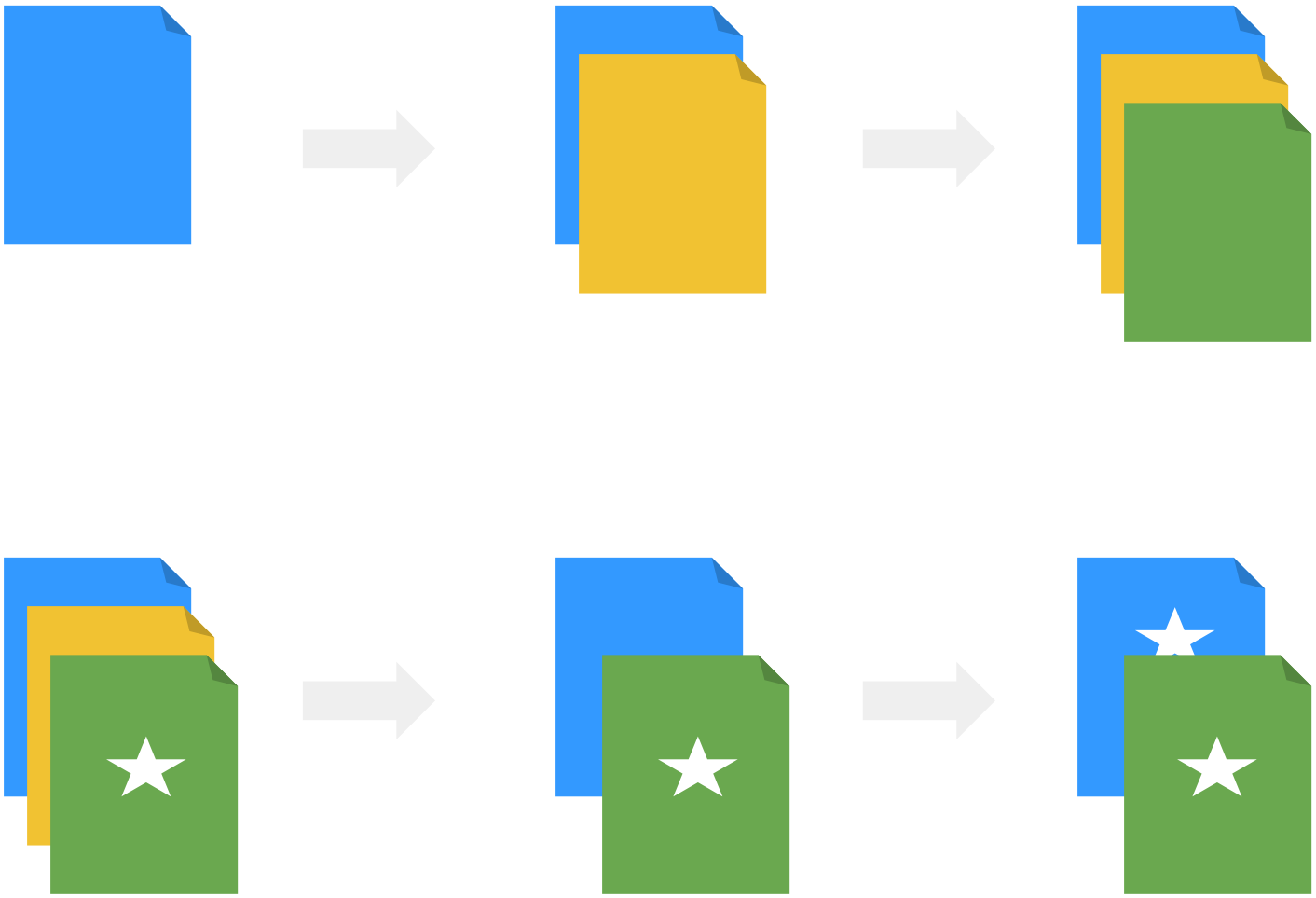
NOT VERY EFFICIENT

Saving snapshots of changes ...

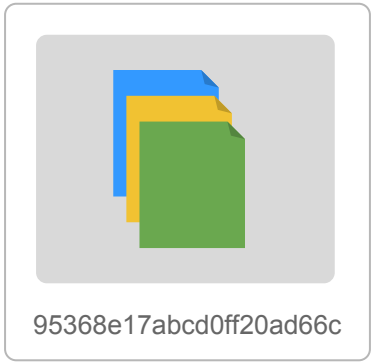
project's directory



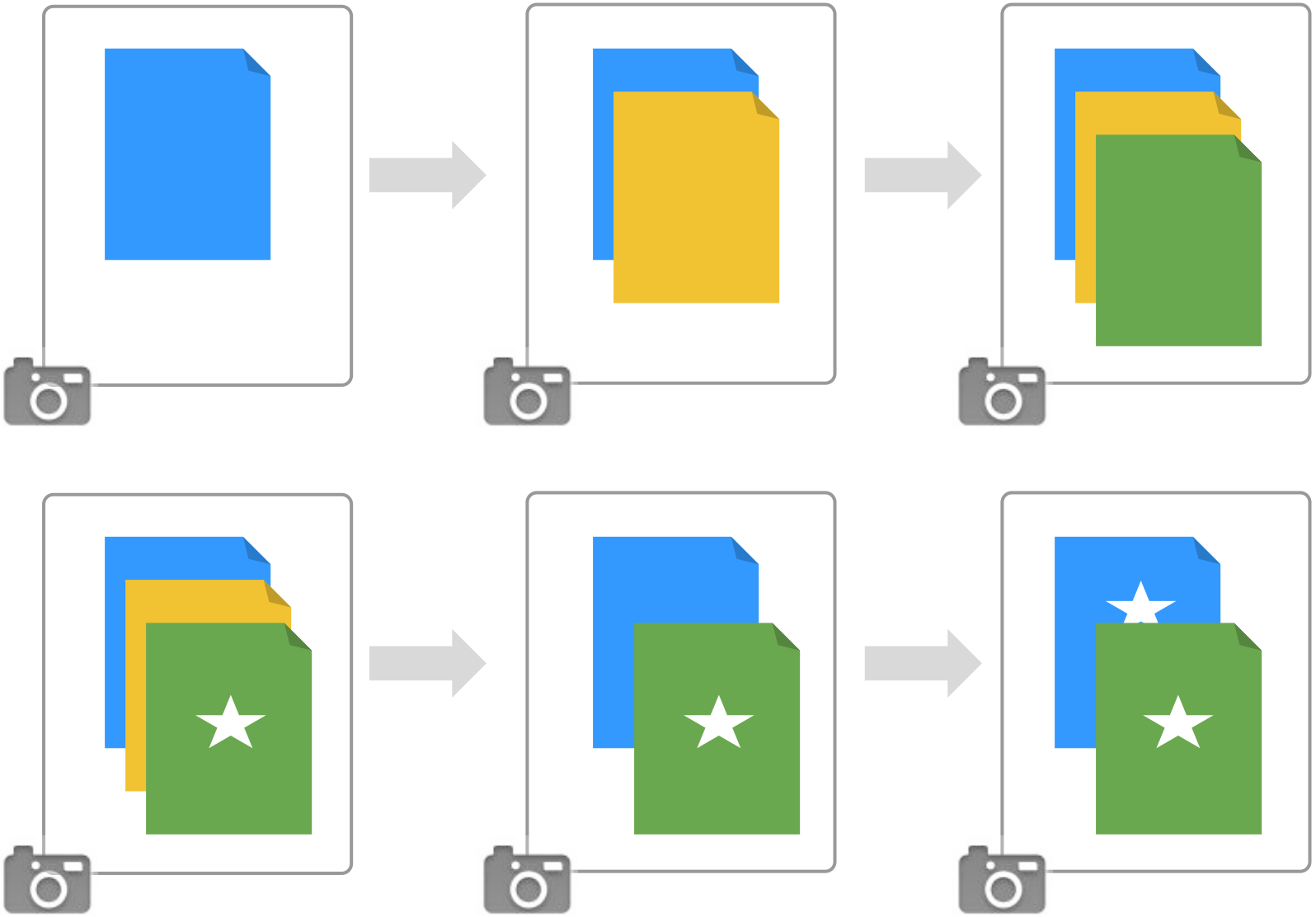
Project snapshots



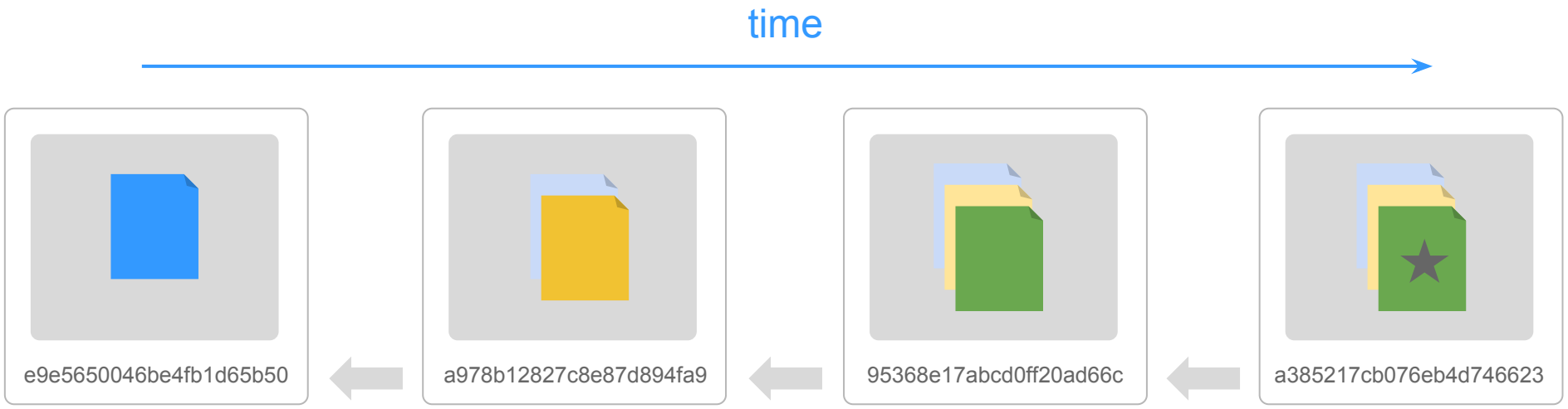
Git records the **changes** made on a project's files (not their versions) by taking "snapshots"



Project snapshots



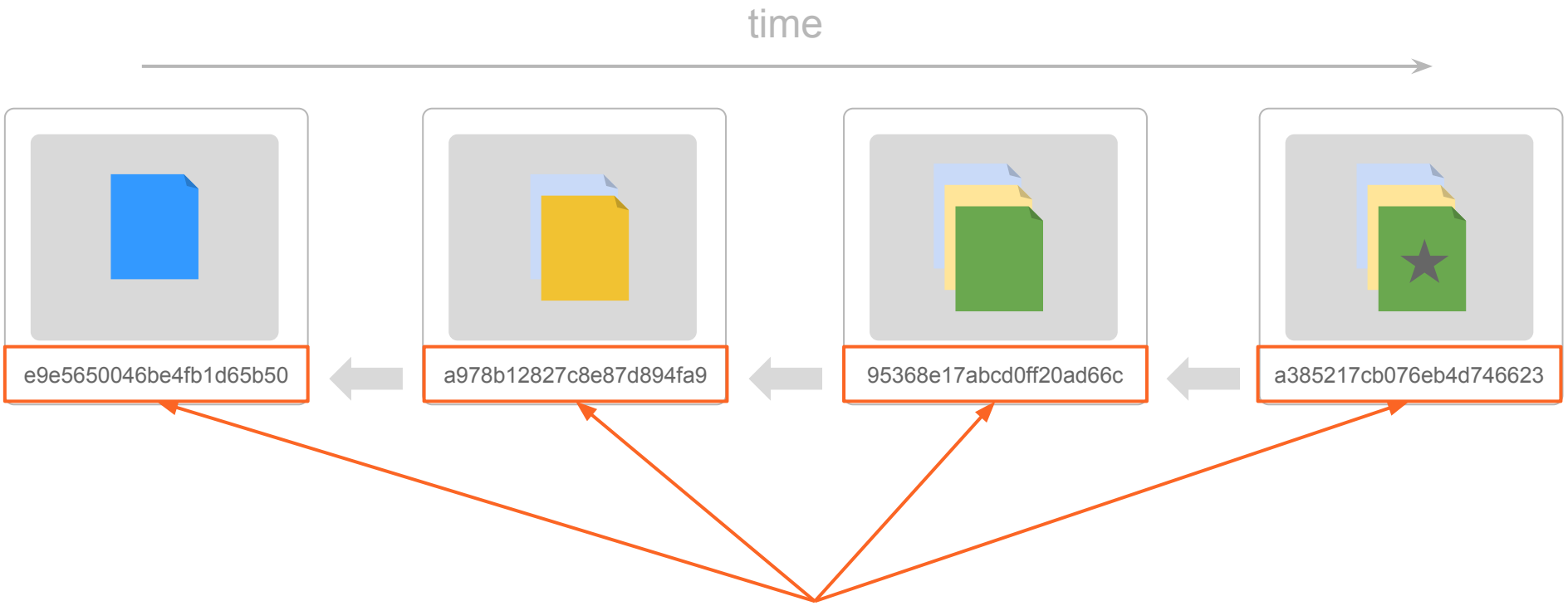
Git stores “snapshots”



A snapshot is a set of changes

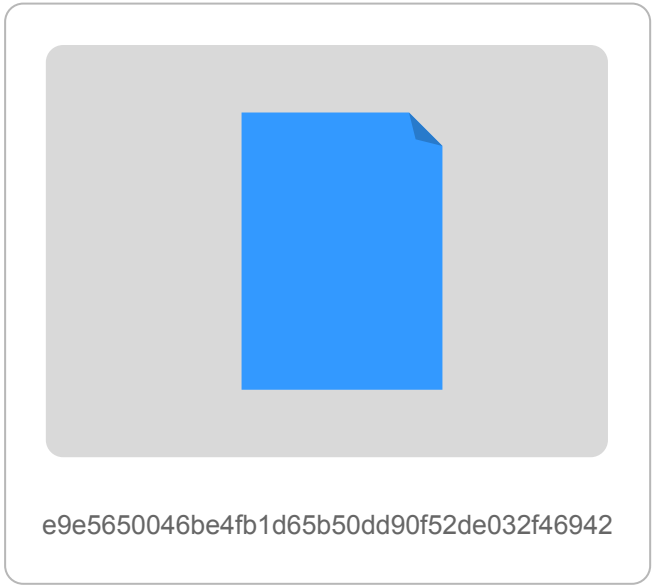
Each snapshot is known as a **commit**, i.e. a specific set of changes

Only new changes are tracked from one commit to the next one



Each commit (“snapshot”) has a unique ID or hash commit

SHA-1 values



`e9e5650046be4fb1d65b50dd90f52de032f46942`

SHA-1 value is 40-characters long

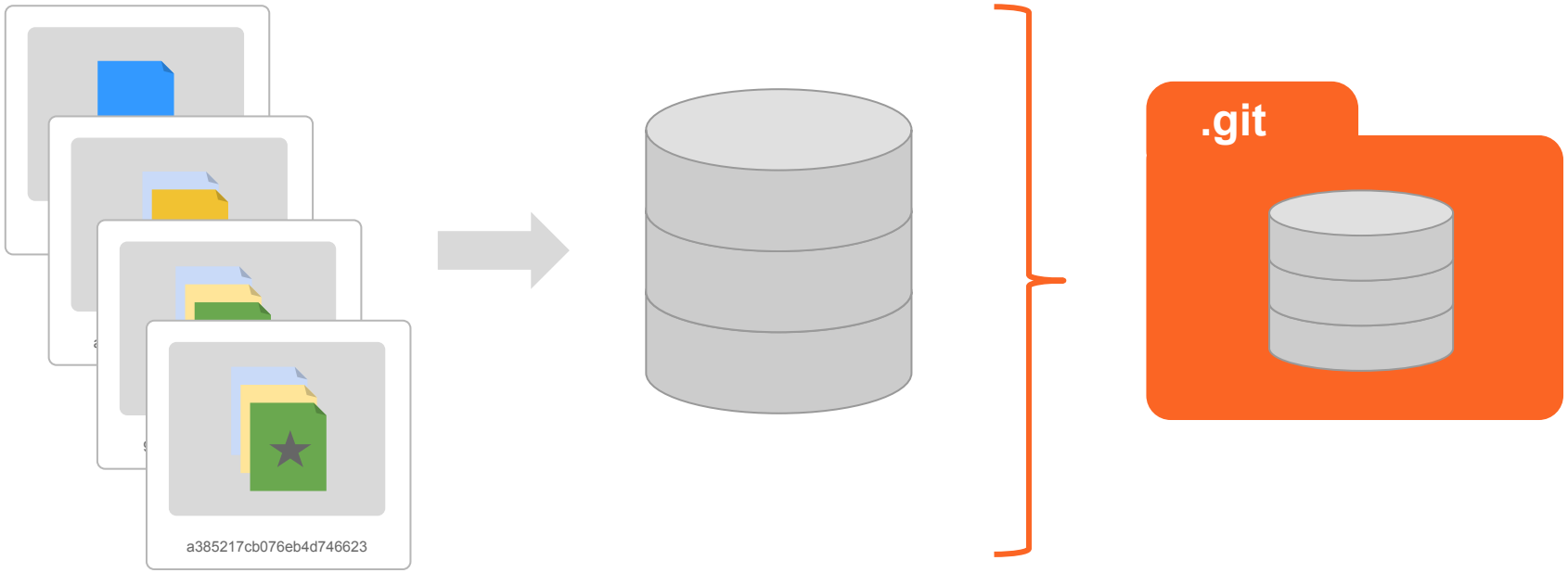
40 hexadecimal digits

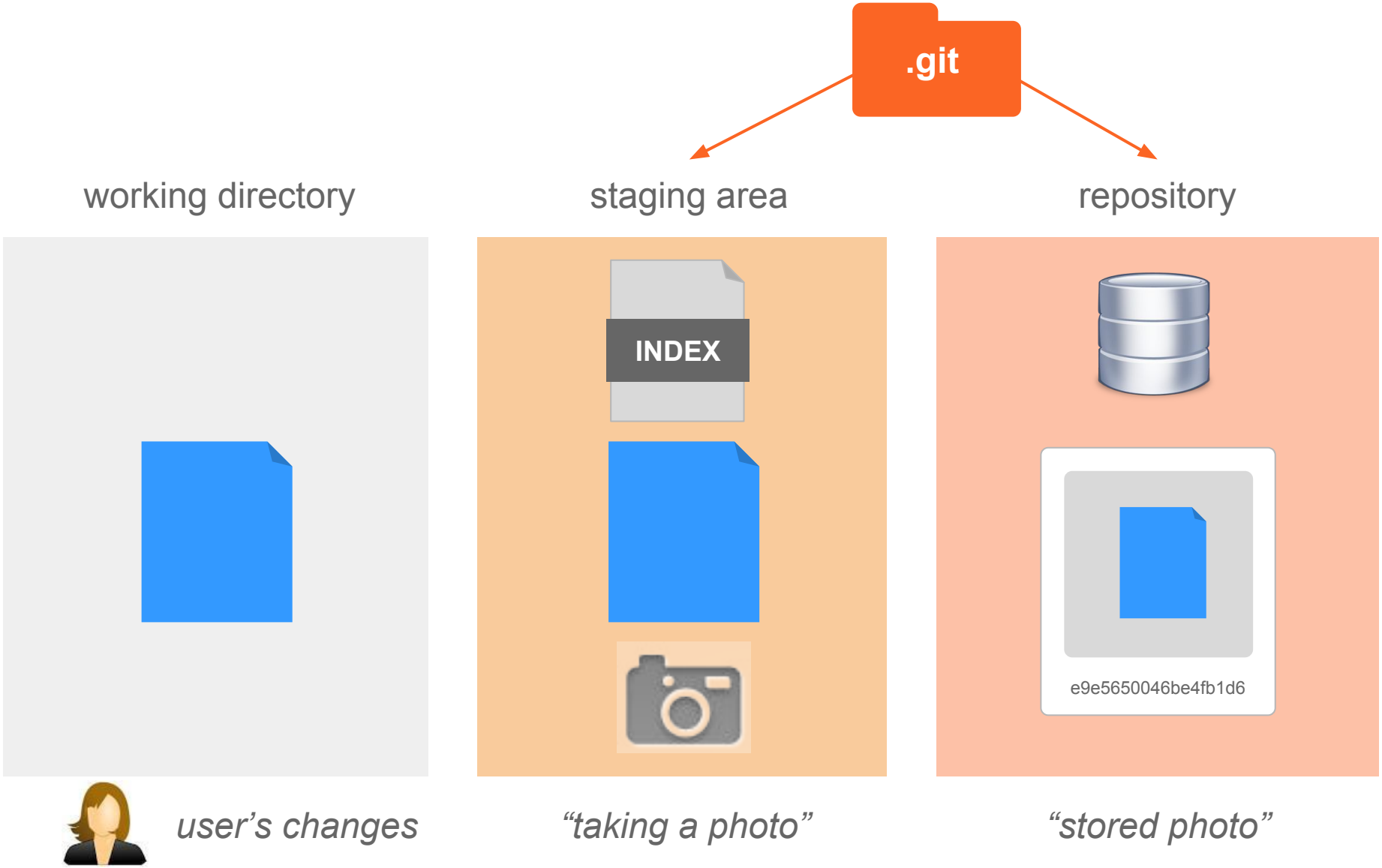
ID = hash commit

Determined by the SHA-1 algorithm https://en.wikipedia.org/wiki/SHA-1

How does Git  “take and store snapshots”?

Git keeps information about all commits in its database (inside the **.git** directory)

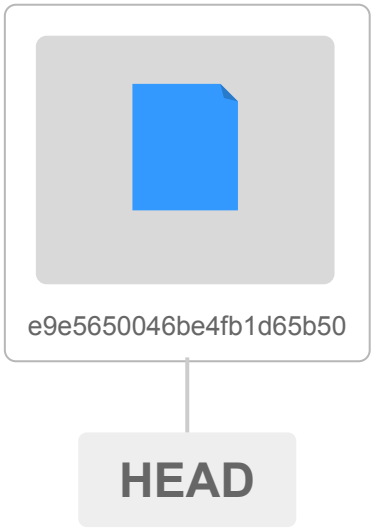




GIT USES A "THREE TREE" ARCHITECTURE

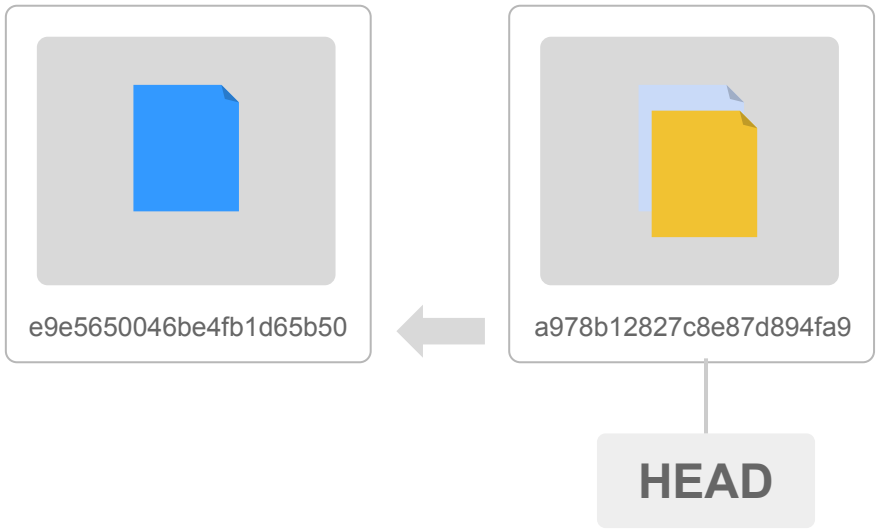
Basic Concept

HEAD



HEAD is a pointer

Typically, HEAD points to the last commit

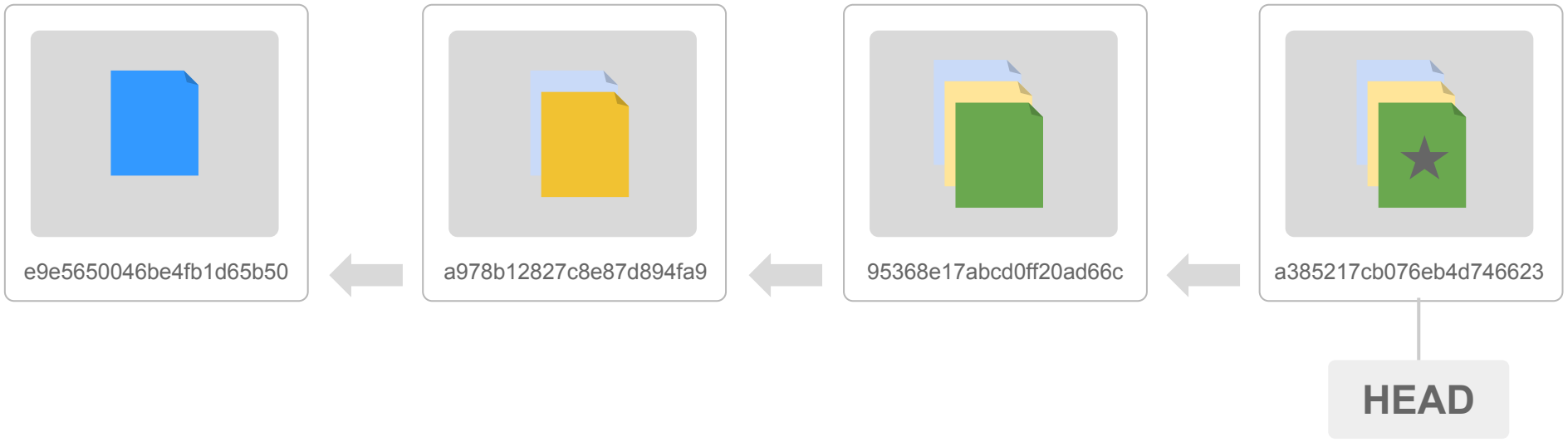


HEAD is a pointer

Typically, HEAD points to the last commit

HEAD is a pointer

Typically, HEAD points to the last commit



HEAD is a pointer

Typically, HEAD points to the last commit

Getting Started with Git

Installation & Configuration

Installation

Git is available for Mac, Windows, and Linux.

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

I'm assuming you already have Git installed in your computer.

Configuration

After installing Git, the next step involves the so-called **Git Configuration**.

Think of the configuration steps as "introducing yourself to Git".

The main command is `git config`

Global configuration

Tell Git who you are, e.g.:

```
git config --global user.name "Gaston Sanchez"

git config --global user.email "gasigiri@berkeley.edu"

git config --global color.ui "auto"
```

We recommend that you use your berkeley.edu email (which you should also use for your GitHub account)

Repository Initialization

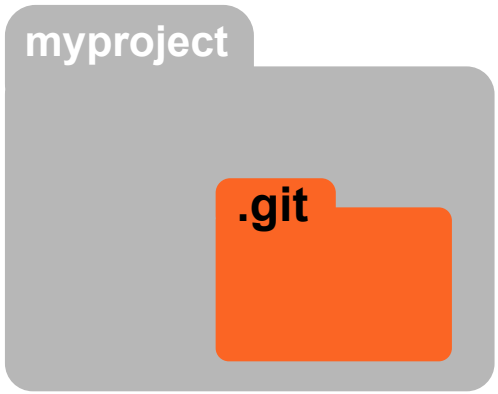
Git Initialization (of a repository)

Let's assume you are starting to work on a project (in its directory).

Open a terminal and `cd` to the project's directory.

Run `git init` to tell Git to start tracking changes.

Repository



Repository:
Database (hooked to a project) where the VCS stores all the versions and metadata of the project.

a project's repository is the `.git` directory

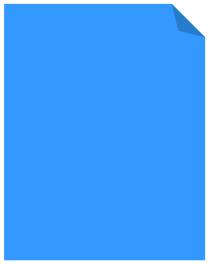
Initialization message

```
Initialized empty Git repository in
/Users/gaston/Documents/myproject/.git/
```

this is where Git will be storing
information about its tracking

Basic Workflow

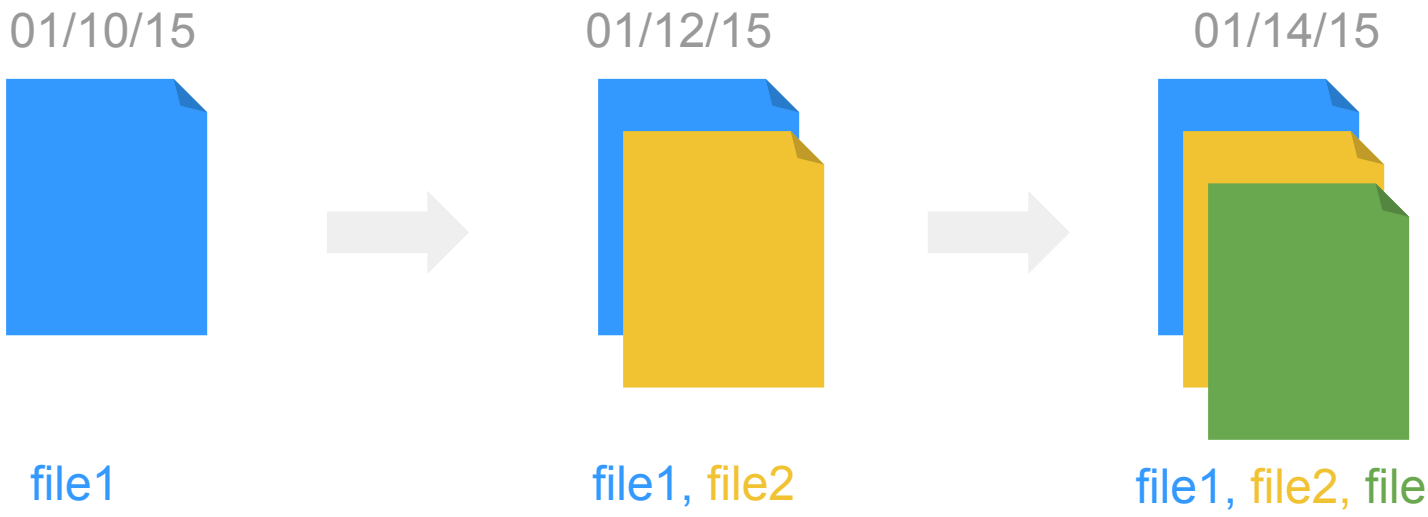
01/10/15



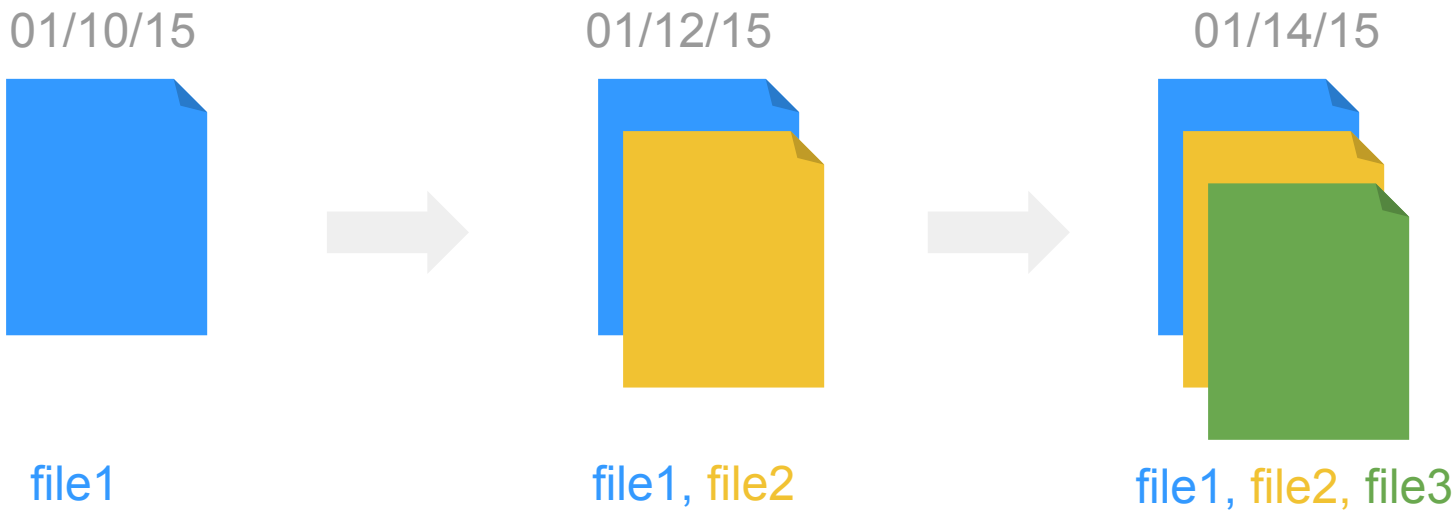
file1

```
git add file1
git status
git commit -m 'add file1'
```


```
git add file2
git status
git commit -m 'add file2'
```




```
git add file3
git status
git commit -m 'add file3'
```




```
git status
git log
```

Basic Workflow

Three basic commands

```
git add file1
```

```
git status
```

```
git commit -m "add file1"
```

We recommend that you use **git status** as often as possible

Commit Messages

Commit

- to give in trust or charge; consign.
- to entrust, especially for safekeeping; commend: *(to commit one's soul to God)*
- to do; perform; perpetrate *(to commit murder; to commit an error)*

DICTIONARY DEFINITIONS

Commit

- to give in trust or charge; consign.
- to entrust, especially for safekeeping; commend: *(to commit one's soul to God)*
- to do; perform; perpetrate *(to commit murder; ... error)*

VCS MEANING

- Snapshot of a complete project's state in a given point in time

Short Commit Messages

```
git commit -m 'first commit'
```

```
git commit -m 'add index file'
```

Long Commit Messages

```
git commit -m "something short

A longer description of what the
commit is all about (what it does)"
```

Suggestions

Bullet points are usually asterisks or hyphens

develop shorthand notation style:

- "`[py,R]`" (modifying python, R files)
- "`bugfix: …`" (all bug fixes commits)
- "`#35890`" (tracking numbers)

Writing Commit Messages

Bad example:

```
'fix typo'
```

Better:

```
'add missing > in index.html'
```

Writing Commit Messages

Bad:

```
'Updated cleaning data function'
```

Better:

```
'Remove missing values in cleaning
data function'
```

Writing Commit Messages

Bad:

```
'Changed correlation function. We
sould discuss this next meeting'
```

Better:

```
'Changed correlation function'
```