

Version control systems

Version control systems

Assume you have a project:

```
myproject.py
```

You would like to [try](#) a modification, but do not know if it will work.

```
mkdir versions/  
mkdir versions/v1/  
cp myproject.py versions/v1/
```

Result:

```
myproject.py  
versions/  
    v1/  
        myproject.py
```

You proceed to modify `myproject.py`:

```
myproject.py          <-- modified
versions/
  v1/
    myproject.py
```

If you like the modification and want to **commit** to it:

```
mkdir versions/v2/
cp myproject.py versions/v2/
```

Otherwise, you **revert** to the old version:

```
cp versions/v1/myproject.py myproject.py
```

If we **committed** to the modification:

```
myproject.py          <-- same as "versions/v2/myproject.py"
versions/
  v1/
    myproject.py
  v2/
    myproject.py
```

Use cases

- try things
- determine when a bug was introduced
- multiple people working on a project

Version control systems (VCS) / source code management (SCM)

- Revision control system (RCS), 1982
- Concurrent versions system (CVS), 1986
- Apache Subversion (“SVN”), 2000
- Mercurial (“Hg”), 2005
 - Used internally at Facebook/Meta
- Piper (not public)
 - Used for internal monorepo at Google
- Git, 2005
 - Spawned large hosting industry
(GitHub USD 7.5bn 2018, GitLab market cap around USD 8bn)
 - Used internally at Microsoft, Amazon

Git

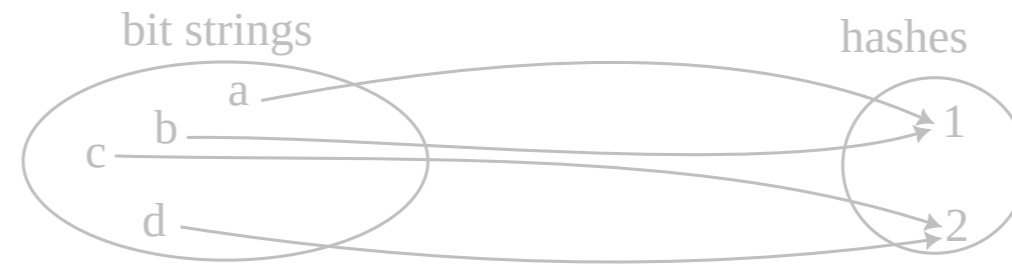
Git fundamentals

- a **repository** stores the complete history of a project
 - ~~versions/~~ → **.git/**
- a **commit** is a unit of change; it captures:
 - a snapshot of all the project files
 - an author, a date, ...
 - an indication of the “**parent commit**” (the one it is based on)
- **commits** are designated by ~~v1/~~ **a hash**

Hashes

- A **hash** maps any sequence of bits to a fixed-length bit string

- The map is **not injective**



- “SHA-1”: 160 bits / 20 bytes / 40 hex digits

Example: 1e6cac37c5c8c5ee99ec104954d09b07e96116ba

- Git assumes SHA-1 is **injective**

- ... and is currently migrating to SHA-256 (256 bits / 32 bytes / 64 hex digits)

Hashes in Git

- Git **commits** are designated by SHA-1 hashes

Example: 1e6cac37c5c8c5ee99ec104954d09b07e96116ba

- When referring to a **commit**, a hash prefix can be used if unambiguous

Example: 1e6cac

Git command-line interface

- General usage:
 - `git <command> [<arguments...>]`
 - Example: `git status`
- Getting help:
 - `git help <command>`
 - Example: `git help status`
 - `man git-<command>`
 - Example: `man git-status`

Configuration

- For anything too long for the CLI, `git` will make you edit a temporary file:

```
git config --global core.editor "code --wait"
```

- Commits capture the author's name and email address:

```
git config --global user.name "John Doe"  
git config --global user.email johndoe@example.com
```

Creating a repository

- Creating a new project:

```
mkdir my_new_project/  
cd my_new_project/  
git init
```

```
my_new_project/  
  .git/  
  ...
```

- “Cloning” an existing project:

```
git clone  
      https://github.com/ggerganov/llama.cpp.git  
cd llama.cpp/
```

```
llama.cpp/  
  .git/  
  ...  
  ggml-alloc.c  
  ggml-alloc.h  
  ...
```

Building a commit

Working tree, staging, commit

- We never access the content of `.git/` directly
- Instead, we modify files in the **working tree** (everything not in `.git/`)
 - We can ask `git` to “**check out**” any past commit into the working tree (i.e., make the working tree reflect that commit)
- In order to prepare a new commit, we “**stage**” the relevant modifications (i.e., we tell `git` which files we want part of the new commit)
- Once ready, we create the new commit, along with a commit message

Staging and committing example

- We create
 - `new_file_A.py`
 - `new_file_B.py`
 - `new_file_C.py`
- We **stage** `new_file_A.py` and `new_file_B.py`:

```
git add new_file_A.py new_file_B.py
```

- We **commit** them to the repository

```
git commit -m "My first commit."
```

Listing past commits

```
git log
```

```
commit 6ea8433cf989c7c8580194035c7871b7de3c7c08 (HEAD -> main)  
Author: Laurent Poirrier <poirrier@dev>  
Date:   Fri Sep 29 02:04:18 2023 +0200
```

```
    My first commit.
```

#

(all other directories)

working tree

(stored inside .git/ directory)

staging area

commits

```
# (creating new_file_A, new_file_B, new_file_C)
```

(all other directories)

working tree

```
new_file_A.py  
new_file_B.py  
new_file_C.py
```

(stored inside .git/ directory)

staging area

commits

```
git add new_file_A.py new_file_B.py
```

(all other directories)

working tree

```
new_file_A.py  
new_file_B.py  
new_file_C.py
```

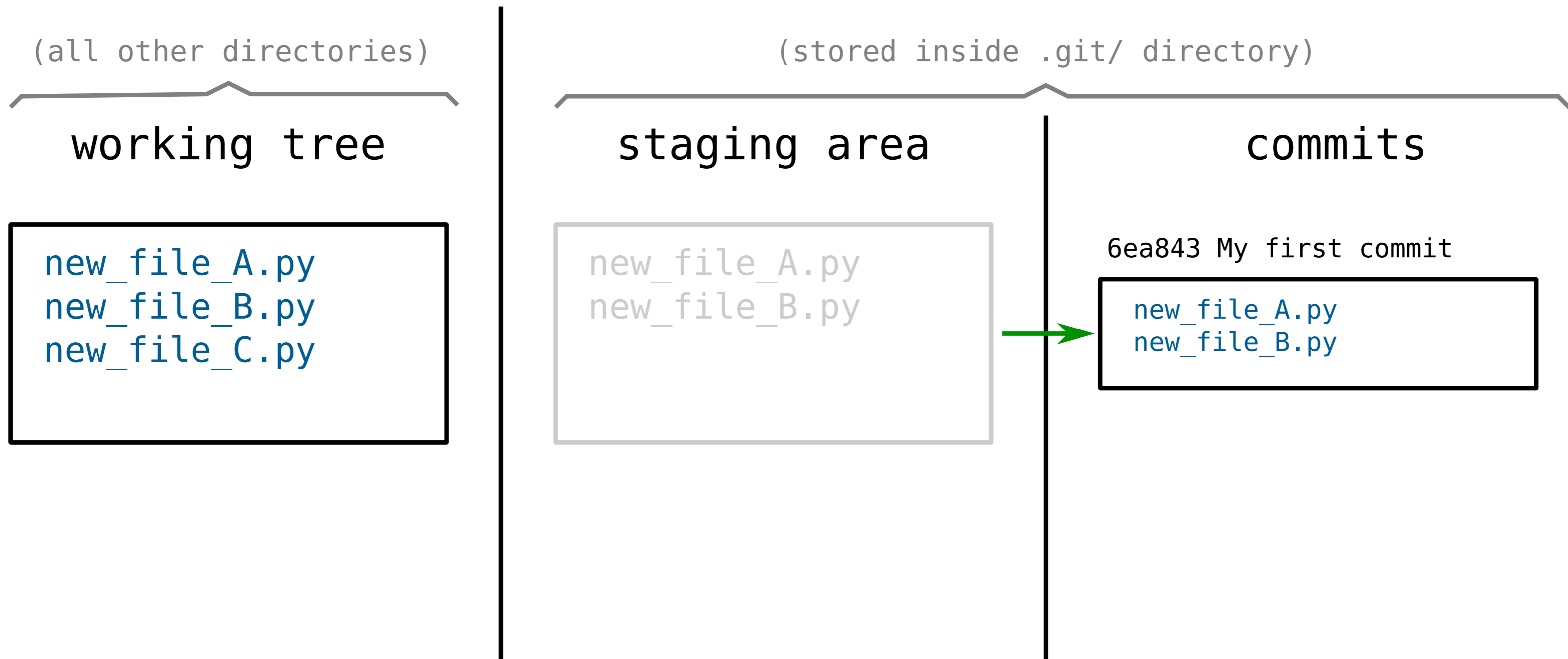
(stored inside .git/ directory)

staging area

```
new_file_A.py  
new_file_B.py
```

commits

```
git commit -m "My first commit."
```



```
#
```

(all other directories)

working tree

```
new_file_A.py  
new_file_B.py  
new_file_C.py
```

(stored inside .git/ directory)

staging area

```
new_file_A.py  
new_file_B.py
```

commits

6ea843 My first commit

```
new_file_A.py  
new_file_B.py
```

Automatic adding

- Add multiple files at once using a pattern (including in subdirectories):

```
git add '*.py'
```

- Add all the files in the working tree:

```
git add -A
```

- Exclude some files from “git add -A”:
Put corresponding patterns in “.gitignore”:

```
*.o  
/build/  
/my_executable
```

Observing the state of the working tree and staging area

```
git status
```

On branch main

Untracked files:

```
(use "git add <file>..." to include in what will be committed)
  new_file_C.py
```

nothing added to commit but untracked files present (use "git add" to track)

Let us modify `new_file_A.py`:

```
git status
```

On branch main

Changes not staged for commit:

```
(use "git add <file>..." to update what will be committed)
(use "git restore <file>..." to discard changes in working directory)
  modified:   new_file_A.py
```

Untracked files:

```
(use "git add <file>..." to include in what will be committed)
  new_file_C.py
```

no changes added to commit (use "git add" and/or "git commit -a")

Showing differences

```
git diff
```

```
diff --git a/new_file_A.py b/new_file_A.py
index e69de29..ec7780c 100644
--- a/new_file_A.py
+++ b/new_file_A.py
@@ -0,0 +1 @@
+print('Hello, world!')
```

Staging again

```
git add -A
```

```
git status
```

On branch main

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

modified: new_file_A.py

new file: new_file_C.py

```
# (previous state after first commit)
```

(all other directories)

working tree

```
new_file_A.py  
new_file_B.py  
new_file_C.py
```

(stored inside .git/ directory)

staging area

```
new_file_A.py  
new_file_B.py
```

commits

6ea843 My first commit

```
new_file_A.py  
new_file_B.py
```

```
# (modifying new_file_A)
```

(all other directories)

working tree

```
new_file_A.py  
new_file_B.py  
new_file_C.py
```

(stored inside .git/ directory)

staging area

```
new_file_A.py  
new_file_B.py
```

commits

6ea843 My first commit

```
new_file_A.py  
new_file_B.py
```

```
git add -A
```

(all other directories)

working tree

```
new_file_A.py  
new_file_B.py  
new_file_C.py
```

(stored inside .git/ directory)

staging area

```
new_file_A.py  
new_file_B.py  
new_file_C.py
```

commits

6ea843 My first commit

```
new_file_A.py  
new_file_B.py
```

Showing differences again

```
git diff
```

```
git diff --staged
```

```
diff --git a/new_file_A.py b/new_file_A.py
index e69de29..ec7780c 100644
--- a/new_file_A.py
+++ b/new_file_A.py
@@ -0,0 +1 @@
+print('Hello, world!')
diff --git a/new_file_C.py b/new_file_C.py
new file mode 100644
index 0000000..e69de29
```

Committing again

```
git commit -m "My second commit."
```

```
git log
```

```
commit 31a05126a56b8156de47ee53092b6996d75a0c8c (HEAD -> main)
Author: Laurent Poirrier <poirrier@dev>
Date:   Fri Sep 29 02:15:19 2023 +0200
```

My second commit.

```
commit 6ea8433cf989c7c8580194035c7871b7de3c7c08
Author: Laurent Poirrier <poirrier@dev>
Date:   Fri Sep 29 02:04:18 2023 +0200
```

My first commit.

```
# (previous state after "git add -A")
```

(all other directories)

working tree

```
new_file_A.py  
new_file_B.py  
new_file_C.py
```

(stored inside .git/ directory)

staging area

```
new_file_A.py  
new_file_B.py  
new_file_C.py
```

commits

6ea843 My first commit

```
new_file_A.py  
new_file_B.py
```

```
git commit -m "My second commit."
```

(all other directories)

working tree

```
new_file_A.py  
new_file_B.py  
new_file_C.py
```

(stored inside .git/ directory)

staging area

```
new_file_A.py  
new_file_B.py  
new_file_C.py
```

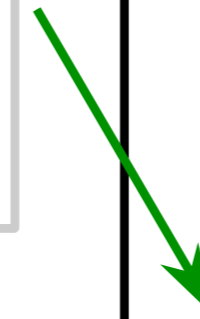
commits

6ea843 My first commit

```
new_file_A.py  
new_file_B.py
```

31a051 My second commit

```
new_file_A.py  
new_file_B.py  
new_file_C.py
```



```
#
```

(all other directories)

working tree

```
new_file_A.py  
new_file_B.py  
new_file_C.py
```

(stored inside .git/ directory)

staging area

```
new_file_A.py  
new_file_B.py  
new_file_C.py
```

commits

6ea843 My first commit

```
new_file_A.py  
new_file_B.py
```

31a051 My second commit

```
new_file_A.py  
new_file_B.py  
new_file_C.py
```

Checking out specific commits

```
git checkout 6ea843
```

```
git log
```

```
commit 6ea8433cf989c7c8580194035c7871b7de3c7c08 (HEAD)
Author: Laurent Poirrier <poirrier@dev>
Date:   Fri Sep 29 02:04:18 2023 +0200

    My first commit.
```

```
git log --all
```

```
commit 31a05126a56b8156de47ee53092b6996d75a0c8c (main)
Author: Laurent Poirrier <poirrier@dev>
Date:   Fri Sep 29 02:15:19 2023 +0200

    My second commit.

commit 6ea8433cf989c7c8580194035c7871b7de3c7c08 (HEAD)
Author: Laurent Poirrier <poirrier@dev>
Date:   Fri Sep 29 02:04:18 2023 +0200

    My first commit.
```

#

(all other directories)

working tree

```
new_file_A.py  
new_file_B.py  
new_file_C.py
```

(stored inside .git/ directory)

staging area

```
new_file_A.py  
new_file_B.py  
new_file_C.py
```

commits

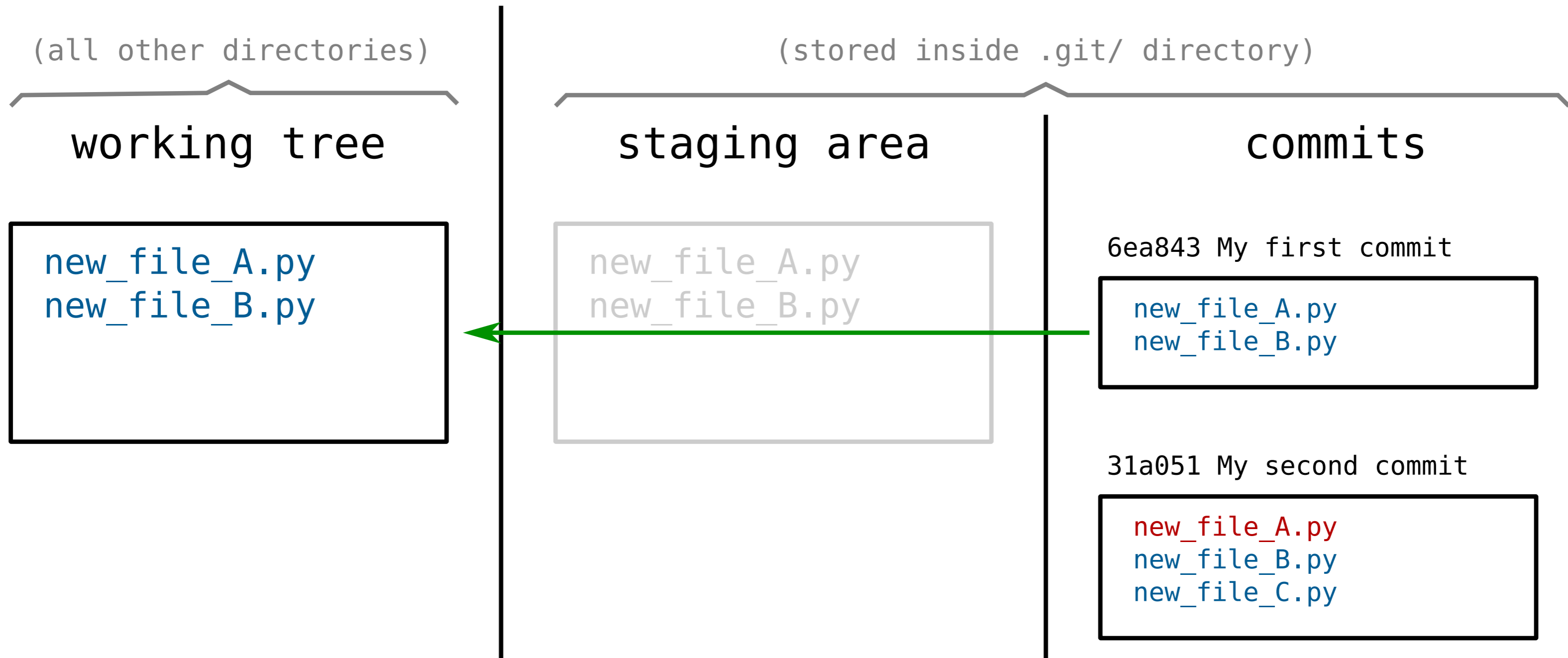
6ea843 My first commit

```
new_file_A.py  
new_file_B.py
```

31a051 My second commit

```
new_file_A.py  
new_file_B.py  
new_file_C.py
```

```
git checkout 6ea843
```



```
#
```

(all other directories)

working tree

```
new_file_A.py  
new_file_B.py
```

(stored inside .git/ directory)

staging area

```
new_file_A.py  
new_file_B.py
```

commits

6ea843 My first commit

```
new_file_A.py  
new_file_B.py
```

31a051 My second commit

```
new_file_A.py  
new_file_B.py  
new_file_C.py
```

Branches

Commit structure

```
31a051 ("My second commit.")  
  ^  
  |  
6ea843 ("My first commit.")
```

```
git checkout 31a051 # "My second commit"
```

```
31a051 ("My second commit.") <-- HEAD  
  ^  
  |  
6ea843 ("My first commit.")
```

```
git checkout 6ea843 # "My first commit"
```

```
31a051 ("My second commit.")  
  ^  
  |  
6ea843 ("My first commit.") <-- HEAD
```

```
# modify some files  
git add -A  
git commit -m "Another commit."
```

```
31a051 ("My second commit.")          07714c ("Another commit.")    <-- HEAD  
      ^                               ^  
      |                               |  
      6ea843 ("My first commit.")
```

```
git log --all --graph
```

```
* commit 07714cbadc8f13939039c07ac4b063d8b9b92506 (HEAD)
| Author: Laurent Poirrier <poirrier@dev>
| Date:   Fri Sep 29 03:06:02 2023 +0200
|
|     Another commit.
|
| * commit 31a05126a56b8156de47ee53092b6996d75a0c8c (main)
|/ Author: Laurent Poirrier <poirrier@dev>
|   Date:   Fri Sep 29 02:15:19 2023 +0200
|
|     My second commit.
|
| * commit 6ea8433cf989c7c8580194035c7871b7de3c7c08
|   Author: Laurent Poirrier <poirrier@dev>
|   Date:   Fri Sep 29 02:04:18 2023 +0200
|
|     My first commit.
```

Problem: if we “`git checkout`” back to the first or second commit, we lose “Another commit.”

Solution: [named branches](#)

Creating branches

```
git branch <branch-name>
```

(initial state after two commits)

```
HEAD, main --> 31a051 ("My second commit.")
                ^
                |
                6ea843 ("My first commit.")
```

```
git checkout 6ea843
```

```
main --> 31a051 ("My second commit.")
          ^
          |
HEAD --> 6ea843 ("My first commit.")
```

```
git branch my_branch
```

```
main --> 31a051 ("My second commit.")
          ^
          |
HEAD --> 6ea843 ("My first commit.") <-- my_branch
```

```
git checkout my_branch
```

```
main --> 31a051 ("My second commit.")
          ^
          |
          6ea843 ("My first commit.") <-- HEAD, my_branch
```

```
git add ...; git commit
```

```
main --> 31a051 ("My second commit.")          07714c ("Another commit.")    <-- HEAD, my_branch
          ^
          |
          6ea843 ("My first commit.")          ^
          |
```

Merging

```
git checkout main
```

```
HEAD, main --> 31a051 ("My second commit.")          07714c ("Another commit.")    <-- my_branch
                  ^                               ^
                  |                               |
                6ea843 ("My first commit.")
```

```
git merge my_branch
```

```
HEAD, main --> 81db75de ("Merge remote-tracking branch")
                 ^           ^
                 |           |
31a051 ("My second commit.") 07714c ("Another commit.") <-- my_branch
                 ^           ^
                 |           |
                 6ea843 ("My first commit.")
```

Merge conflicts

```
<<<<<< HEAD:new_file_A.py  
print('Hello, world!")  
=====  
print('Bye, world')  
>>>>>> my_branch:new_file_A.py
```

- resolve merge conflicts by editing files
- `git add ... ; git commit`

Rebase

```
(just committed to branch my_branch)
```

```
main --> 31a051 ("My second commit.")          07714c ("Another commit.")    <-- HEAD, my_branch
           ^
           |
           6ea843 ("My first commit.")          ^
           |
           6ea843 ("My first commit.")
```

```
git rebase main
```

```
          3c7c08 ("Another commit.")      <-- HEAD, my_branch
              ^
              |
main -->    31a051 ("My second commit.")
              ^
              |
          6ea843 ("My first commit.")
```

```
git checkout main
```

```
          3c7c08 ("Another commit.")      <-- my_branch
            ^
            |
HEAD, main --> 31a051 ("My second commit.")
            ^
            |
            6ea843 ("My first commit.")
```

```
git merge my_branch
```

```
HEAD, main --> 3c7c08 ("Another commit.") <-- my_branch
                 ^
                 |
                31a051 ("My second commit.")
                 ^
                 |
                6ea843 ("My first commit.")
```

Remotes

Sharing repositories

Git is distributed: there is no notion of a central server.

If the repository was created using

```
git clone <URL>
```

then

```
git fetch
```

checks for new commits from the same <URL> [origin](#)

To download commits from a remote repository:

```
git fetch <URL>
```

Note: <URL> must be public, or we must have appropriate credentials

- as an alternative,

```
git format-patch
```

saves commits in files that can be sent by email.

Fetch and pull, remotes

- Just fetch repository data, do not affect working tree:

```
git fetch [<repository>] [remote_branch:local_branch]
```

- Fetch data and attempt merge (or rebase):

```
git pull [<repository>]
```

- Setup a remote:

```
git remote add [options] <name> <URL>
```

Configuration and more

Configuration

- For anything too long for the CLI, `git` will make you edit a temporary file:

```
git config --global core.editor "code --wait"
```

- Commits capture the author's name and email address:

```
git config --global user.name "John Doe"  
git config --global user.email johndoe@example.com
```

Q: How many git subcommands are there?

```
man git | grep -E '^ *git-.*\ (1\)$'
```

A: 147

⇒ Use `git help / man git`

