Let's use version control (git)!

Daniel Heimgartner

June 12, 2025

Highly recommended: Read the first three chapters of https://git-scm.com/book/en/v2

Code to be executed in a terminal running bash looks like this

```
git init .  # what's the connection between git and bash? dot?
```

1 WHY?

What is git

- git is a (there are many!) *version control system* that helps you track and manage changes to source code (files) over time.
- Think of a (time-ordered) linked chain of cabinets where you store your current state of your work. And in each cabinet you also put a list, describing the work you did since the previous archiving in the previous cabinet. Thinking about this metaphor, what benefits could you think of?

What are the benefits?

- Don't be afraid to experiment/mess with a document (better than `Ctrl+Z`)
- Keep track of how a document changed (e.g., submitted vs. re-submitted paper/script)
- If your in the habit of `foo_v01.R`, `foo_v02.R`, ..., `foo_v99.R`, git is for you!
- Own mental freedom. But also key benefits when collaborating (next week).

What will I learn?

- Today we will focus on how *you* for yourself can use git. Collaboration (i.e., when multiple people work on the same "git repository") is the topic of the next meeting.
- The goal for today: 1. Understand the git cycle, 2. Start tracking a project with git, 3. Inspect changes you've made, 4. Accept or restore changes, 5. Commit changes, 6. Inspect (checkout) a previous version, 7. Experiment on a new branch, 8. Accept (merge) the changes of the new branch.
- Learn where to learn more!

2 SETTLING A POTENTIAL CONFUSION

git is an executable program (not constrained to bash). It's a command-line-interface (CLI) tool – an API following the `command -options args` syntax (and not a programming language or something like that). I.e., we have to learn (!) the API/commands...

You could clone (next session) the source `https://github.com/git/git` (adjust it) and build the executable program yourself!

```
which git
/usr/bin/git
```

3 GETTING HELP

Do you remember from last week? Where can you get help?

```
git help
git help <verb>    # git help init
git <verb> --help  # git init --help
# this is what we've discussed, using man...
man git-<verb>     # man git-init
```

4 LET'S GET STARTED

Let's imagine we have a project currently consisting of one file only (so unfortunately, it's not an R-package...).

```
cd foo
tree .
.
└── script.R

0 directories, 1 file

cat script.R
#= script.R
f <- function() {
  NULL
}
```

We use version control for a project/directory (and then add or exclude individual files)...

The git cycle

And we want to start tracking changes in that project, i.e., we want to use git version control in that repository!

The general workflow (for today) looks something like this

"blob" is not really a meaningful commit message. Try to be precise about what you did. Try to make meaningful chunks you commit (i.e., `git add` files and changes that belong together)

```
  +-------------------------+
  |    git init .           |
  |    git add .            |
  |    git commit -m "init" |
  +-------------------------+
              |
```

```
             v
   +------------------+
   |  Work on foo.R   | <----+
   +------------------+      |
             |               |
             v               |
     +---------------+       |
     | git add foo.R |       |
     +---------------+       |
             |               |
             v               |
+----------------------+     |
| git commit -m "blob" |-----+
+----------------------+
```

git init Let's go through this step by step

"Create an empty git repository or reinitialize an existing one"

```
# don't worry about this...
mkdir foo
cd foo
printf "#= script.R\nf <- function() {\n  NULL\n}\n" > script.R

# this is the important line
git init .
```

git status

"Show the working tree status"

```
git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        script.R

nothing added to commit but untracked files present (use "git ad l" to track)
```

git add

"Add file contents to the index"

```
git add .      # add all changes
git add foo.R  # add changes in foo.R
```

Again, insightful to check the status

```
git status
On branch master

No commits yet
```

```
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   script.R
```

git commit

As always, it's a good idea to read the output of a command – it is there to help you!

"Record changes to the repository" `git add` is like adding a file to the file cabinet, but the door has not yet been closed (you can still replace it) and the list recording the changes is not yet written...

```
git commit -m "init"  # the usual first commit message
```

Now, it's officially a git repository – yay!!

What makes a repository a git repository? Hint: `ls -a .`

git diff

...Now, you work on script.R...

"Show changes between commits, commit and working tree, etc."

```
git diff .  # or script.R
diff --git a/script.R b/script.R
index 1165da1..a008e94 100644
--- a/script.R
+++ b/script.R
 #= script.R
 f <- function() {
-  NULL
+  "foo"
 }
```

git log

...Add and commit again...

"Show commit logs"

```
git log
commit e62788c24e2d9fdc5b79b111256935ad8a87d97c (HEAD -> master)
Author: daniehei <daniel.heimgartner@ivt.baug.ethz.ch>
Date:   Tue May 20 15:36:39 2025 +0200

    foo

commit 60a73d8900f91290397f595afec5b7e88cf8d18a
Author: daniehei <daniel.heimgartner@ivt.baug.ethz.ch>
Date:   Tue May 20 15:34:18 2025 +0200

    init
```

git checkout

"Switch branches or restore working tree files"

```
git checkout HEAD~1  # go one commit back
cat script.R
#= script.R
f <- function() {
  NULL
}
```

git branch

Where has our valuable work gone???

"List, create, or delete branches" When you really want to mess with your work, I would recommend to generate a new branch first (as it is easier to checkout, restore branches instead of going back in the commit history)...

As we will learn next session, GitHub allows you essentially to host (remote) branches, such that multiple participants can inspect and share work. And then they wrap of course other useful functionality around it...

```
git branch dev    # you can call it whatever you like
git checkout dev  # don't forget to checkout!
```

Now, we are on a new branch – completely leaving the `master` branch unaffected!

Exercise: Try to rename the branch. Hint: `man git-branch` and then type `/rename` and hit `Enter`.

There are two possibilities going forward: 1. We like the changes we made on our `dev` branch, or, 2. We don't like them at all, we hate them, and want get rid of them! Let's start with 2. to confuse you!

```
git checkout master # switch to main again
git branch -d dev   # and delete the branch
```

git merge

"Join two or more development histories together" Let's say we like the changes made on `dev` branch... Oups, sorry, you just deleted the branch – the changes are actually gone forever! Let's say we did not execute the code above.

```
git checkout master
git diff main dev  # diff works also with branches
git merge dev      # this integrates/merges the changes
```

5 SPECIAL FILES AND FOLDERS

`.git`
`.gitignore`

This is where the magic happens!

Put all files or folders there which should not be tracked by git (e.g., passwords)...

Careful: Once you've committed a file containing sensitive information, this information is part of the history and can be recovered, even if you remove the information from the file again. There are ways to remove sensitive information completely from the history, but it is still a good to be sensitized (?)

6 HELPFUL TOOLS

gitk

gitk is a graphical repository browser. It can be thought of as a GUI wrapper for `git diff` or `git log` and is useful for exploring and visualizing the history of a repository. Try it! ...Make changes to script.R...

```
gitk
```

`usethis::use_git()`

Almost every IDE (e.g., RStudio or VScode have built-in support to visualize such changes. Of course, use them if you prefer!)

If you want to start using git in an R-project, you can call `usethis::use_git()` in an R-console (with the project root as the working directory). This is more or less equivalent to

```
git init .
git add .
git commit -m "Initial commit"
```

Resources

- Foo