^ (../)

Introduction to Git and GitHub (../)

> (../02-history/)

Tracking changes with a local repository

Overview

Teaching: 35 min
Exercises: 0 min
Questions

- How do I get started with Git?

Objectives

- Know how to set up a new Git repository.
- Understand how to start tracking files.
- Be able to commit changes to your repository.

Version control is centered round the notion of a *repository* which holds your directories and files. We'll start by looking at a local repository. The local repository is set up in a directory in your local file system (local machine). For this we will use the command line interface.

Why use the command line?

There are lots of graphical user interfaces (GUIs) for using Git: both stand-alone and integrated into IDEs (e.g. MATLAB, Rstudio). We are deliberately not using a GUI for this course because:

- you will have a better understanding of how the git commands work
- you will be able to use Git on any computer (e.g. remotely accessing HPC systems, which generally only have Linux command line access)
- you will be able to use any GUI, rather than just the one you have learned

Setting up Git

Git should already be installed on your machine. If you still need to install git, instructions are under setup (../setup).

Tell Git who we are

As part of the information about changes made to files Git records who made those changes. In teamwork this information is often crucial (do you want to know who rewrote your 'Conclusions' section?). So, we need to tell Git about who we are (note that you need to enclose your name in quote marks):

```
$ git config --global user.name "Your Name"                    # Put your quote marks around yo
ur name
$ git config --global user.email yourname@yourplace.org
```

Set a default editor

When working with Git we will often need to provide some short but useful information. In order to enter this information we need an editor. We'll now tell Git which editor we want to be the default one (i.e. Git will always bring it up whenever it wants us to provide some information).

You can choose any editor available on your system. For the purpose of this session we'll use *nano*:

```
$ git config --global core.editor nano                         # Linux users only.
                                                               # Windows users should use notep
ad: see below.
                                                               # Mac users should use TextEdit:
see below.
```

To set up alternative editors, follow the same notation e.g. `git config --global core.editor notepad` , `git config --global core.editor vi` , `git config --global core.editor xemacs` .

Mac users can use *TextEdit*: `git config --global core.editor 'open -W -n'` .

Git's global configuration

We can now preview (and edit, if necessary) Git's global configuration (such as our name and the default editor which we just set up). If we look in our home directory, we'll see a `.gitconfig` file,

```
$ cat ~/.gitconfig
    [user] name = Your Name email = yourname@yourplace.org
    [core] editor = nano
```

These global configuration settings will apply to any new Git repository you create on your computer. i.e. the `--global` commands above are only required once per computer.

Create a new repository with Git

We will be working with a simple example in this tutorial. It will be a paper that we will first start writing as a single author and then work on it further with one of our colleagues.

First, let's create a directory within your home directory:

```
$ cd                                                           # Switch to your home directory.
$ pwd                                                          # Print working directory (outpu
t should be /home/<username>)
$ mkdir papers
$ cd papers
```

Now, we need to set up this directory up to be a Git repository (or "initiate the repository"):

```
$ git init
```

```
Initialized empty Git repository in /home/user/papers/.git/
```

The directory “papers” is now our working directory.

If we look in this directory, we’ll find a `.git` directory:

```
$ ls .git
```

```
branches  config  description  HEAD  hooks  info  objects refs
```

The `.git` directory contains Git’s configuration files. Be careful not to accidentally delete this directory!

Tracking files with a git repository

Now, we’ll create a file. Let’s say we’re going to write a journal paper, so we will start by adding the author names and a title, then save the file.

```
$ nano journal.md                                          # Windows users: use notepad ins
tead of nano (throughout this course)
# Add author names and paper title
```

📌 Accessing files from the command line

In this lesson we create and modify text files using a command line interface (e.g. terminal, Git Bash etc), mainly for convenience. These are normal files which are also accessible from the file browser (e.g. Windows explorer), and by other programs.

`git status` allows us to find out about the current status of files in the repository. So we can run,

```
$ git status
```

```
On branch master

Initial commit

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

journal.md

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

Information about what Git knows about the directory is displayed. We are on the `master` branch, which is the default branch in a Git repository (one way to think of branches is like parallel versions of the project - more on branches later).

For now, the important bit of information is that our file is listed as **Untracked** which means it is in our working directory but Git is not tracking it - that is, any changes made to this file will not be recorded by Git.

Add files to a Git repository

To tell Git about the file, we will use the `git add` command:

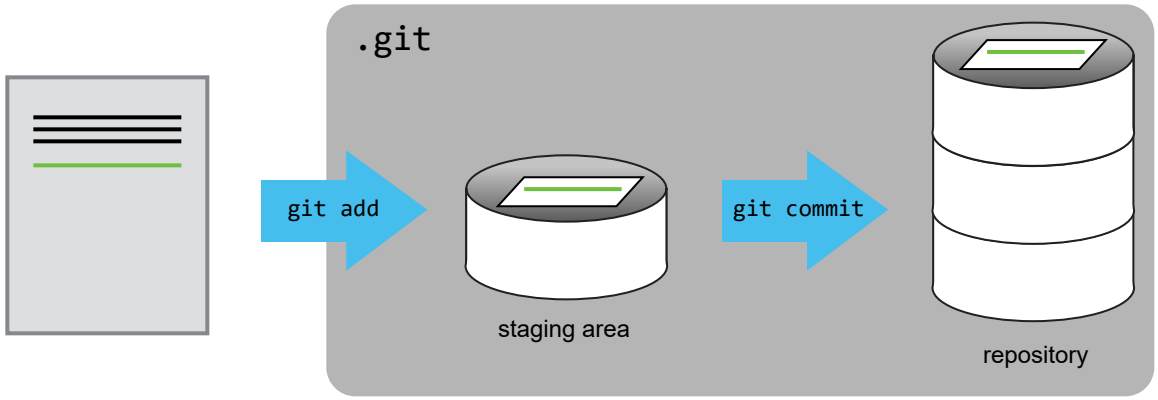
```
$ git add journal.md
$ git status
```

```
On branch master

Initial commit

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

        new file:   journal.md
```

Now our file is listed underneath where it says **Changes to be committed**.

`git add` is used for two purposes. Firstly, to tell Git that a given file should be tracked. Secondly, to put the file into the Git **staging area** which is also known as the *index* or the *cache*.

The staging area can be viewed as a “loading dock”, a place to hold files we have added, or changed, until we are ready to tell Git to record those changes in the repository.



Commit changes

In order to tell Git to record our change, our new file, into the repository, we need to **commit** it:

```
$ git commit
# Type a commit message: "Add title and authors"
# Save the commit message and close your text editor (nano, notepad etc.)
```

Our default editor will now pop up. Why? Well, Git can automatically figure out that directories and files are committed, and by whom (thanks to the information we provided before) and even, what changes were made, but it cannot figure out why. So we need to provide this in a commit message.

If we save our commit message **and exit the editor**, Git will now commit our file.

```
[master (root-commit) 21cfbde]
1 file changed, 2 insertions(+) Add title and authors
create mode 100644 journal.md
```

This output shows the number of files changed and the number of lines inserted or deleted across all those files. Here, we have changed (by adding) 1 file and inserted 2 lines.

Now, if we look at its status,

```
$ git status
```

```
On branch master
nothing to commit, working directory clean
```

our file is now in the repository. The output from the `git status` command means that we have a clean directory i.e. no tracked but modified files.

Now we will work a bit further on our *journal.md* file by writing the introduction section.

```
$ nano journal.md
# Write introduction section
```

If we now run,

```
$ git status
```

we see changes not staged for commit section and our file is marked as modified:

```
On branch master
Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git checkout -- <file>..." to discard changes in working directory)

     modified:   journal.md

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

This means that a file Git knows about has been modified by us but has not yet been committed. So we can add it to the staging area and then commit the changes:

```
$ git add journal.md
$ git commit                                                  # "Write introduction"
```

Note that in this case we used `git add` to put journal.md to the staging area. Git already knows this file should be tracked but doesn't know if we want to commit the changes we made to the file in the repository and hence we have to add the file to the staging area.

It can sometimes be quicker to provide our commit messages at the command-line by doing `git commit -m "Write introduction section"` .

Let's add a directory *common* and a file *references.txt* for references we may want to reuse:

```
$ mkdir common
$ nano common/references.txt                                    # Add a reference
```

We will also add a citation in our introduction section (in journal.md).

```
$ nano journal.md                                               # Use reference in introduction
```

Now we need to record our work in the repository so we need to make a commit. First we tell Git to track the references. We can actually tell Git to track everything in the given sub-directory:

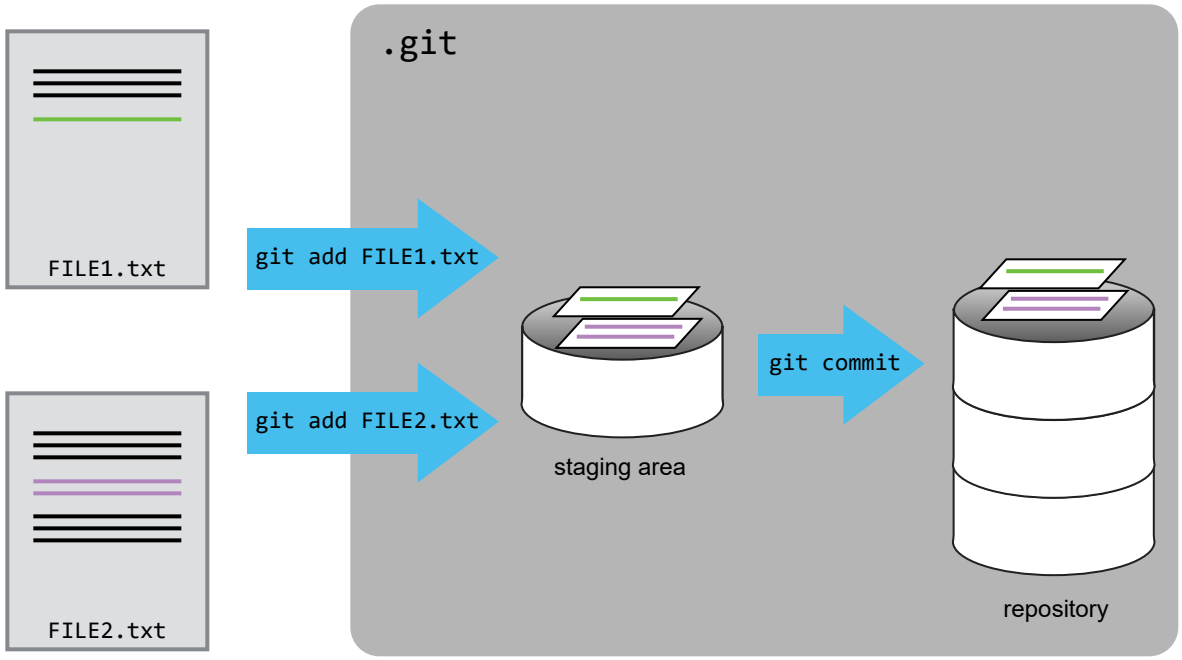
```
$ git add common                                                # Track everything currently in
 the 'common' directory
$ git status                                                    # Verify that common/references.
txt is now tracked
```

All files that are in *common* are now tracked. We would also have to add journal.md in the staging area. But there is a shortcut. We can use `commit -a` . This option means "commit all files that are tracked and that have been modified".

```
$ git commit -am "Reference J Bloggs and add references file"   # Add and commit all tracked fil
es
```

and Git will add, then commit, both the directory and the file.

In order to add all tracked files to the staging area, use `git commit -a` (which may be very useful if you edit e.g. 10 files and now you want to commit all of them).



Key Points

- `git init` initializes a new repository
- `git status` shows the status of a repository
- Files can be stored in a project's `working directory` (which users see), the `staging area` (where the next commit is being built up) and the `local repository` (where commits are permanently recorded)
- `git add` puts files in the staging area
- `git commit` saves the staged content as a new commit in the local repository
- Always write a log message when committing changes

^ (../)

> (../02-history/)

Edit on GitHub (https://github.com/emdupre/git-course/edit/gh-pages/_episodes/01-local.md) / Contributing (https://github.com/emdupre/git-course/blob/gh-pages/CONTRIBUTING.md) / Source (https://github.com/emdupre/git-course/) / Cite (https://github.com/emdupre/git-course/blob/gh-pages/CITATION) / Contact (mailto:elizabeth.dupre@mail.mcgill.ca?subject=Git course)