Version Control System - Git

Quantitative Applications for Data Analysis

Alexey Fedoseev

February 1, 2024

Last version

You were asked to share the source code of your article with your colleague. You spent several weeks polishing your article and happy to share your work, although it was almost a year ago. You go into the directory with your article...

```
user@scinet article $ ls
article/            article_final2/     article_last/
article_20220128/   article_final_last/ article_rev1/
article_final/      article_final_rev2/ article_rev2/
```

Well, it is not that very obvious which version actually has the latest changes...

Very often you want to save your ideas, approaches even if they were wrong, simply because you came up with a smart solution, but discovered a better or alternative approach.

You end up having a lot of different versions of the same file.

A simplest solution is to always use dates and a small comment in you your directory and/or file names. However you end up having a lot of files and it becomes obscure where is what.

Version Control System

A better way to keep track of the changes in the files is to use the Version Control System (VCS). There are many Version Control System available, however currently Git is considered the standard in the software development industry.

Git is an open-source software that supports many platforms including Windows, OSX and Linux. It allows you to save the history of the changes in your files, revert the files back to their previous state and compare the changes over time.

Starting git

To check whether you have `git` installed, open the terminal and type `git`.

```
user@scinet ~ $ git
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
...
```

If you see an error "`command not found`" ask for help and we will assist you to install it.

Currently the latest stable release version is `2.39.1`.

Configuring git

The first thing you need to do is to introduce yourself to Git since it is heavily used as a collaborative tool between many people and it is important that Git keeps the name and the email address of the person who made the change, so it possible to connect with this person if you have any questions.

To set up your credentials specify your first, last name and your email using the following commands

```
user@scinet ~ $ git config --global user.name "Firstname Lastname"
user@scinet ~ $ git config --global user.email "username@utoronto.ca"
```

Run the following commands to verify that Git knows who you are

```
user@scinet ~ $ git config user.name
Alexey Fedoseev
user@scinet ~ $ git config user.email
alexey.fedoseev@scinet.utoronto.ca
```

Creating a repository

We are going to develop a simple project and will track the changes in it. Let us create an empty directory `cars`.

```
user@scinet ~ $ mkdir cars
user@scinet ~ $ cd cars
```

Now that we are in the empty directory `cars` let us initialize a new git repository using the command `git init`.

```
user@scinet cars $ git init
Initialized empty Git repository in /Users/alexey/cars/.git/
```

A Git repository is simply a place where the history of your work is stored.

Git repository

If you type `ls` it seems like nothing happened.

```
user@scinet cars $ ls
user@scinet cars $
```

This is where plentiful additional flags of `ls` come in handy.

```
user@scinet cars $ ls -a
./    ../   .git/
```

In Unix-like operating systems, any file or folder that starts with a dot character is to be treated as hidden – that is, the `ls` command does not display them unless the `-a` flag (`ls -a`) is used.

As we see, Git initialized the repository in a hidden directory named `.git`. Remember that if you remove this directory `.git`, your history of changes will be lost.

Adding files

In our project we would rely on the small `cars` dataset. It contains specifications like fuel consumption, engine and transmission type, etc., collected for a variety of cars.

You can download and explore the dataset using the following link:

https://pages.scinet.utoronto.ca/~afedosee/cars.csv

Now let us add the `cars` dataset to our project.

```
user@scinet cars $ curl -O https://pages.scinet.utoronto.ca/~afedosee/cars.csv
user@scinet cars $ ls
cars.csv
```

Checking the status

After we added the `cars` dataset to our project it is important to save this change.

Git already knows about our modification and you can use `git status` to verify which files were modified.

```
user@scinet cars $ git status
On branch master

No commits yet

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

    cars.csv

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

Checking the status and staging the files

```
user@scinet cars $ git status
...
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    cars.csv
```

Git noticed our new file however, we have a choice whether we want to track the changes in it or not.

In order to start tracking the file we use the command `git add <file>`.

Using Git terminology, it is called “staging”.

```
user@scinet cars $ git add cars.csv
```

Staging files

Saving the changes in files in Git is called "committing" the changes.

After we "staged" our new file `cars.csv` we can use `git status` to verify that the file `cars.csv` is going to be saved or "committed" to our Git repository.

```
user@scinet cars $ git status
On branch master

No commits yet

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

    new file:   cars.csv
```

Committing the changes

In order to commit our changes we must specify a brief description of the modifications we made. It should be a simple note for yourself and others as to what has been done, it will help you to distinguish the file versions.

To commit our changes to the repository together with the note about modifications use the following command

```
user@scinet cars $ git commit -m "Added cars.csv"
[master (root-commit) 53c7ae6] Added cars.csv
 1 file changed, 5077 insertions(+)
 create mode 100644 cars.csv
```

Checking the history

To view the history of changes use the command `git log`.

```
user@scinet cars $ git log
commit 53c7ae61871b7968baae115def85bf4261c3ec5b (HEAD -> master)
Author: Alexey Fedoseev <alexey.fedoseev@scinet.utoronto.ca>
Date:   Mon Jan 28 10:19:51 2023 -0500

    Added cars.csv
```

So far we have only one commit. Let us add one more.

More changes

I have decided to move my dataset file into the directory `data`. First create the directory and then use `mv` command to move the file `cars.csv` into it.

```
user@scinet cars $ mkdir data
user@scinet cars $ mv cars.csv data/
user@scinet cars $ ls
data/
user@scinet cars $ ls data/
cars.csv
```

Checking status

You can check `git status` to verify that Git noticed the change. However, the directory `data` is not tracked by Git yet.

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

    deleted:    cars.csv

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

    data/

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

Staging multiple changes

Two things happened when we moved the file `cars.csv`: it was copied into the `data` directory and the original file `cars.csv` was deleted.

Therefore we need to tell Git that the file `cars.csv` was removed and start tracking the changes in the `data` directory.

```
user@scinet cars $ git add cars.csv data
user@scinet cars $ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    renamed:    cars.csv -> data/cars.csv
```

You can see that Git figured out that the file was moved (`cars.csv -> data/cars.csv`)

Commiting the change

```
user@scinet cars $ git commit -m "Moved cars.csv to data directory"
[master 3f81319] Moved cars.csv to data directory
 1 file changed, 0 insertions(+), 0 deletions(-)
 rename cars.csv => data/cars.csv (100%)

user@scinet cars $ git log
commit 3f813198daad3337227ff7327bd6b617def92674 (HEAD -> master)
Author: Alexey Fedoseev <alexey.fedoseev@scinet.utoronto.ca>
Date:   Mon Jan 28 10:24:07 2023 -0500

    Moved cars.csv to data directory

commit 53c7ae61871b7968baae115def85bf4261c3ec5b
Author: Alexey Fedoseev <alexey.fedoseev@scinet.utoronto.ca>
Date:   Mon Jan 28 10:19:51 2023 -0500

    Added cars.csv
```

One line history

Notice how much information `git log` gives on every commit. If you have many commits, `git log --oneline` is a minimalistic alternative that will show you only the titles and first 7 symbols of the commit id.

```
user@scinet cars $ git log --oneline
3f81319 (HEAD -> master) Moved cars.csv to data directory
53c7ae6 Added cars.csv
```

Multiple changes

Create a file `car_search.R` in the directory `cars` that takes command line arguments and displays them back to the terminal.

```
user@scinet cars $ cat car_search.R
args <- commandArgs(trailingOnly = TRUE)
cat("The command line arguments are:", args, "\n")

user@scinet cars $ Rscript car_search.R Audi Porsche
The command line arguments are: Audi Porsche
```

Additionally, add the `README.md` file with the short description of our project.

```
user@scinet cars $ cat README.md
Car search project
```

Checking status

The command `git status` shows us the files that were modified.

```
user@scinet cars $ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    README.md
    car_search.R

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

Staging multiple files

In order to stage multiple changes we can `git add` the top-level directory of our project which includes all of our changes.

```
user@scinet cars $ git add .
user@scinet cars $ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   README.md
    new file:   car_search.R
```

Committing the changes

Finally we can commit the changes with a descriptive comment.

```
user@scinet cars $ git commit -m "Added car_search.R and README.md"
[master 9a03a90] Added car_search.R and README.md
 2 files changed, 3 insertions(+)
 create mode 100644 README.md
 create mode 100644 car_search.R
```

The `git log` command immediately reflects our commit.

```
user@scinet cars $ git log --oneline
9a03a90 (HEAD -> master) Added car_search.R and README.md
3f81319 Moved cars.csv to data directory
53c7ae6 Added cars.csv
```

Unstaging files

Sometimes you may accidentally remove the files.

```
user@scinet cars $ rm car_search.R data/cars.csv
user@scinet cars $ git add .
```

```
user@scinet cars $ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    deleted:    car_search.R
    deleted:    data/cars.csv
```

Unstaging files

After you've realized that you do not want to commit the removal of files, you can unstage the change using the command `git reset`. This command will unstage all staged files and preserve the changes.

```
user@scinet cars $ git reset
Unstaged changes after reset:
D   car_search.R
D   data/cars.csv
```

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

    deleted:    car_search.R
    deleted:    data/cars.csv
```

Discarding changes

You can completely discard all changes since the last commit. For example, the command `git checkout -- car_search.R` completely restores the file `car_search.R`

```
user@scinet cars $ ls
README.md  data/
user@scinet cars $ git checkout -- car_search.R
user@scinet cars $ ls
README.md     car_search.R  data/
user@scinet cars $ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    deleted:    data/cars.csv

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

Discarding changes

You can use `git checkout -- .` to discard all changes in the current directory since the last commit

```
user@scinet cars $ git checkout -- .
user@scinet cars $ git status
On branch master
nothing to commit, working tree clean
```

Workflow

Your typical workflow in Git should consist of staging the changes and committing them to the repository.

```
user@scinet cars $ git add .
user@scinet cars $ git status
user@scinet cars $ git commit -m "Added vectorToString() to car_search.R"
[master 6d2d144] Added vectorToString() to car_search.R
 1 file changed, 5 insertions(+)
```

Viewing the commit

To view the commit use `git show` with the commit id (you can use first 7 symbols if they are unique).

```
user@scinet cars $ git log --oneline -n 2
6d2d144 (HEAD -> master) Added vectorToString() to car_search.R
9a03a90 Added car_search.R and README.md

user@scinet cars $ git show 6d2d144
commit 6d2d1446a40754bdb93ce68b725687f2409342f8 (HEAD -> master)
...
 cat("The command line arguments are:", args, "\n")
+
+# Concatenates the vector using commas between the elements
+vectorToString <- function(vector) {
+  return(paste(vector, collapse = ", "))
+}
```

References

This is a very brief introduction to Git, however it is enough to start using it in your projects.

Additionally, you can find many tutorials and cheat sheets online, for example:

https://git-scm.com/docs/gittutorial

https://training.github.com/downloads/github-git-cheat-sheet/