Software Project Management - Laboratory

Lecture n° 2
A.Y. 2020-2021

Prof. Fabrizio Fornari

Version Control

Use Case

Consider this scenario:

- It is 11:00 PM
- You have a homework submission for today and the assignment is ready for submission
- While testing it, you discover a minor bug and decide to fix it
- You try to fix the bug, you change several lines of code on different files and try to run the code
- You discover now that your code is not working anymore as expected
- You no longer remember what you changed

It is 11:59 PM!!!

Use Case

Consider this scenario:

- It is 11:00 PM
- You have a homework submission for today and the assignment is ready for submission
- While testing it, you discover a minor bug and decide to fix it
- You try to fix the bug, you change several lines of code on different files and try to run the code
- You discover now that your code is not working anymore as expected
- You no longer remember what you changed

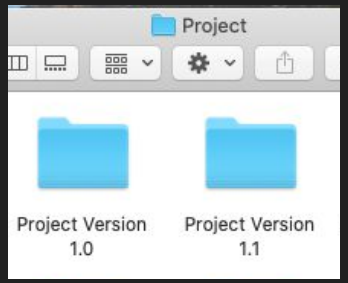
Ctrl + Z is not working!

Version Control

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions

How to do it?

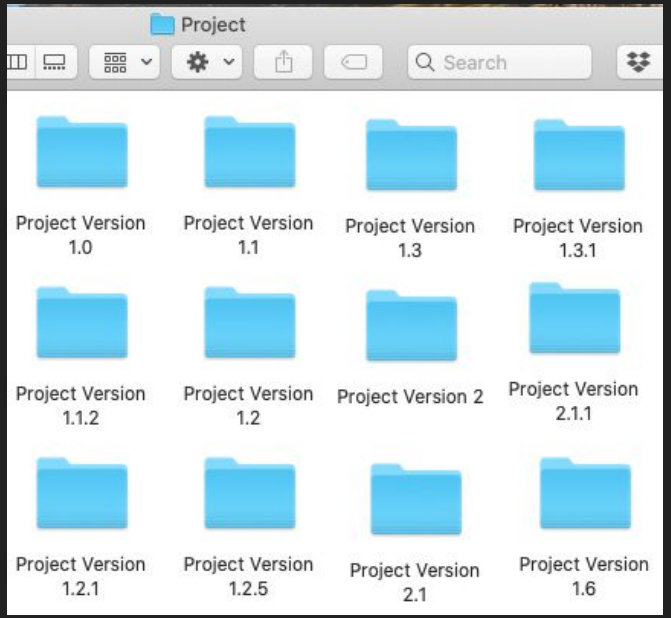
Manually



Any issues?

How to do it?

Manually



How to do it?

Local Version Control Systems

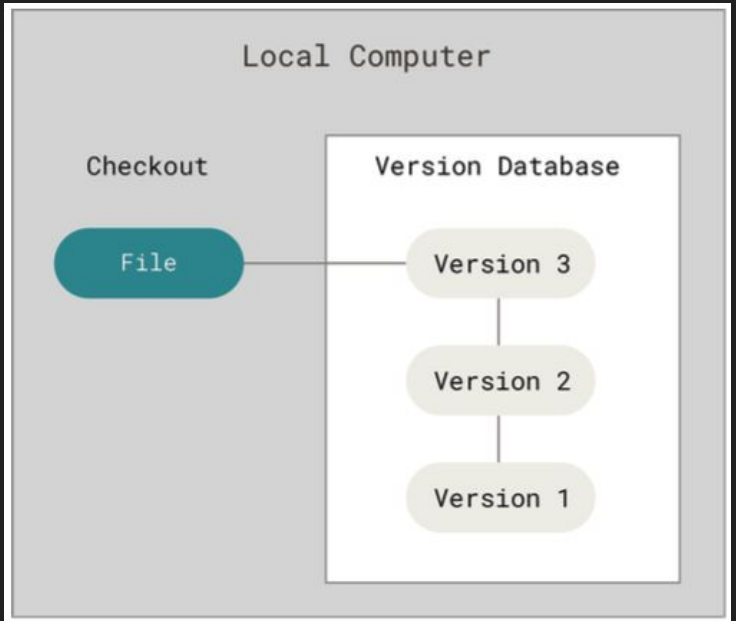


Figure 1. Local version control

How to do it?

Local Version Control Systems

Any issues?

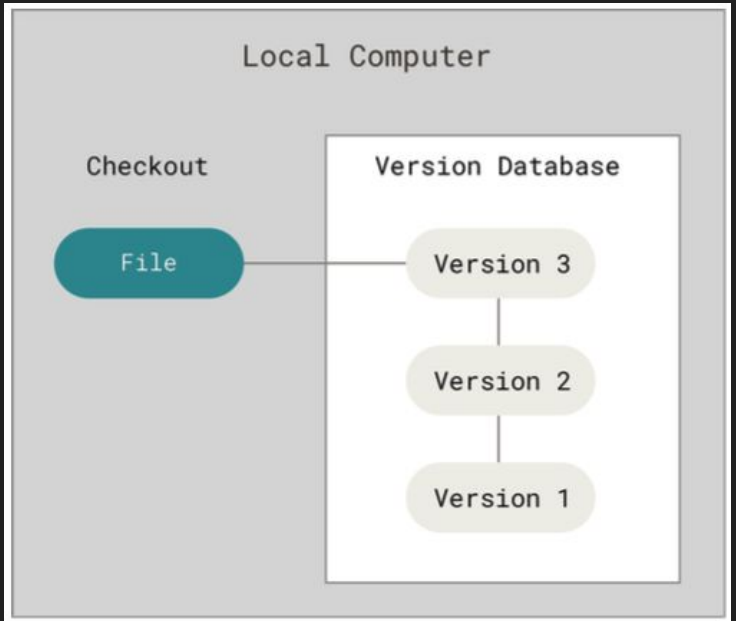


Figure 1. Local version control

How to do it?

Centralized Version Control Systems

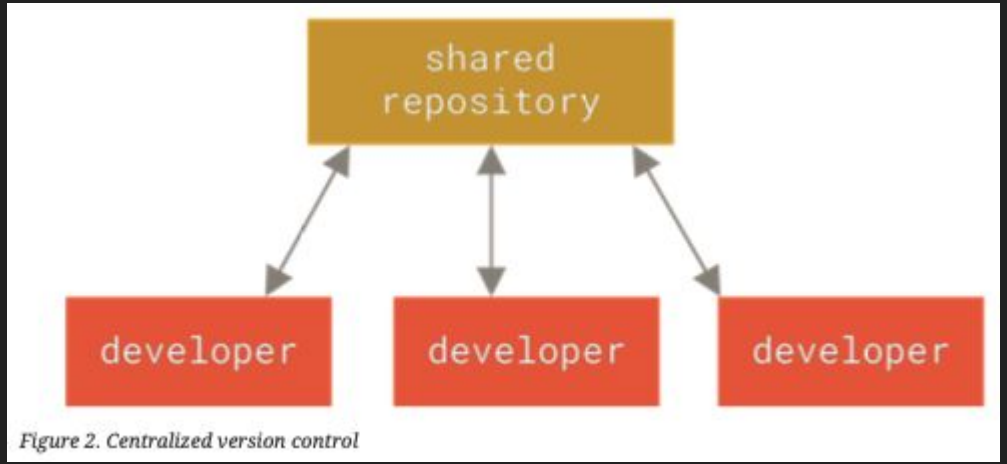


Figure 2. Centralized version control

How to do it?

Centralized Version Control Systems

Any issues?

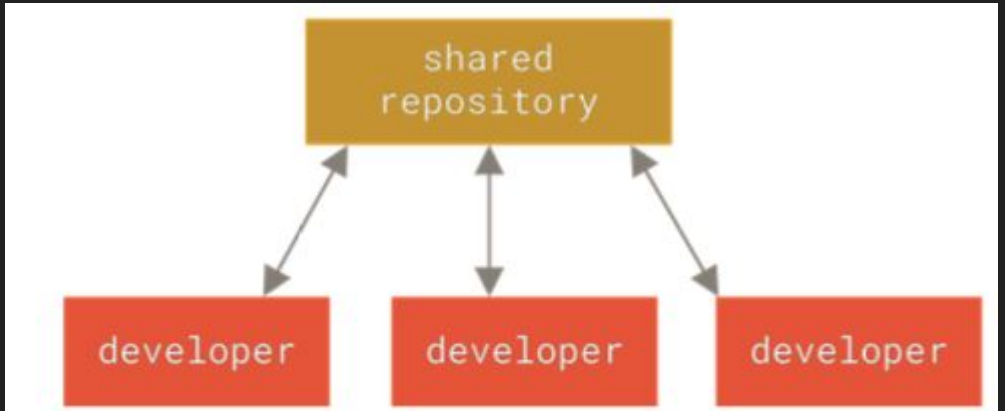


Figure 2. Centralized version control

How to do it?

Distributed Version Control Systems

Distributed means that there is no main server and all of the full history of the project is available once you cloned the project.

Figure 3. Distributed version control

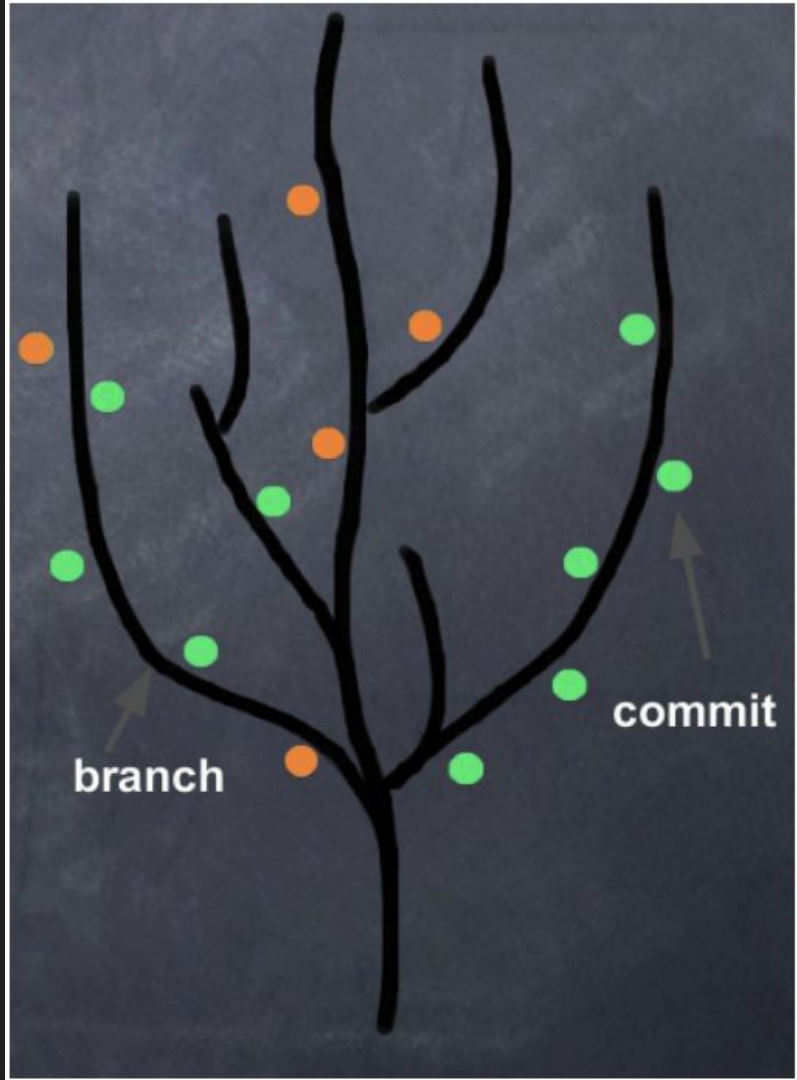
What is git?

- A distributed version control system - DVCS
- Open source project originally developed in 2005 by Linus Torvalds
- A command line utility
- You can imagine git as something that sits on top of your file system and manipulates files.

https://git-scm.com/

Git

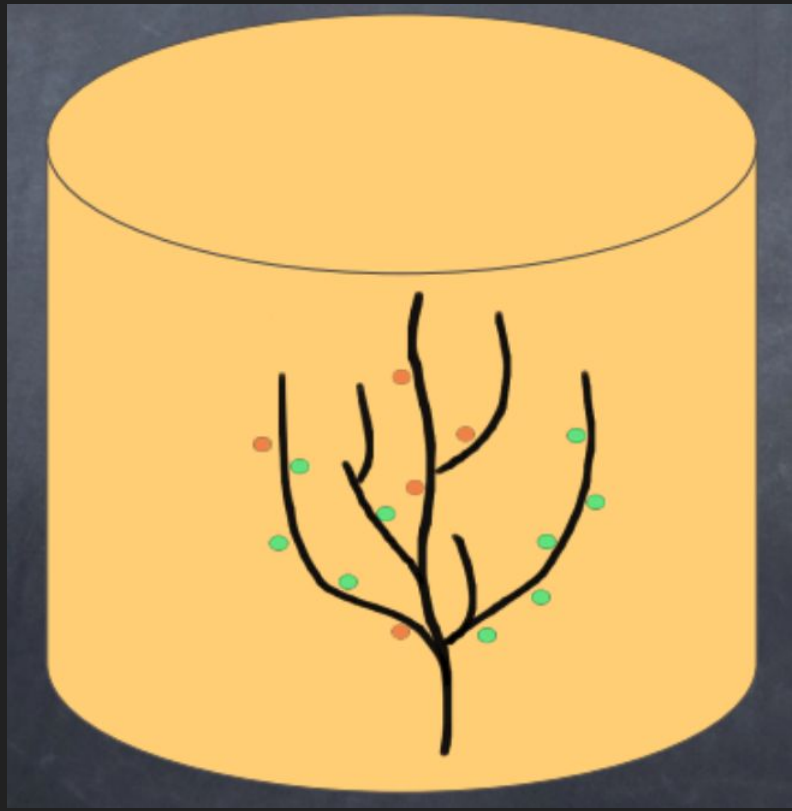
- You can imagine git as something that sits on top of your file system and manipulates files.
- This “something” is a tree structure where each commit creates a new node in that tree.
- Nearly all git commands actually serve to navigate on this tree and to manipulate it accordingly.



Git

The purpose of git is to manage a project, or a set of files, as they change over time. Git stores this information in a data structure called a repository

A git repository contains, mainly: A set of commits



Git - Three States

Git has three main states that your files can reside in:

- Modified - it means you have changed the file but have not committed it to your database yet.
- Staged - it means that you have marked a modified file in its current version to go into your next commit snapshot.
- Committed - it means that you have marked a modified file in its current version to go into your next commit snapshot.

Git - Three Sections

Three main sections of a Git project: the working tree, the staging area, and the Git directory.

Git Workflow

1. Modify file in working directory
2. Stage changes you want to commit
3. Commit, takes the file as they are in the saging area and stores that snapshot permanently to your Git directory

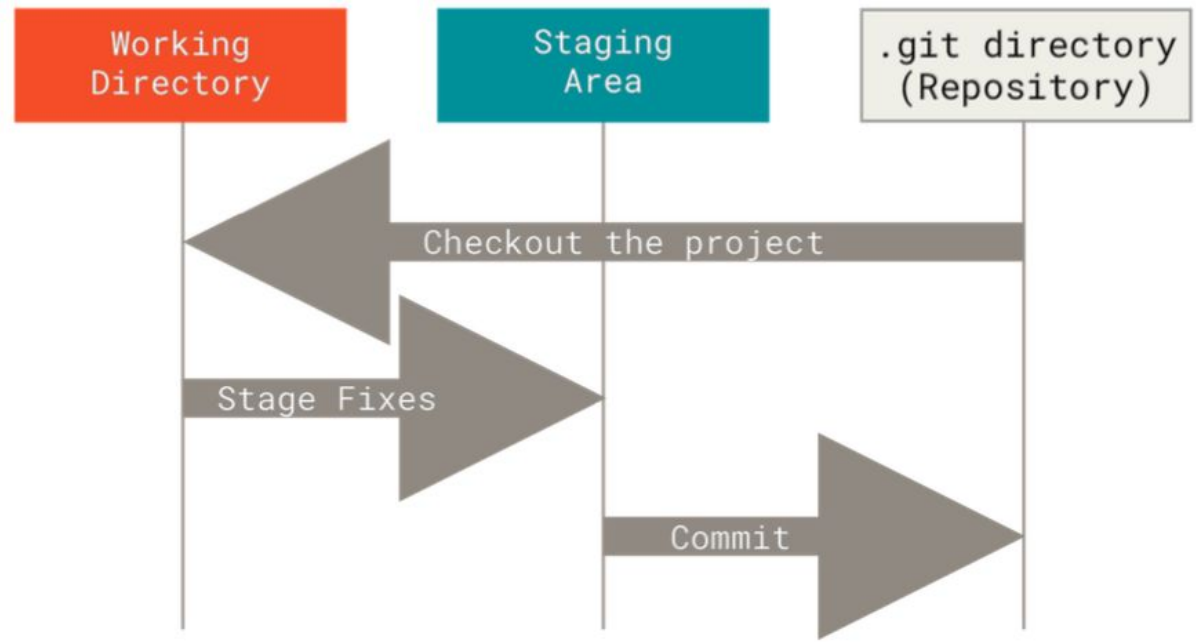


Figure 6. Working tree, staging area, and Git directory

Git - Commit

A commit object mainly contains three things:

- A hash, a 40-character string that uniquely identifies the commit object
- Commit message describing the changes
- A set of changes the commit introduces

What is an hash?

The result of the application of a cryptographic hash function (CHF).

CHF is a mathematical algorithm that maps data of arbitrary size (often called the "message") to a bit array of a fixed size (the "hash value", "hash", or "message digest"). It is a function which is practically infeasible to invert.

Secure Hash Algorithm 1 (SHA1) https://www.hjp.at/(st_a)/doc/rfc/rfc3174.html

Give it a try: http://www.sha1-online.com/

Commits

Third Commit

```
commit c5aefa52cab706d79eea4e67481df8aeea4b1bc1 (HEAD -> main, origin/main, origin/HEAD)
Author: FabrizioFornari89 <fabrizio.fornari@unicam.it>
Date:   Thu Oct 8 17:41:52 2020 +0200

    added a second change to the text

diff --git a/Test.txt b/Test.txt
index cf8637c..1fefa18 100644
--- a/Test.txt
+++ b/Test.txt
@@ -1 +1,2 @@
-First change
\ No newline at end of file
+First change
+Second change
```

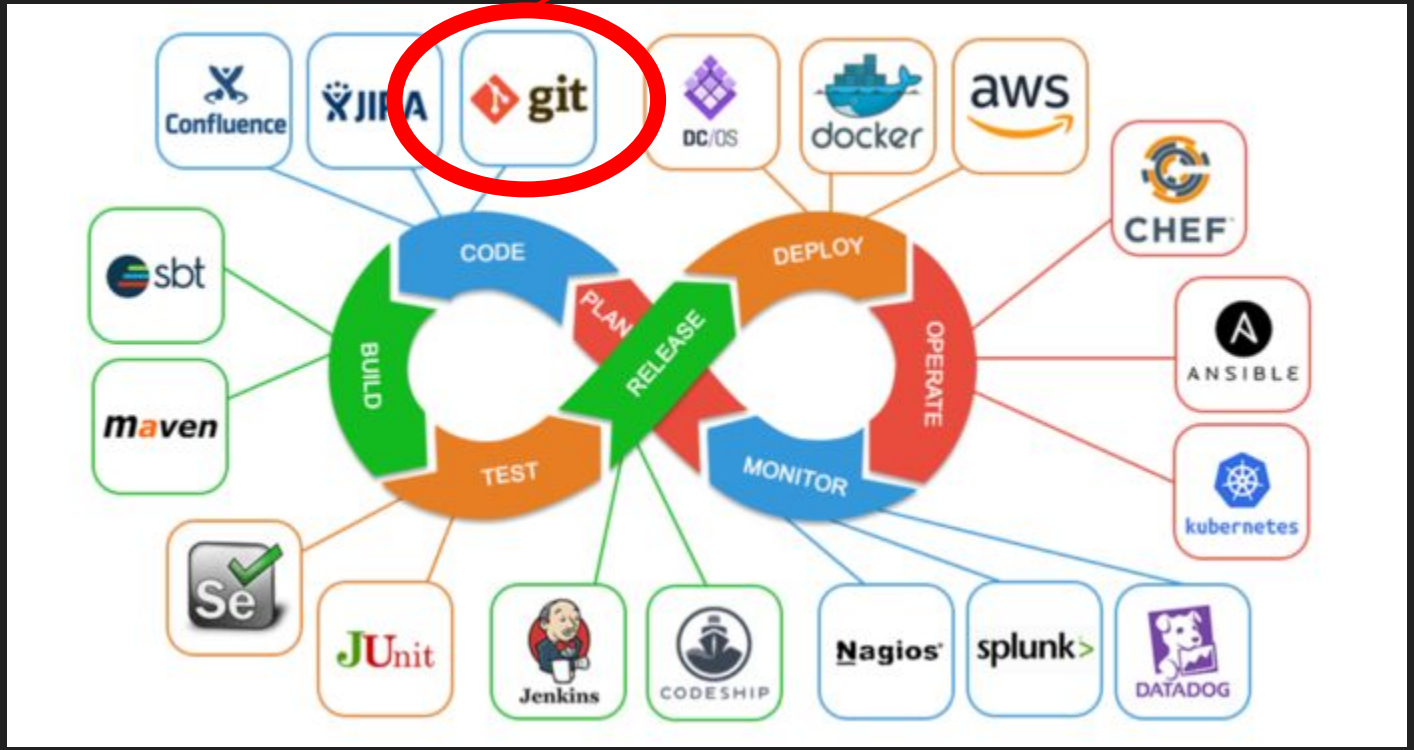
Second Commit

```
commit 69c8fdb551ca8105de8f8ee1b3dd43b5592997d5
Author: FabrizioFornari89 <fabrizio.fornari@unicam.it>
Date:   Thu Oct 8 17:36:43 2020 +0200

    Fixing a change in the text

diff --git a/Test.txt b/Test.txt
index 8b13789..cf8637c 100644
--- a/Test.txt
+++ b/Test.txt
@@ -1 +1 @@
-
+First change
\ No newline at end of file
```

First Commit

```
commit 984dbf2ce07d2fb1524ea6d3fe02fc2d39230564
Author: Fabrizio Fornari <fabrizio.fornari@unicam.it>
Date:   Thu Oct 8 16:08:29 2020 +0200

    Create Test.txt

diff --git a/Test.txt b/Test.txt
new file mode 100644
index 0000000..8b13789
--- /dev/null
+++ b/Test.txt
@@ -0,0 +1 @@
```

DevOps


Our Focus
Confluence
JIRA
git
DC/OS
docker
aws
CHEF
sbt
Maven
Se
JUnit
Jenkins
CODESHIP
Nagios
splunk>
DATADOG
ANSIBLE
kubernetes
CODE
PLAN
BUILD
TEST
RELEASE
DEPLOY
OPERATE
MONITOR

Git Cheat Sheet

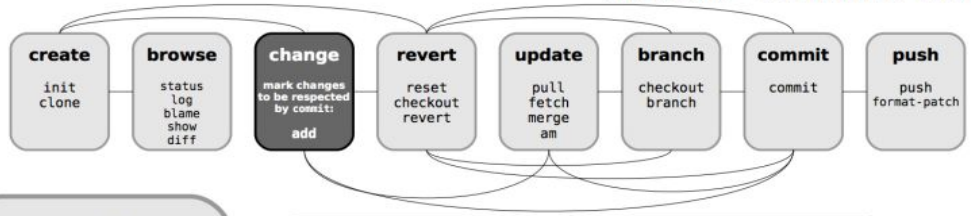
by Jan Krüger <jk@jk.gs>, http://jan-krueger.net/git/
Based on work by Zack Rusin

Basics

Use git help [*command*] if you're stuck.

`master`	default devel branch
`origin`	default upstream branch
`HEAD`	current branch
`HEAD^`	parent of HEAD
`HEAD~4`	great-great grandparent of HEAD
`foo..bar`	from branch *foo* to branch *bar*

(left to right) Command Flow



Create

From existing files

```
git init
git add .
```

From existing repository

```
git clone ~/old ~/new
git clone git://...
git clone ssh://...
```

Publish

In Git, `commit` only respects changes that have been marked explicitly with add.

```
git commit [-a]
```
(-a: add changed files automatically)

```
git format-patch origin
```
(create set of diffs)

```
git push remote
```
(push to origin or *remote*)

```
git tag foo
```
(mark current version)

Useful Tools

```
git archive
```
Create release tarball

```
git bisect
```
Binary search for defects

```
git cherry-pick
```
Take single commit from elsewhere

```
git fsck
```
Check tree

```
git gc
```
Compress metadata (performance)

```
git rebase
```
Forward-port local changes to remote branch

```
git remote add URL
```
Register a new remote repository for this tree

```
git stash
```
Temporarily set aside changes

```
git tag
```
(there's more to it)

```
gitk
```
Tk GUI for Git

Tracking Files

```
git add files
git mv old new
git rm files
git rm --cached files
```
(stop tracking but keep files in working dir)

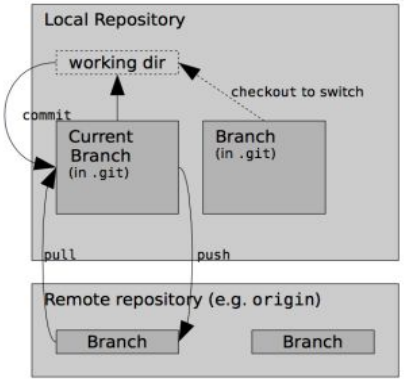
View

```
git status
git diff [oldid newid]
git log [-p] [file|dir]
git blame file
git show id (meta data + diff)
git show id:file
git branch (shows list, * = current)
git tag -l (shows list)
```

Update

```
git fetch (from def. upstream)
git fetch remote
git pull (= fetch & merge)
git am -3 patch.mbox
git apply patch.diff
```

Structure Overview



Revert

In Git, revert usually describes a new commit that undoes previous commits.

```
git reset --hard (NO UNDO)
```
(**reset** to last commit)

```
git revert branch
git commit -a --amend
```
(**replaces** prev. commit)

```
git checkout id file
```

Branch

```
git checkout branch
```
(switch working dir to *branch*)

```
git merge branch
```
(merge into current)

```
git branch branch
```
(branch current)

```
git checkout -b new other
```
(branch *new* from *other* and switch to it)

Conflicts

Use add to mark files as resolved.

```
git diff [--base]
git diff --ours
git diff --theirs
git log --merge
gitk --merge
```

Visual Git Cheat Sheet: https://ndpsoftware.com/git-cheatsheet.html#loc=remote_repo;

Let’s start!

1. Check if you have a version of git installed on your machine $git --version
2. If not, install it https://git-scm.com/book/en/v2/Getting-Started-Installing-Git
3. Set your user name and email address; every Git commit will use this information.

```
$ git config --global user.name "Name Surname"

$ git config --global user.email name.surname@studenti.unicam.it
```

4. You can check your settings at any time:

```
$git config −−l i s t
```

Git Help

If you ever need help while using Git you can get the comprehensive manual page (manpage) help for any of the Git commands by typing:

```
$ git help <verb>
$ git <verb> --help
$ man git-<verb>
```

For example, you can get the manpage help for the git config command by running this:

```
$ git help config
```

If you don’t need the full manpage help, but just a quick refresher on the available options for a Git command you can just type -h:

```
$ git config -h
```

Getting a Git Repository

Typically we obtain a Git repository in one of two ways:

1. Take a local directory that is not currently under version control, and turn it into a Git repository

2. *Clone* an existing Git repository from elsewhere

In either case, you end up with a Git repository on your local machine, ready for work.

Initialize a Repository

Create a new folder and open a terminal in that folder.

$ cd pathToTheFolder/FolderName

$ git init

This creates a new subdirectory named .git that contains all of your necessary repository files — a Git repository skeleton. Git uses this special sub-directory to store all the information about the project . If we ever delete the .git sub-directory, we will lose the project's history.

Type ls -a to see the .git folder (Linux or Mac)

Type dir /a to see the .git folder (Windows)

Git Status

The command used to determine which files are in which state is git status

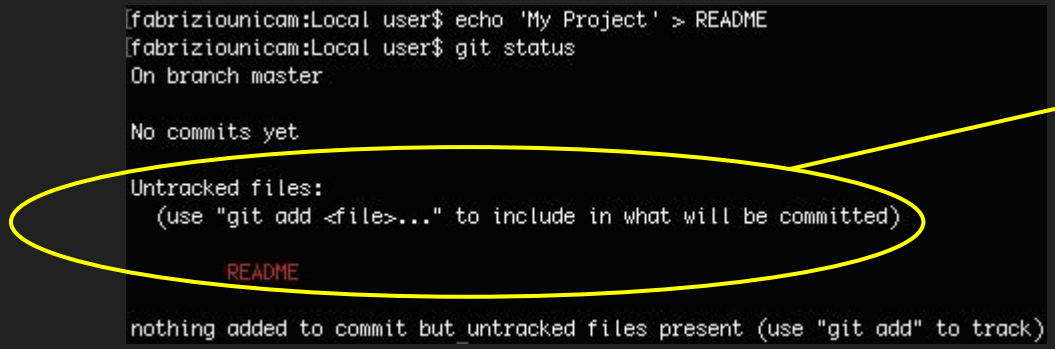
```
fabriziounicam:Local user$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

This means:

- you have a clean working directory.
- no changes have been detected.
- master is the name of the branch.

Tracking a File

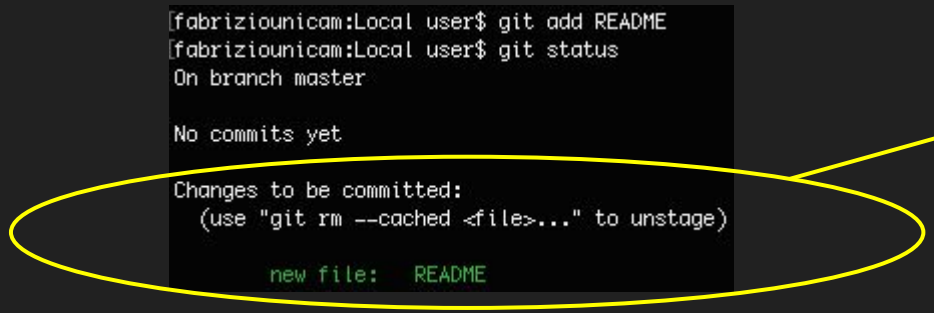
1. Create a file in that folder (by GUI or by command line)



You can see that your new README file is untracked, because it's under the "Untracked files" heading in your status output.

Tracking a File

2. Use the command git add <FileName> to begin tracking the README file



You can see that your README file is staged because it's under the "Changes to be committed" heading.

Tracking a File

3. Modify the README file and run git status

```
fabriziounicam:Local user$ vi README
fabriziounicam:Local user$ git status
On branch master

No commits yet

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

        new file:   README

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:   README
```

Git stages a file exactly as it is when you run the git add command.

If you modify a file after you run git add, you have to run git add again to stage the latest version of the file.

Tracking a File

4. Run git add README and run git status -s

Git stages a file exactly as it is when you run the git add command.

If you modify a file after you run git add, you have to run git add again to stage the latest version of the file.

Tracking a File

5. Create a second File and run git status -s

```
fabriziounicam:Local user$ echo 'My Second File' > SecondFile.txt
fabriziounicam:Local user$ ls
README          SecondFile.txt
fabriziounicam:Local user$ git status -s
A  README
?? SecondFile.txt
```

6. Create a third File and run git status -s

```
fabriziounicam:Local user$ echo 'My Third File' > ThirdFile.txt
fabriziounicam:Local user$ git status -s
A  README
?? SecondFile.txt
?? ThirdFile.txt
```

7. Add all untracked file to staging area by running git add . and run git status

```
fabriziounicam:Local user$ git add .
fabriziounicam:Local user$ git status -s
A  README
A  SecondFile.txt
A  ThirdFile.txt
```

Tracking a File

8. Edit one or more File and run git status -s

```
fabriziounicam:Local user$ git status -s
AD README
A  SecondFile.txt
AM ThirdFile.txt
?? README.txt
```

Can you guess what happened?

```
fabriziounicam:Local user$ git status
On branch master

No commits yet

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

        new file:   README
        new file:   SecondFile.txt
        new file:   ThirdFile.txt

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:    README
        modified:   ThirdFile.txt

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

        README.txt
```

Tracking a File

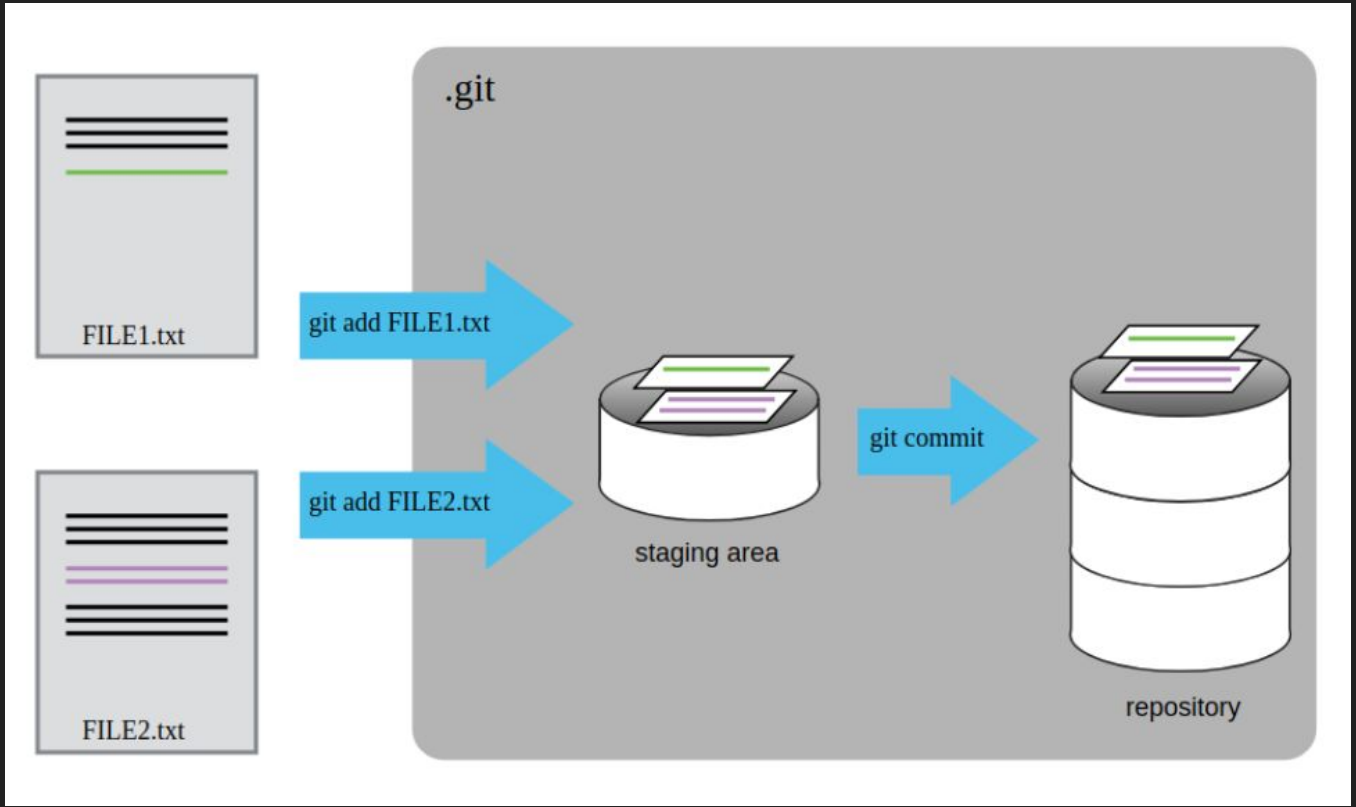
9. Add all the changes to the staging area by running git add . and run git status -s

```
fabriziounicam:Local user$ git add .
fabriziounicam:Local user$ git status -s
A  README.txt
A  SecondFile.txt
A  ThirdFile.txt
```

10. Remove a file from the staging area git rm --cached ThirdFile.txt and run git status -s

```
fabriziounicam:Local user$ git rm --cached ThirdFile.txt
rm 'ThirdFile.txt'
fabriziounicam:Local user$ git status -s
A  README.txt
A  SecondFile.txt
?? ThirdFile.txt
```

Tracking a File



Tracking a File

11. Edit a file and run git status

```
fabriziounicam:Local user$ git status
On branch master

No commits yet

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

        new file:   README.txt
        new file:   SecondFile.txt

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:   README.txt

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

        ThirdFile.txt
```

12. Run git diff

```
fabriziounicam:Local user$ vi README.txt
fabriziounicam:Local user$ git diff
diff --git a/README.txt b/README.txt
index 0956cbe..5d02c56 100644
--- a/README.txt
+++ b/README.txt
@@ -1,2 +1,3 @@
 My Project
 Adding a second line
+Adding a third line
```

diff compares what is in your working directory with what is in your staging area. The result tells you the changes you've made that you haven't yet staged.

Tracking a File

13. Run git diff --staged

```
fabriziounicam:Local user$ git diff --staged
diff --git a/README.txt b/README.txt
new file mode 100644
index 0000000..0956cbe
--- /dev/null
+++ b/README.txt
@@ -0,0 +1,2 @@
+My Project
+Adding a second line
diff --git a/SecondFile.txt b/SecondFile.txt
new file mode 100644
index 0000000..759e250
--- /dev/null
+++ b/SecondFile.txt
@@ -0,0 +1 @@
+My Second File
```

git diff --staged shows what you've staged that will go into your next commit. It compares your staged changes to your last commit.

14. Stage all by running git add .

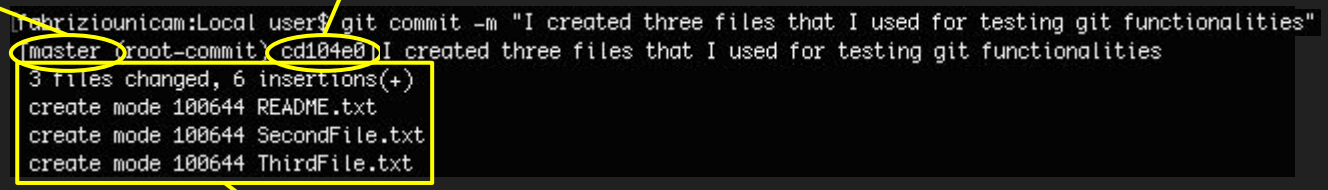
Commit

When your staging area is set up the way you want it, you can commit your changes. Remember, anything that is still unstaged — any files you have created or modified that you haven't run git add on since you edited them — won't go into this commit

15. Run git commit -m "*A message which describes the changes, helping me to remember what i changed with this commit*"

Branch name

SHA-1 checksum



Changed Files and statistics about lines added/removed

Remember

The commit records the snapshot you set up in your staging area.

Anything you didn’t stage is still sitting there modified; you can do another commit to add it to your history.

Every time you perform a commit, you’re recording a snapshot of your project that you can revert to or compare to later.

Skip the Staging Area

Let’s assume you modified a file and you want to commit directly without staging that change

```
fabriziounicam:Local user$ git status
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:   ThirdFile.txt

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

Run git commit -a -m “*A message which describes the changes, helping me to remember what i changed with this commit*”

```
fabriziounicam:Local user$ git commit -a -m "Added one line to ThirdFile"
[master d087029] Added one line to ThirdFile
 1 file changed, 1 insertion(+)
fabriziounicam:Local user$ git status
On branch master
nothing to commit, working tree clean
```

Removing Files

1. Run git rm FileName to remove a file from the working directory and staging the delete

Removing Staged Files

2. What if you staged a file then you realize you don't actually want to commit it? How do you remove a staged file?

git rm --cached FileName

```
fabriziounicam:Local user$ echo 'My Fifth File' > FifthFile.txt
fabriziounicam:Local user$ git add .
fabriziounicam:Local user$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   FifthFile.txt

fabriziounicam:Local user$ git rm FifthFile.txt
error: the following file has changes staged in the index:
    FifthFile.txt
(use --cached to keep the file, or -f to force removal)
fabriziounicam:Local user$ git rm --cached FifthFile.txt
rm 'FifthFile.txt'
fabriziounicam:Local user$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        FifthFile.txt

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

Rename a File

Run git mv OldFileName NewFileName

```
fabriziounicam:Local user$ git mv README.txt FirstFile.txt
fabriziounicam:Local user$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        renamed:    README.txt -> FirstFile.txt
```

Commit History

Run git log

```
fabriziounicam:Local user$ git log
commit d96f4c1a0be06f4762c301650c87e18efe14c772 (HEAD -> master)
Author: FabrizioFornari89 <fabrizio.fornari@unicam.it>
Date:   Mon Oct 12 16:18:37 2020 +0200

    deleted the fourth file

commit da1b4eae420b33430e0bb3eb7c0c4d3d5eeec88f
Author: FabrizioFornari89 <fabrizio.fornari@unicam.it>
Date:   Mon Oct 12 16:15:29 2020 +0200

    added a fourth file

commit d0870297bc9aafe2745cc479a3ddb8029837cd53
Author: FabrizioFornari89 <fabrizio.fornari@unicam.it>
Date:   Mon Oct 12 16:00:01 2020 +0200

    Added one line to ThirdFile

commit cd104e038841cb2866727a0fa400de14b4733b14
Author: FabrizioFornari89 <fabrizio.fornari@unicam.it>
Date:   Mon Oct 12 15:30:46 2020 +0200

    I created try files that I used for testing git functionalities
```

git log lists the commits made in that repository in reverse chronological order

Try:
git log -p -2
git log --stat
git log --pretty=oneline
git log --pretty=format:"%h - %an, %ar : %s"

Undoing Things

I Added a FifthFile and committed it

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

        FifthFile.txt

nothing added to commit but untracked files present (use "git add" to track)
fabriziounicam:Local user$ git add .
fabriziounicam:Local user$ git commit -m "added a FifthFile"
[master 3c955ea] added a FifthFile
 1 file changed, 1 insertion(+)
 create mode 100644 FifthFile.txt
```

I Created a SixthFile

```
fabriziounicam:Local user$ echo "Sixth File" > SixthFile.txt
```

I realized that FifthFile and SixthFile are related and it makes sense to commit them together in a single commit

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

        SixthFile.txt

nothing added to commit but untracked files present (use "git add" to track)
fabriziounicam:Local user$ git add SixthFile.txt
fabriziounicam:Local user$ git commit --amend
```

I run git commit --amend

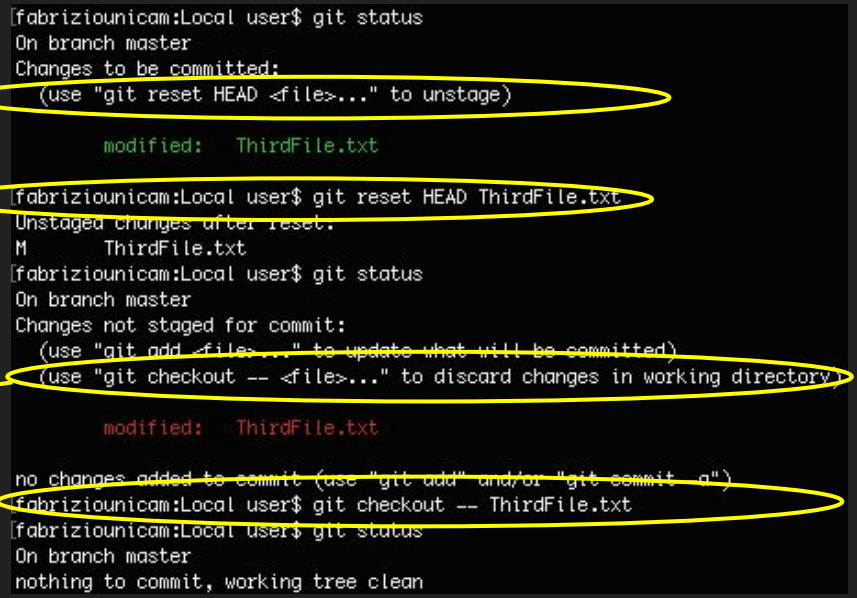
I committed both file together and I also changed the commit message

```
[master dd11a39] added a FifthFile and a SixthFile
 Date: Mon Oct 12 17:07:01 2020 +0200
 2 files changed, 2 insertions(+)
 create mode 100644 FifthFile.txt
 create mode 100644 SixthFile.txt
```

Unstaging & Unmodifying

To unstage run git reset HEAD FileName

To unmodify git checkout -- FileName



Ignoring Things

Often, you'll have a class of files that you don't want Git to automatically add or even show you as being untracked. These are generally automatically generated files such as log files or files produced by your build system. In such cases, you can create a file listing patterns to match them named .gitignore.

```
# ignore all .a files
*.a

# but do track lib.a, even though you're
#ignoring .a files above
!lib.a

# only ignore the TODO file in the current
# directory, not subdir/TODO
/TODO
```


```
# ignore all .a files
*.a

# but do track lib.a, even though you're
#ignoring .a files above
!lib.a

# only ignore the TODO file in the current
# directory, not subdir/TODO
/TODO
```


Example of .gitignore https://github.com/github/gitignore

Ignoring Things

Run cat .gitignore

```
fabriziounicam:Local user$ cat .gitignore
# ignore all .a files
*.a

# but do track lib.a, even though you're ignoring .a files above
!lib.a
fabriziounicam:Local user$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        lib.a

nothing added to commit but untracked files present (use "git add" to track)
fabriziounicam:Local user$ ls
FifthFile.txt   FirstFile.txt   SecondFile.txt  SixthFile.txt   ThirdFile.txt   lib.a           testIgnore.a
```