[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



GIT for Beginners

Anthony Baire

Universit´e de Rennes 1 / UMR IRISA

June 2, 2022



This tutorial is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 France License

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Objectives

* Understand the basics about version control systems
* Getting started with GIT
  + working with a local repository
  + synchronising with a remote repository
  + setting up a server

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Summary

1. About Version Control Tools
2. Overview of GIT
3. Working locally
4. Branching & merging
5. Interacting with a remote repository
6. Administrating a server
7. Working with third-party contributors
8. Extras

[Ve](#_bookmark0)[rsion Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



# Part 1.

About Version Control Tools

* + Definition
  + Use cases
  + Base concepts
  + History

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## What is a version control system ?

From: <http://en.wikipedia.org/wiki/Revision_control>

*Revision control* [...] *is the management of changes to documents, computer programs, large web sites, and other collections of information.*

*Changes are usually identified by a number or letter code, termed the ”revision number”* [...]. *For example, an initial set of files is ”revision 1”. When the first change is made, the resulting set is ”revision 2”, and so on.*

*Each revision is associated with a timestamp and the person making the change.*

*Revisions can be compared, restored, and with some types of files, merged.*

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Use case 1: keeping an history

The life of your software/article is recorded from the beginning

* + at any moment you can revert to a previous revision 1
  + the history is browseable, you can inspect any revision 2
    - when was it done ?
    - who wrote it ?
    - what was changed ?
    - why ?
    - in which context ?
  + all the deleted content remains accessible in the history

1let’s say your not happy with your latest changes

2this is useful for understanding and fixing bugs

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Use case 2: working with others

VC tools help you to:

* + share a collection of files with your team
  + merge changes done by other users
  + ensure that nothing is accidentally overwritten
  + ~~know who you must blame when something is broken~~

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Use case 3: branching

You may have multiple variants of the same software, materialised as **branches**, for example:

* + a main branch
  + a maintainance branch *(to provide bugfixes in older releases)*
  + a development branch *(to make disruptive changes)*
  + a release branch *(to freeze code before a new release)*

VC tools will help you to:

* + handle multiple branches concurrently
  + merge changes from a branch into another one

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Use case 4: working with external contributors

VC tools help working with third-party contributors:

* + it gives them visibility of what is happening in the project
  + it helps them to submit changes (patches) and it helps you to integrate these patches
  + forking the development of a software and merging it back into mainline3

3decentralised tools only

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Use case 5: scaling

Some metrics4 about the Linux kernel (developed with GIT):

* + about 10000 changesets in each new version (every 2 or 3 months)
  + 1000+ unique contributors

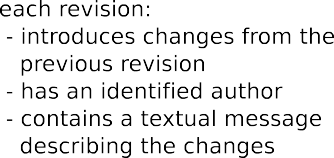
4source: the Linux Foundation

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Some illustrations



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Some illustrations



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Some illustrations



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



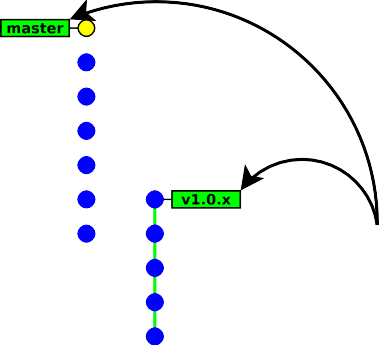
## Some illustrations



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Some illustrations



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Some illustrations



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



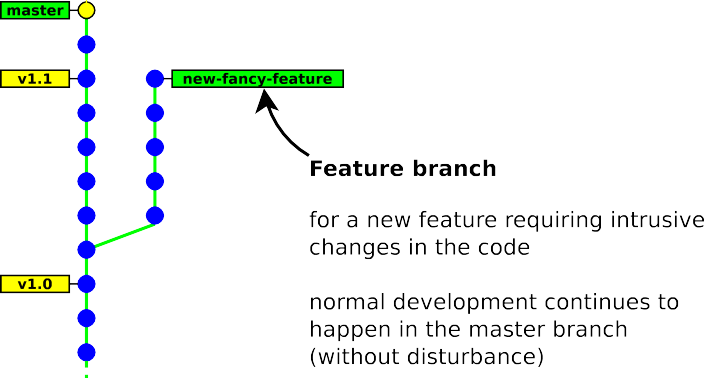
## Some illustrations



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



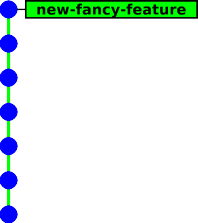
## Some illustrations



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



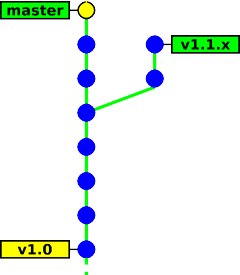
## Some illustrations



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



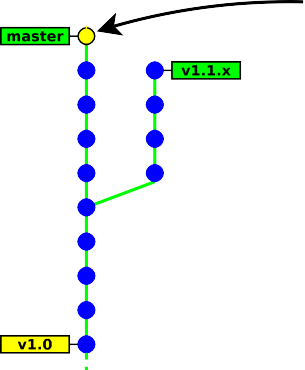
## Some illustrations



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



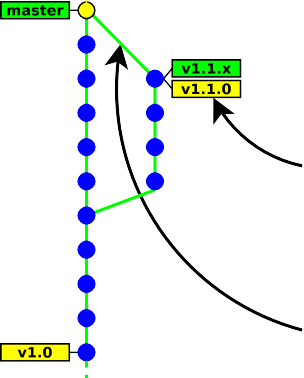
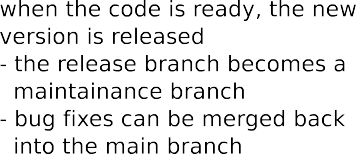
## Some illustrations



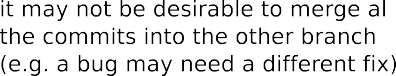
[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Some illustrations



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Some illustrations

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Taxinomy

Architecture:

* **centralised** *→* everyone works on the same unique repository
* **decentralised** *→* everyone works on his own repository Concurrency model:
* **lock before edit** (mutual exclusion)
* **merge after edit** (may have conflicts) History layout:
* **tree** (merges are not recorded)
* **direct acyclic graph**

Atomicity scope: **file** vs **whole tree GIT**

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Other technical aspects

**Space efficiency**: storing the whole history of a project requires storage space *(storing every revision of every file)*

*→* most VC tools use delta compression to optimise the space

*(except Git which uses object packing instead)*

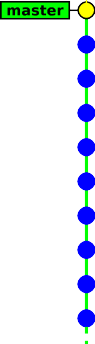
**Access method**: A repository is identified with a URL. VC tools offer multiple ways of interacting with remote repositories.

* dedicated protocol (*svn:// git://*)
* direct access to a local repository (*file://path* or just *path*)
* direct access over SSH (*ssh:// git+ssh:// svn+ssh://*)
* over http (*http:// https://*)

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



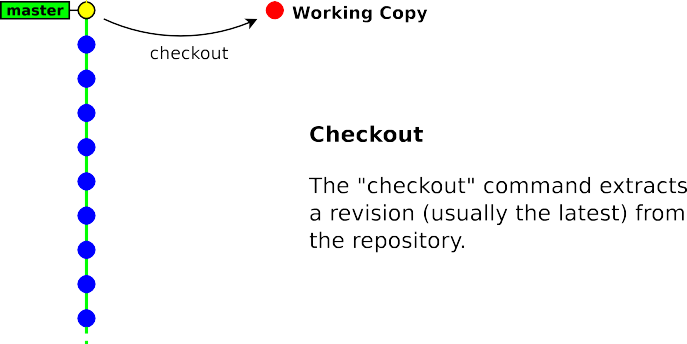
## Creating new revisions



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



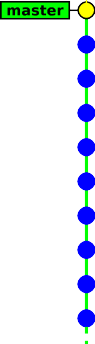
## Creating new revisions



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



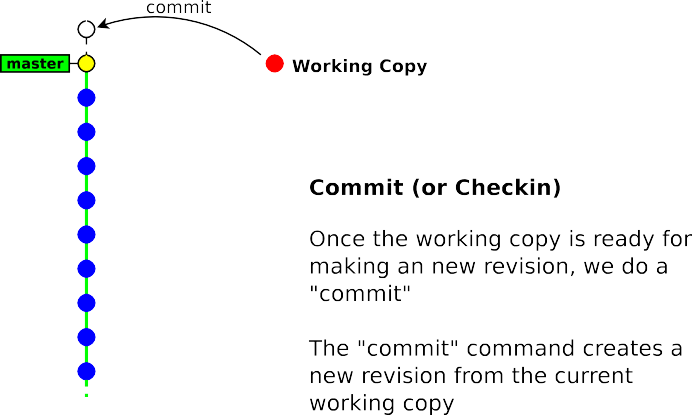
## Creating new revisions



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



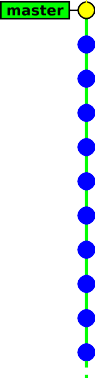
## Creating new revisions



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



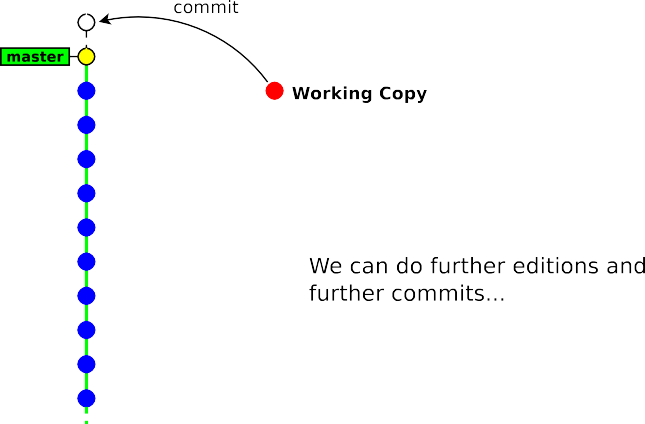
## Creating new revisions



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Creating new revisions



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## What shall be stored into the repository ?

You should store all files that are not generated by a tool:

* source files (.c .cpp .java .y .l .tex . . . )
* build scripts / project files (Makefile configure.in Makefile.am CMakefile.txt wscript .sln)
* documentation files (.txt README . . . )
* resource files (images, audio, . . . )

You should not store generated files

(*or you will experience many unneccessary conflicts*)

* .o .a .so .dll .class .jar .exe .dvi .ps .pdf
* source files / build scripts when generated by a tool (like autoconf, cmake, lex, yacc)

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Guidelines for committing

* commit often
* commit independent changes in separate revisions
* in commit messages, describe the rationale behind of your changes (*it is often more important than the change itself*)

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## History (Centralised Tools)

* 1*st* generation *(single-file, local-only, lock-before-edit)*
  + 1972: **SCCS**
  + 1982: **RCS**
  + 1985: PVCS
* 2*nd* generation *(multiple-files, client-server, merge-before-commit)*
  + 1986: **CVS**
  + 1992: Rational ClearCase
  + 1994: Visual SourceSafe
* 3*rd* generation *(+ repository-level atomicity)*
  + 1995: Perforce
  + 2000: **Subversion**

*•* + many others

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## History (Decentralised tools)



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



# Part 2. Overview of GIT

* History
* Git’s design & features
* User interfaces

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## History

* before 2005: Linux sources were managed with Bitkeeper (proprietary DVCS tool) 5
* April 2005: revocation of the free-use licence (because of some reverse engineering)
* No other tools were enough mature to meet Linux’s dev constraints (distributed workflow, integrity, performance).

*⇒* Linus Torvald started developing Git

* June 2005: first Linux release managed with Git
* December 2005: Git 1.0 released

5now open source! (since 2016)

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Git Design objectives

* distributed workflow (decentralised)
* easy merging (merge deemed more frequent than commit)
* integrity (protection against accidental/malicious corruptions)
* speed & scalability
* ~~ease of use~~

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Git Design choices

* Easily hackable
  + simple data structures (blobs, trees, commits, tags)
  + no formal branch history

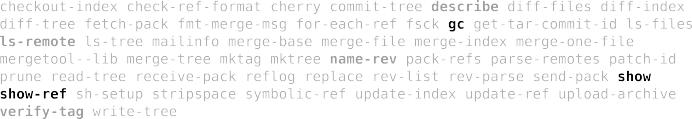
(a branch is just a pointer to the last commit)

* + low-level commands exposed to the user
* Integrity
  + cryptographic tracking of history (SHA-1 hashes)
  + tag signatures (GPG)
* Merging
  + pluggable merge strategies
  + staging area (index)
* Performance
  + no delta encoding

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



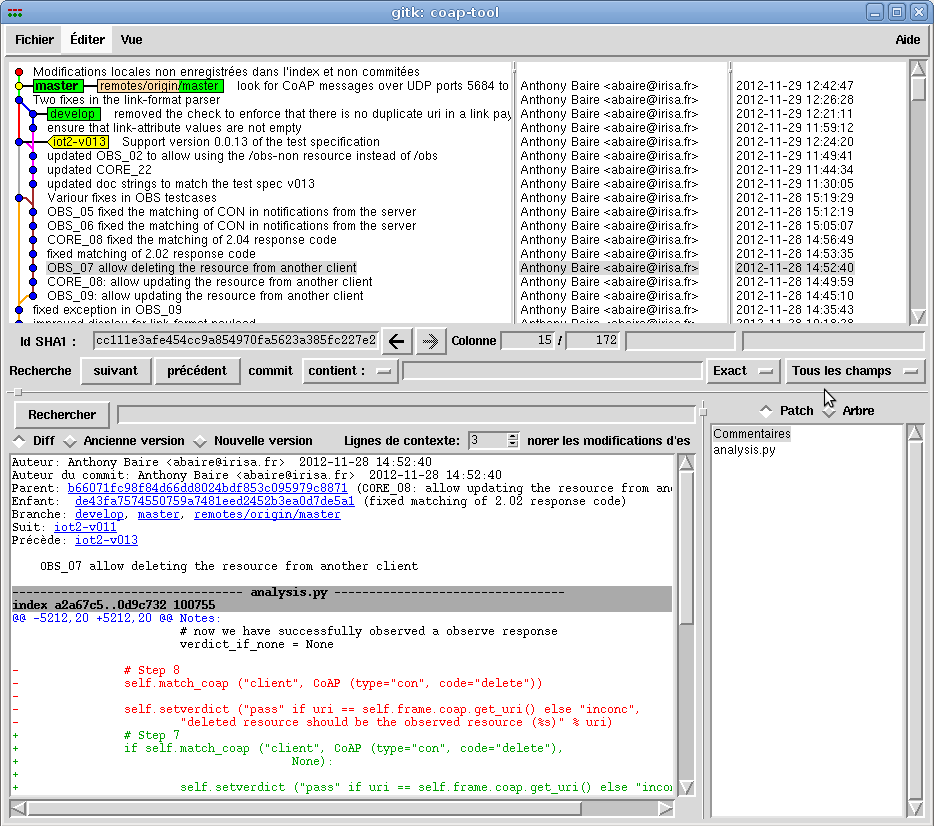
## Git Commands



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



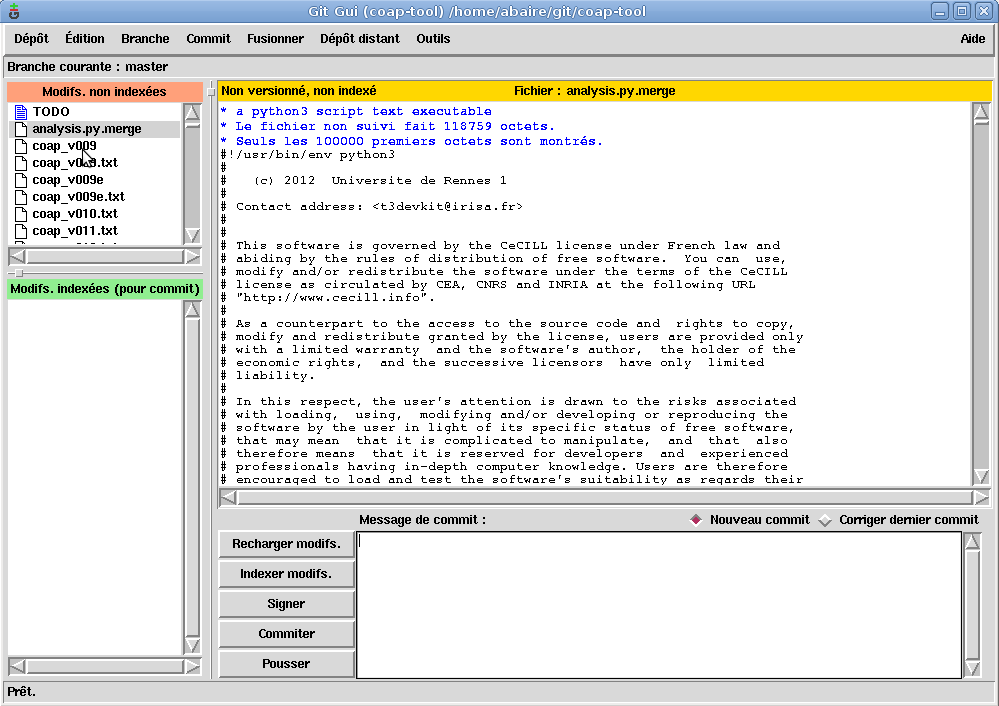
## Git GUIs: gitk *→* browsing the history



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Git GUIs: git gui *→* preparing commits



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## 3rd party GUIs

* Turtoise git (Windows)
* GitUp, Gitx (MacOS-X)
* Smartgit (java, multiplatform)
* Eclipse git plugin

[Ve](#_bookmark0)[rsion Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



# Part 3. Working locally

* creating a repository
* adding & committing files
* the staging area (or index)

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Create a new repository

git init *myrepository*

This command creates the directory *myrepository*.

* the repository is located in *myrepository*/.git
* the (initially empty) working copy is located in *myrepository*/

*, ,*

### $ pwd

/tmp

### $ git init helloworld

Initialized empty Git repository in /tmp/helloworld/.git/

### $ ls -a helloworld/

. .. .git

### $ ls helloworld/.git/

branches config description HEAD hooks info objects refs

*z r*

**Note:** The /.git/ directory contains your whole history,

 **do not delete it**6

6unless your history is merged into another repository

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Commit your first files

git add *file*

git commit [ -m *message* ]

*, ,*

**$ cd helloworld**

**$ echo 'Hello World!' > hello**

**$ git add hello**

**$ git commit -m "added file 'hello'"**

[master (root-commit) e75df61] added file 'hello'

1 files changed, 1 insertions(+), 0 deletions(-) create mode 100644 hello

*z r*

**Note:** “master” is the name of the default branch created by

git init

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## The staging area (aka the “index”)

Usual version control systems provide two spaces:

* the **repository**

*(the whole history of your project)*

* the **working tree** (or **local copy**)

*(the files you are editing and that will be in the next commit)*

Git introduces an intermediate space : the **staging area**

(also called **index**)

The index stores the files scheduled for the next commit:

* git add *files →* copy files into the index
* git commit *→* commits the content of the index

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## The staging area (aka the “index”)







[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Update a file

*, ,*

### $ echo 'blah blah blah' >> hello

**$ git commit**

# On branch master

# Changed but not updated:

# (use "git add <file>..." to update what will be committed)

# (use "git checkout -- <file>..." to discard changes in working directory) #

# modified: hello #

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

*z r*

Git complains because the index is unchanged (nothing to commit)

*→* We need to run git add to copy the file into the index

*, ,*

### $ git add hello

**$ git commit -m "some changes"**

[master f37f2cf] some changes

1 files changed, 1 insertions(+), 0 deletions(-)

*z r*

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Bypassing the index7

Running git add & git commit for every iteration is tedious. GIT provides a way to bypass the index.

git commit *file1* [ *file2 . . .* ]

This command commits files (or dirs) directly from the working tree

**Note:** when bypassing the index, GIT ignores new files:

* “git commit .” commits only files that were present in the last commit (updated files)
* “git add . && git commit” commits everything in the working tree (including new files)

7also named “partial commit”

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Bypassing the index







[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Deleting files

git rm *file*

*→* remove the file from the index and from the working copy

git commit

*→* commit the index

*, ,*

### $ git rm hello

rm 'hello'

### $ git commit -m "removed hello"

[master 848d8be] removed hello

1. files changed, 0 insertions(+), 3 deletions(-) delete mode 100644 hello

*z r*

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Showing differences

*→* shows the differences between two revisions *rev a* and *rev b*

git diff [ *rev a* [ *rev b* ] ] [ -- *path . . .* ]

(*in a format suitable for the* patch *utility*)

* by default *rev a* is the **index**
* by default *rev b* is the **working copy**

*→* shows the differences between *rev a* and the index

git diff --staged [ *rev a* ] [ -- *path . . .* ]

* by default *rev a* is HEAD *(a symbolic references pointing to the last commit)*

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## About git diff and the index



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



*,*

**$ echo foo >> hello**

**$ git add hello**

**$ echo bar >> hello**

## Diff example *,*

### $ git diff

--- a/hello

+++ b/hello

@@ -1,2 +1,3 @@ Hello World! foo

+bar

### $ git diff --staged

--- a/hello

+++ b/hello @@ -1 +1,2 @@ Hello World!

+foo

### $ git diff HEAD

--- a/hello

+++ b/hello @@ -1 +1,3 @@ Hello World!

+foo

+bar

*z r*38 / 96

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Resetting changes

git reset [ --hard ] [ -- *path . . .* ]

git reset cancels the changes in the index (and possibly in the working copy)

* git reset drops the changes staged into the index8, but the working copy is left intact
* git reset --hard drops all the changes in the index **and** in the working copy

8it restores the files as they were in the last commit

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Resetting changes in the working copy

git checkout -- *path*

This command restores a file (or directory) as it appears in the index (thus it drops all unstaged changes)

*, ,*

### $ git diff HEAD

--- a/hello

+++ b/hello @@ -1 +1,3 @@ Hello World!

+foo

+bar

### $ git checkout -- .

**$ git diff HEAD**

--- a/hello

+++ b/hello @@ -1 +1,2 @@ Hello World!

+foo

*z r*

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Other local commands

* git status *→* show the status of the index and working copy
* git show *→* show the details of a commit (metadata + diff)
* git log *→* show the history
* git mv *→* move/rename a file9
* git tag *→* creating/deleting tags (to identify a particular revision)

9note that git mv is strictly equivalent to: “cp *src dst* && git rm *src* && git add *dst*” (file renaming is not handled formally, but heuristically)

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Exercises

1. create a new repository
2. create a new file, add it to the index and commit it
3. launch gitk to display it. Keep the window open and hit F5 after each command (to visualise the results of your commands)
4. modify the file and make a new commit
5. rename the file (either with git mv or git add+git rm), do a git status

before committing (to ensure the renaming is correctly handled)

1. delete the file and commit it
2. create two new files and commit them. Then modify their content in the working copy and display the changes with git diff
3. add one file into the index but keep the other one. Display the changes between:
   * the index and the working copy
   * the last commit and the index
   * the last commit and the working copy
4. run git reset to reset the index
5. run git reset --hard to reset the index and the working copy

[Ve](#_bookmark0)[rsion Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



# Part 4.

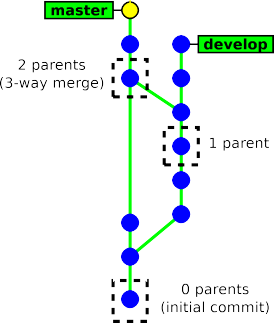
Branching & merging

* How GIT handles its history
* Creating new branches
* Merging & resolving conflicts

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## How GIT handles its history

Each **commit** object has a list of **parent commits**:

* 0 parents *→* initial commit
* 1 parent *→* ordinary commit
* 2+ parents *→* result of a merge

*→* This is a Direct Acyclic Graph

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## How GIT handles its history

* There is no formal “branch history”

*→* a **branch** is just a pointer on the latest commit.

(*git handles branches and tags in the same way internally*)

* Commits are identified with **SHA-1 hash** (160 bits) computed from:
  + the commited files
  + the meta data (commit message, author name, . . . )
  + the hashes of the parent commits

*→* A commit id (hash) identifies **securely** and **reliably** its content and all the previous revisions.

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Creating a new branch

* *new branch* is the name of the new branch

git checkout -b *new branch* [ *starting point* ]

* *starting point* is the starting location of the branch (possibly a commit id, a tag, a branch, . . . ). If not present, git will use the current location.

*, ,*

### $ git status

# On branch master

nothing to commit (working directory clean)

### $ git checkout -b develop

Switched to a new branch 'develop'

### $ git status

# On branch develop

nothing to commit (working directory clean)

*z r*

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Switching between branches

git checkout [-m] *branch name*

*,*

*,*

### $ git status

# On branch develop

nothing to commit (working directory clean)

### $ git checkout master

Switched to branch 'master'

*z r*

**Note:** it may fail when the working copy is not clean. Add -m to request merging your local changes into the destination branch.

*, ,*

### $ git checkout master

error: Your local changes to the following files would be overwritten by checkout: hello Please, commit your changes or stash them before you can switch branches.

Aborting

### $ git checkout -m master

M hello

Switched to branch 'master'

*z r*

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Merging a branch

This will merge the changes in *other branch* into the current branch.

git merge *other branch*

*, ,*

### $ git status

# On branch master

nothing to commit (working directory clean)

### $ git merge develop

Merge made by recursive. dev | 1 +

hello | 4 +++-

1. files changed, 4 insertions(+), 1 deletions(-) create mode 100644 dev

*z r*

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Notes about merging

* The result of git merge is immediately committed (unless there is a conflict)
* The new commit object has **two parents**.

*→* the merge history is recorded

* git merge applies only the changes since the last common ancestor in the other branch.

*→* if the branch was already merged previously, then only the changes since the last merge will be merged.

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Branching example



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Branching example





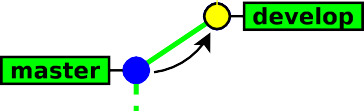


[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Branching example



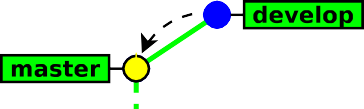


[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Branching example



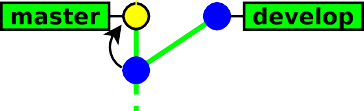


[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Branching example



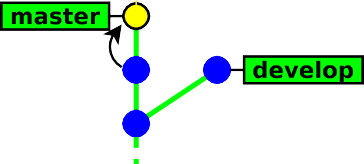


[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Branching example



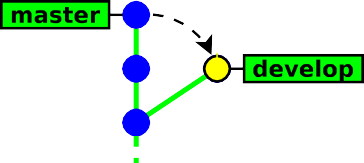


[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Branching example



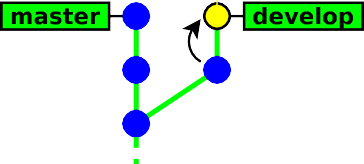


[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Branching example



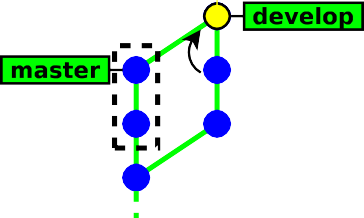


[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Branching example



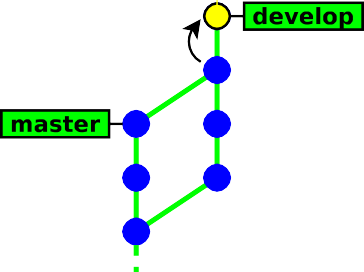


[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Branching example



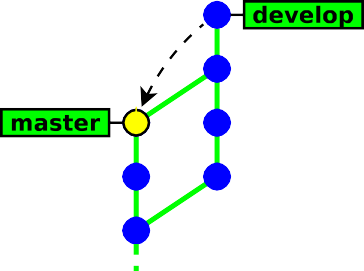


[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Branching example



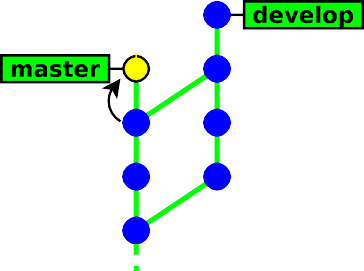


[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Branching example

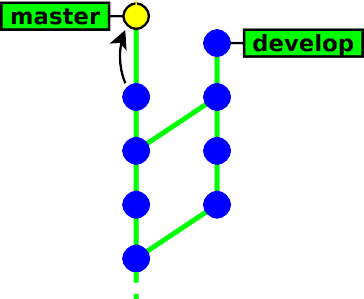




[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Branching example

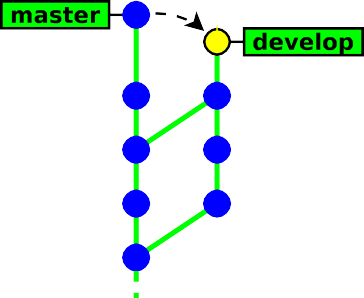




[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Branching example

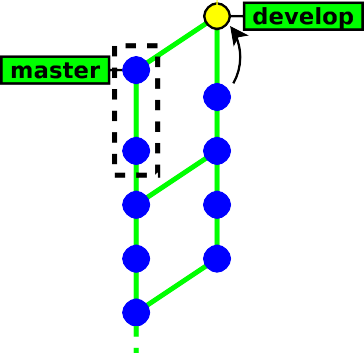




[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Branching example

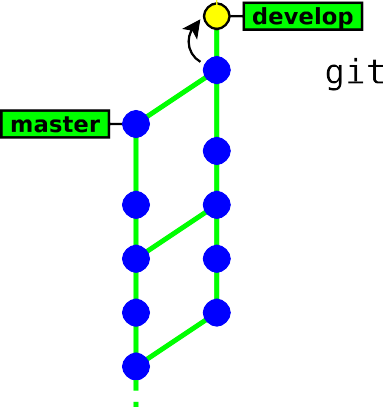




[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Branching example

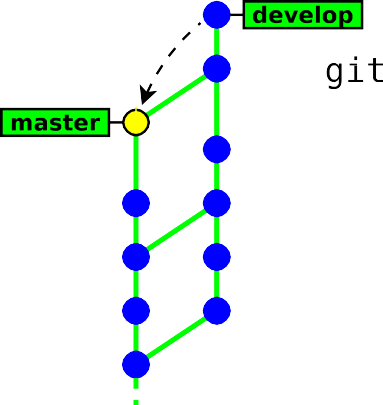




[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Branching example

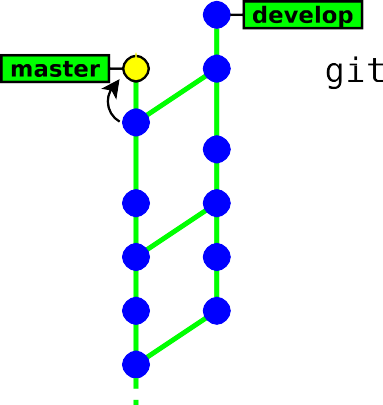




[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Branching example

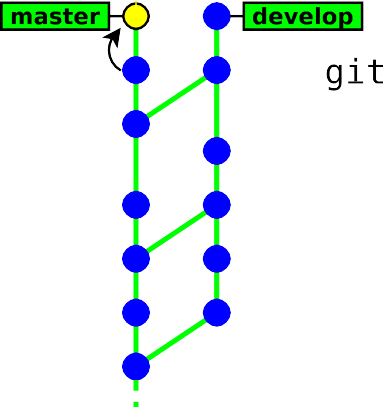




[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Branching example

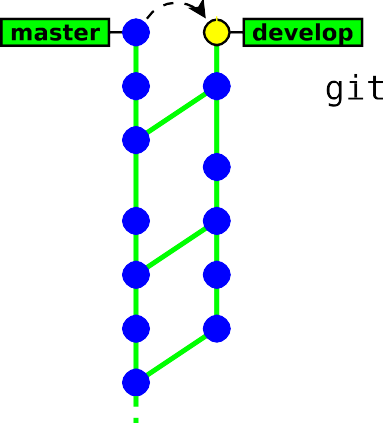




[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Branching example

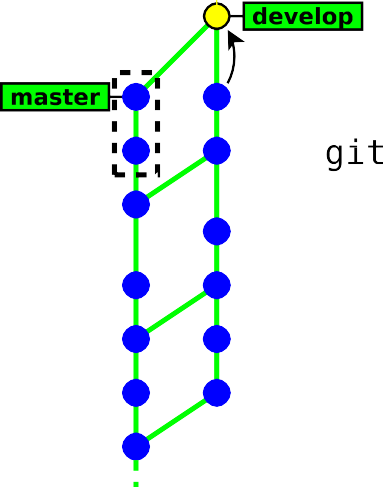




[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Branching example

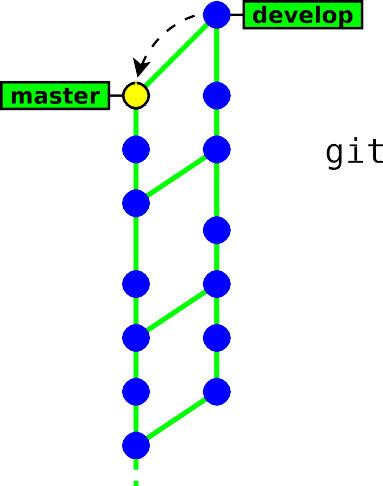




[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Branching example

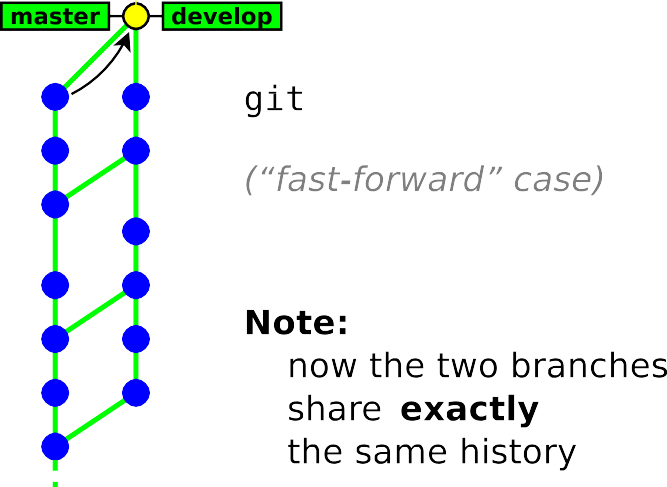




[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



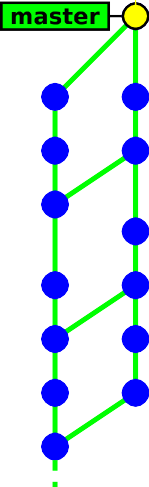
## Branching example



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Branching example





[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## How Git merges files ?

If the same file was independently modified in the two branches, then Git needs to merge these two variants

* **textual files** are merged on a per-line basis:
  + lines changed in only one branch are automatically merged
  + if a line was modified in the two branches, then Git reports a conflict. Conflict zones are enclosed within <<<<<<< >>>>>>>

*, ,*

Here are lines that are either unchanged from the common ancestor, or cleanly resolved because only one side changed.

<<<<<<< yours:sample.txt Conflict resolution is hard; let's go shopping.

=======

Git makes conflict resolution easy.

>>>>>>> theirs:sample.txt

And here is another line that is cleanly resolved or unmodified.

*z r*

* **binary files** always raise a conflict and require manual merging

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Merge conflicts

In case of a conflict:

* **unmerged files** (those having conflicts) are left **in the working tree** and marked as “unmerged”10
* **the other files** (free of conflicts) and the metadata (commit message, parents commits, ...) are automatically added **into the index** (the staging area)

10Git will refuse to commit the new revision until all the conflicts are explicitely resolved by the user

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Resolving conflicts

There are two ways to resolve conflicts:

* either edit the files manually, then run

git add *file →* to check the file into the index or

git rm *file →* to delete the file

* or with a conflict resolution tool(xxdiff, kdiff3, emerge, ...)

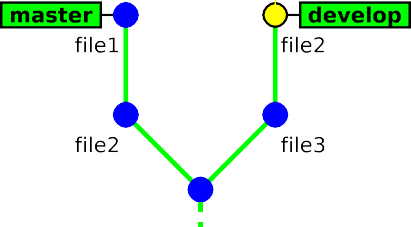
git mergetool [ *file* ]

Then, once all conflicting files are checked in the index, you just need to run git commit to commit the merge.

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Conflict example



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Conflict example



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



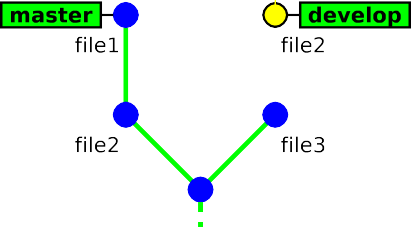
## Conflict example



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



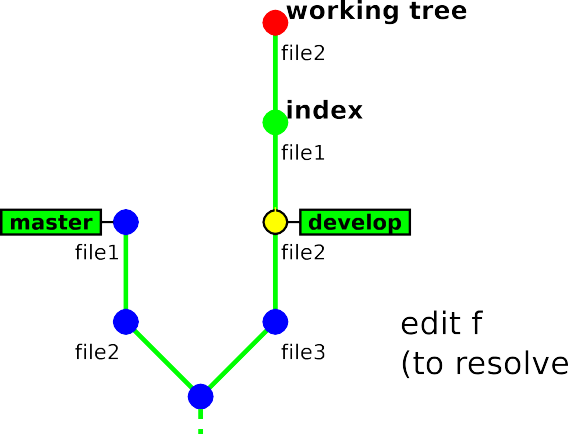
## Conflict example



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



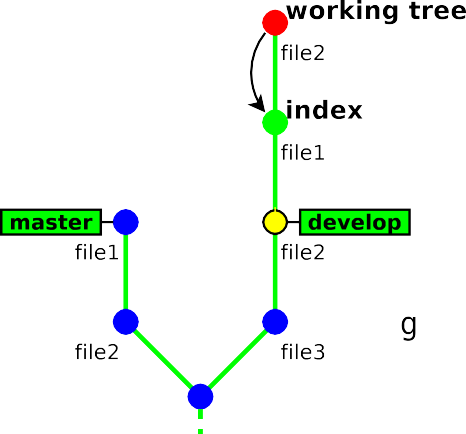
## Conflict example



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



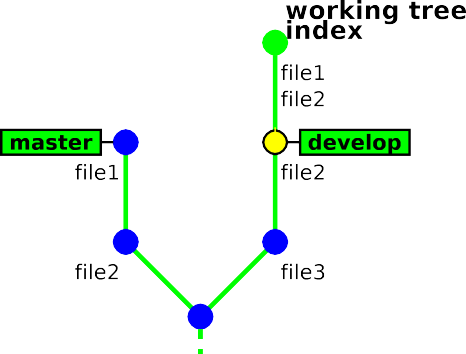
## Conflict example



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



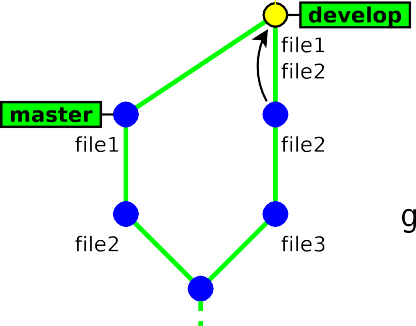
## Conflict example



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Conflict example



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Deleting branches

git branch -d *branch name*

This command has some restrictions, it cannot delete:

* the current branch (HEAD)
* a branch that has not yet been merged into the current branch

*, ,*

### $ git branch -d feature-a

Deleted branch feature-a (was 45149ea).

### $ git branch -d feature-b

error: The branch 'feature-b' is not fully merged.

If you are sure you want to delete it, run 'git branch -D feature-b'.

### $ git branch -d master

error: Cannot delete the branch 'master' which you are currently on.

*z r*

*→* git branch -d is safe11

11unlike git branch -D which **deletes unconditionnally** () the branch

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Exercises

1. use “gitk --all” to display all branches

*(and remember to hit F5 after each command to visualise the changes)*

1. create a new branch named “develop”
2. make some commits in this branch
3. go back to branch “master” and make some commits
4. merge branch “develop” into “master”
5. make a new commit in each branch so as to generate a conflict (edit the same part of a file)
6. merge branch “develop” into “master”, and fix the conflict
7. merge “master” into “develop”

[Ve](#_bookmark0)[rsion Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



# Part 5.

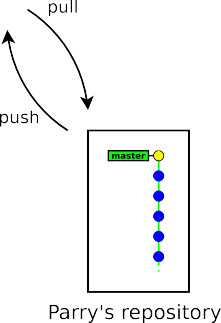
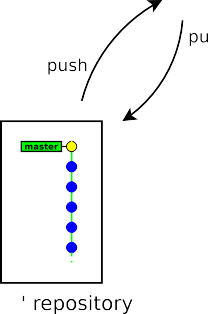
Interacting with a remote repository

* + Overview
  + Creating a shared repository
  + Configuring a remote repository
  + Sending changes (push)
  + Receiving changes (pull)

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



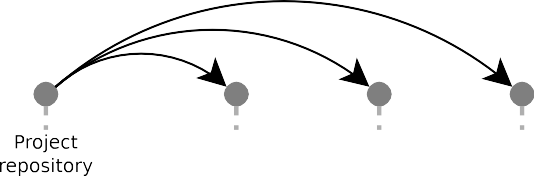
## Simple workflow (Centralised)



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Team Workflow



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Team Workflow

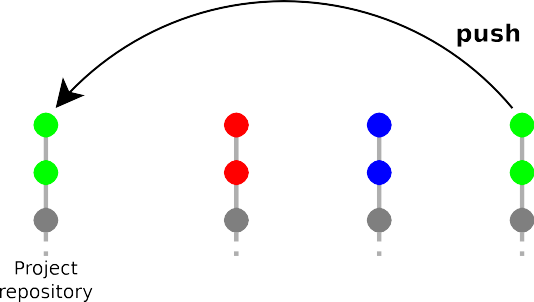


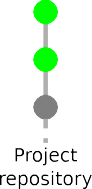


[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Team Workflow



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



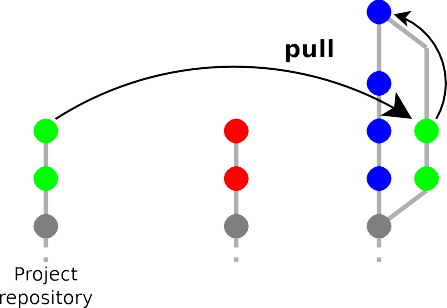
## Team Workflow



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



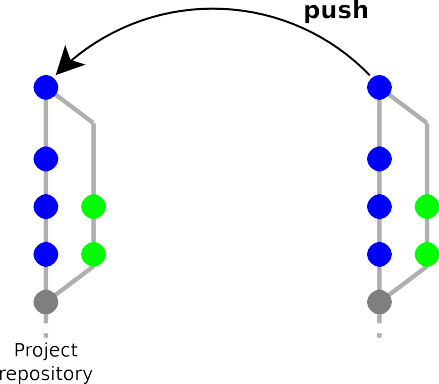
## Team Workflow



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



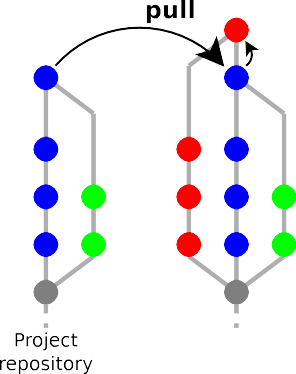
## Team Workflow



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



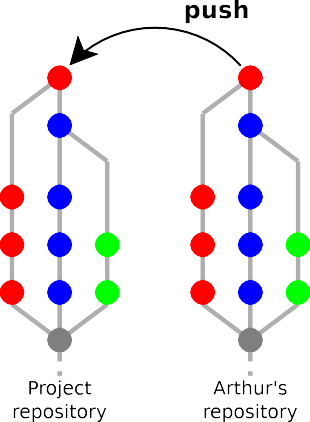
## Team Workflow



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Team Workflow

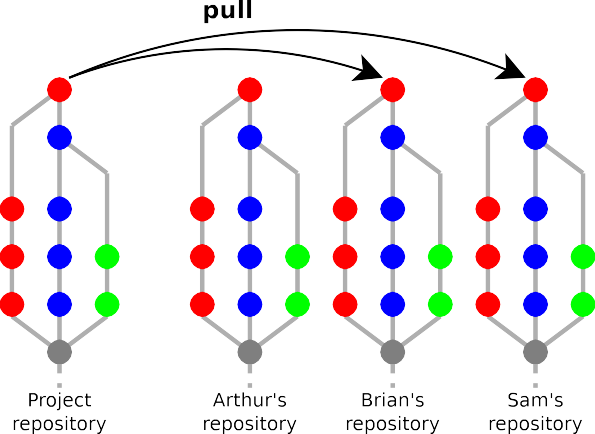




[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Team Workflow



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## How git handles remote repositories

* It is possible to work with multiple remote repositories
* Each remote repository is identified with a local alias. When working with a unique remote repository, it is usually named **origin**12
* Remote repositories are mirrored within the local repository
* Remote branches are mapped in a separate namespace:

remote/*name*/*branch*. Examples:

* + master refers to the local master branch
  + remote/origin/master refers to the master branch of the remote repository named origin

12default name used by git clone

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Adding a remote repository

git remote add *name url*

* *name* is a local alias identifying the remote repository
* *url* is the location of the remote repository Examples:

*, ,*

**$ git remote add origin /tmp/helloworld.git**

**$ git remote add origin ssh://username@scm.gforge.inria.fr/gitroot/helloworld/helloworld.git**

*z r*

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Pushing (uploading) local changes to the remote repository

git push

* git push examines the current branch, then:
  + if the branch is tracking an upstream branch, then the local changes (commits) are propagated to the remote branch
  + if not, then nothing is sent

*(new branches created locally are considered private by default)*

* In case of conflict git push will fail and require to run git pull first

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Pushing a new branch to the remote repository

git push -u *destination repository ref* [*ref . . .* ]

* explicit variant of git push: the local reference *ref* (a branch or a tag) is pushed to the remote *destination repository*
* -u/--set-upstream configures the local branch to track the remote branch13 (this is usually what you want)

*, ,*

### $ git push

fatal: The current branch master has no upstream branch.

To push the current branch and set the remote as upstream, use git push --set-upstream origin master

### $ git push -u origin master

To /tmp/helloworld.git/

\* [new branch] master -> master

Branch master set up to track remote branch master from origin.

*z r*

13so that git pull an git push work with that repository by default

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Fetching (downloading) changes from the remote repository

git fetch updates the local mirror of the remote repository:

git fetch

* it downloads the new commits from the remote repository
* it updates the references remote/*remote name*/*\** to match their counterpart in the remote repository.

Example: the branch remote/origin/master in the local repository is updated to match the new position of the branch master in the remote reposity

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Merging remote changes into the current local branch

Changes in the remote repository can be merged explicitely into the local branch by running git merge

*, ,*

### $ git status

# On branch master

### $ git fetch

...

**$ git merge origin/master**

*z r*

In practice, it is more convenient to use git pull, which is a shortcut for git fetch + git merge

*, ,*

git pull

**$ git pull**

*z r*

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



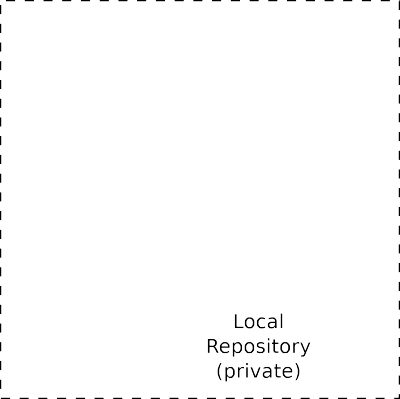
## Remote example



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Remote example

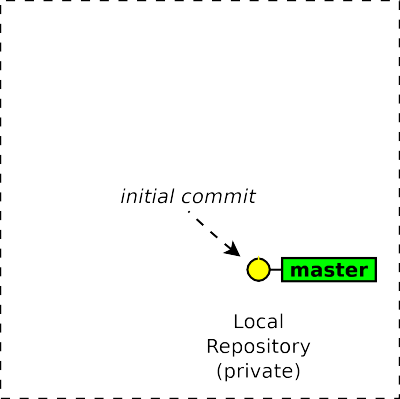




[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Remote example





[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Remote example

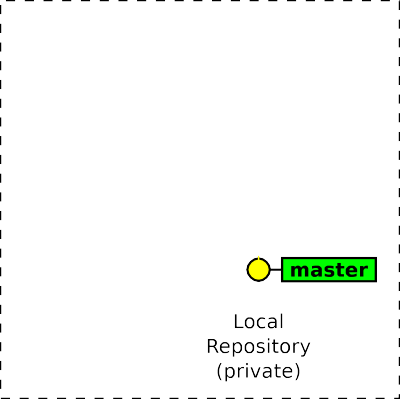




[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Remote example

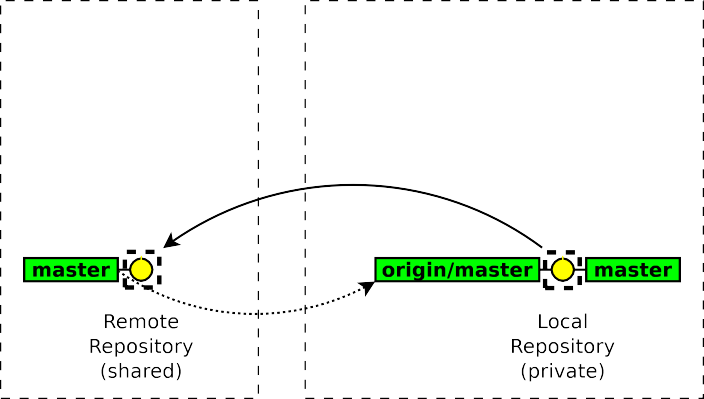




[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Remote example

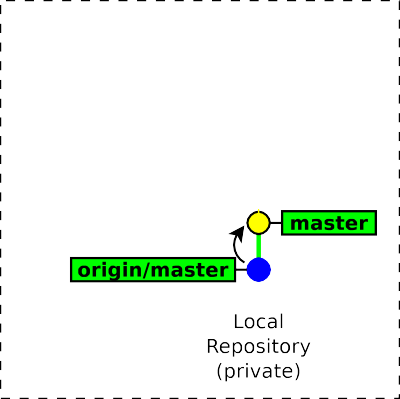
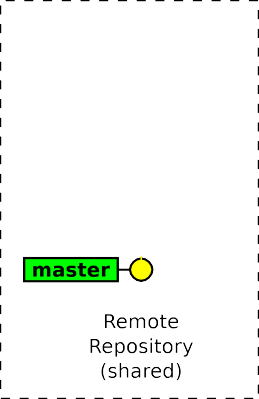




[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Remote example

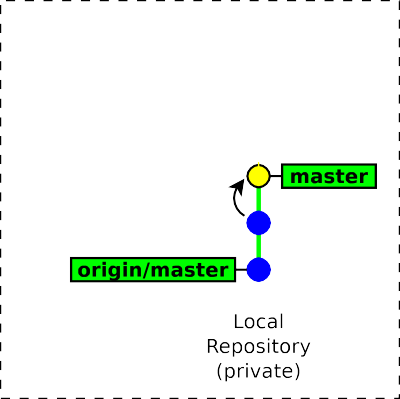
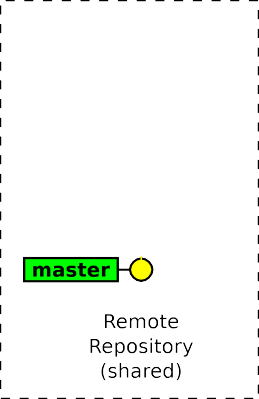




[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Remote example

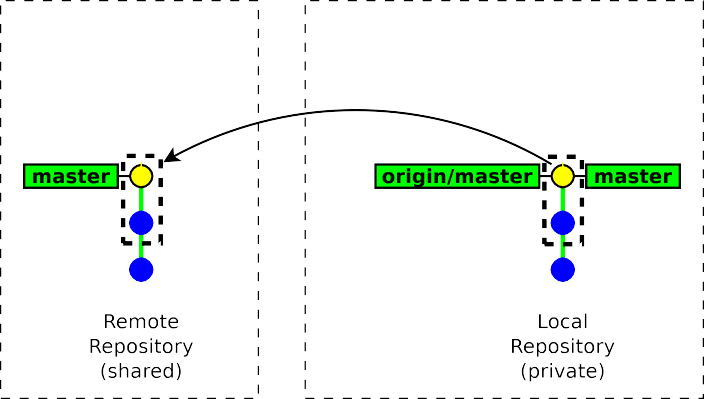




[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



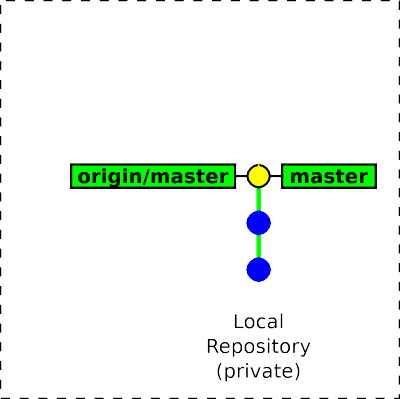
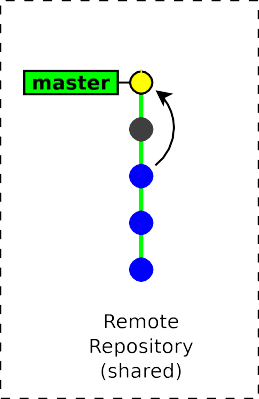
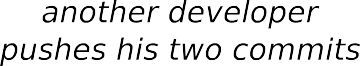
## Remote example



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



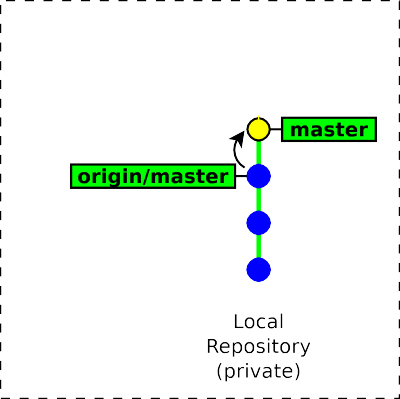
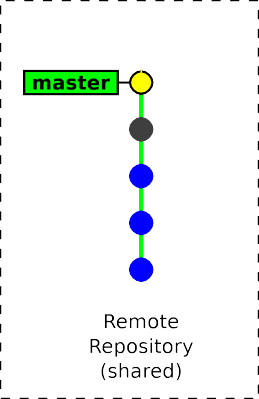
## Remote example



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Remote example

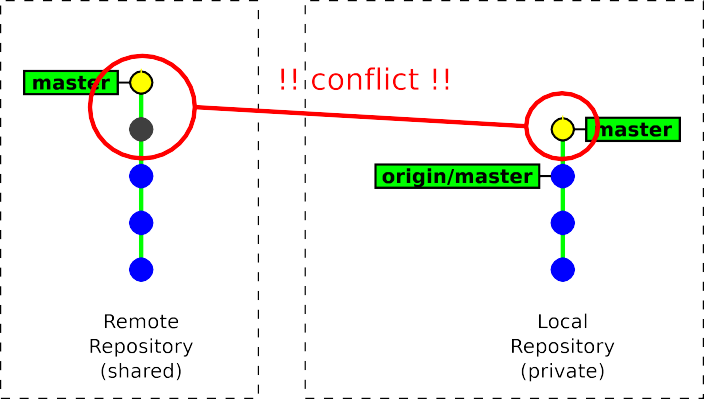




[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



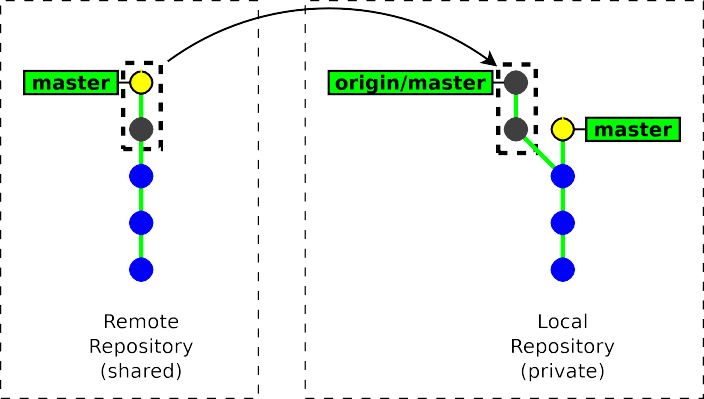
## Remote example



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Remote example

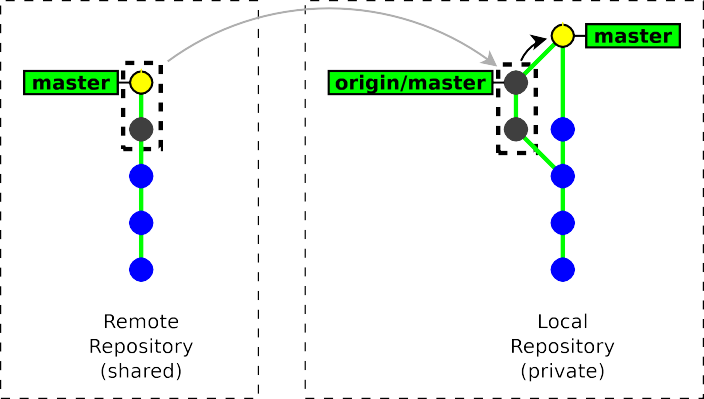




[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Remote example

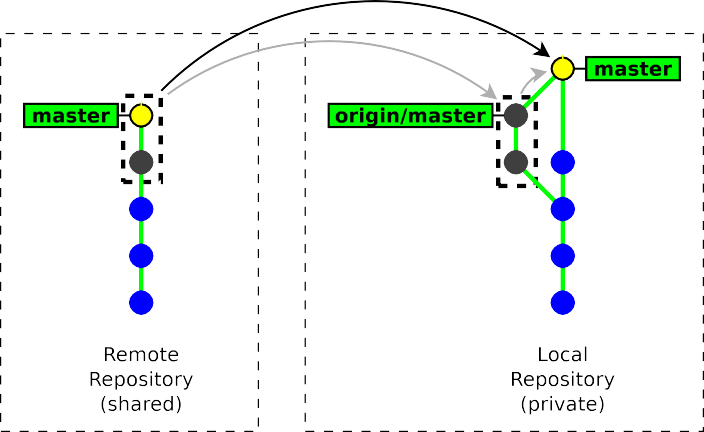




[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



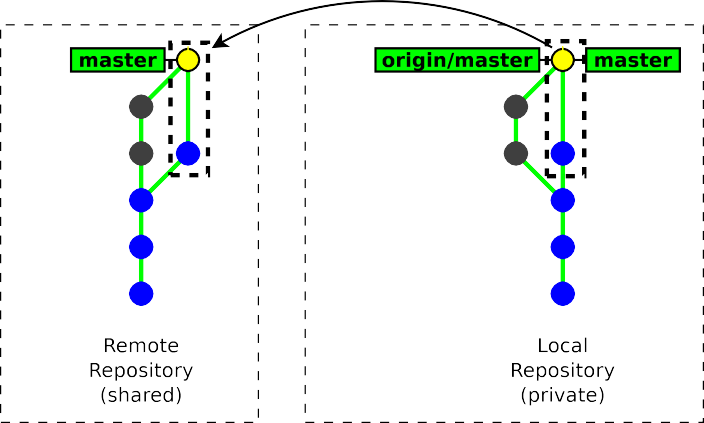
## Remote example



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Remote example



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Importing a new remote branch

If the *branch name* does not exist locally, then GIT looks for it in the remote repositories. If it finds it, then it creates the local branch and configures it to track the remote branch.

git checkout *branch name*

*, ,*

### $ git branch --all

\* master remotes/origin/master

remotes/origin/new-fancy-feature

### $ git checkout new-fancy-feature

Branch new-fancy-feature set up to track remote branch new-fancy-feature from origin. Switched to a new branch 'new-fancy-feature'

### $ git branch --all

master

\* new-fancy-feature remotes/origin/master

remotes/origin/new-fancy-feature

*z r*

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Cloning a repository

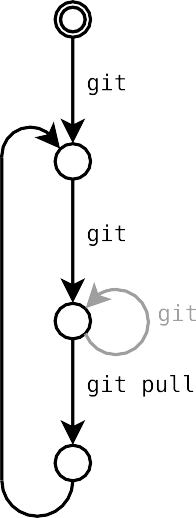
git clone *url* [ *directory* ]

* git clone makes a local copy of a remote repository and configures it as its origin remote repository.
* git clone is a shortcut for the following sequence:
  1. git init *directory*
  2. cd *directory*
  3. git remote add origin *url*
  4. git fetch
  5. git checkout master
* In practice you will rarely use git init, git remote and git fetch directly, but rather use higher-level commands: git clone and git pull.

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Typical Workflow



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Exercises

1. *(remember to visualise your operations with “*gitk --all*” → hit F5)*
2. clone the following repository https://allgo.inria.fr/git/hello
3. use gitk --all (to display remote branches too)
4. make some commits and synchronise (pull/push) with the origin repository
5. do it again so as to experience and resolve a conflict
6. use git fetch to review remote commits **before** merging them
7. create a new branch, make a commit and publish it to the shared repository
8. check out a branch created by another participant

[Ve](#_bookmark0)[rsion Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



# Part 6.

Administrating a server

* + Shared repositories
  + GIT servers
  + Available protocols

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Creating a shared repository

git init --bare --shared *my-shared-repository*.git

* + A bare repository (--bare) is a repository without any working copy.
    - by convention bare repositories use the .git extension
    - bare repository are updated by importing changes from another repository (push operation)
  + --shared is meant to make this repository group-writable (unix group)

*, ,*

### $ git init --bare --shared helloworld.git

Initialized empty shared Git repository in /tmp/helloworld.git/

### $ ls helloworld.git/

branches config description HEAD hooks info objects refs

*z r*

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Admin Considerations

Administrating a GIT server is relatively simple14

* no partial access

*(access is granted to the full repository)*

* no access policies in GIT itself

*(access control to be handled by the HTTP/SSH server)*

* low server load

*(most git commands are local)*

* server outages are much less disruptive

*(user can collaborate by other means)*

* only core developers need write access

14compared to centralised Version Control systems

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## How to publish a GIT repository (1/2)

* Native protocol (git daemon) on tcp port 9418
  + public access only, no authentication

*→* git://server.name.org/path/to/the/repository.git

* GIT over SSH
  + strong authentication & encryption
  + restricted shell possible with git-shell

*→* ssh://username@server.name.org/path/to/the/repository.git

* Local access

*→* /path/to/the/repository.git

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## How to publish a GIT repository (2/2)

* HTTP/HTTPS server
  + firewall friendly
  + many authentication methods (provided by the HTTP server)
  + can provide SSL encryption, even for anonymous users

*→* <http://username@server.name.org/path/to/the/repository.git>

* + *Dumb server* (repository published as static files)
    - very easy to set up (in read-only mode)
    - less efficient
    - read-write mode requires webdav
  + *Smart server* (git http-backend)
    - cgi script running the native daemon over HTTP
    - backward-compatible with the dumb client

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## GIT-centric forges

* Hosting only
  + GitHub

https://github.com/

* + BitBucket

https://bitbucket.com/

* + Google Code

https://code.google.com/

* Open source software
  + Gitlab

[http://gitlab.org](http://gitlab.org/)

* + Gitorious

[http://gitorious.org](http://gitorious.org/)

[Ve](#_bookmark0)[rsion Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



# Part 7.

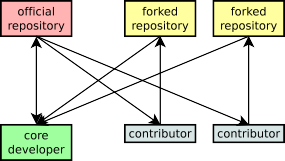
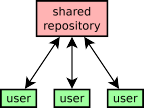
Working with third-party contributors

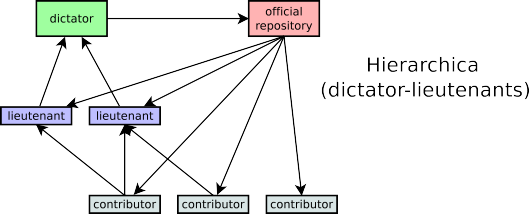
* Common workflows
* Generating & applying patches
* Merging from third-party repositories

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Common workflows





more about workflows at: [https://www.atlassian.com/git/workflows](http://www.atlassian.com/git/workflows)

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## About 3rd party contributions

Third-party contributors15 can submit their contributions by:

* sending patches (the traditional way)
* publishing their own (unofficial) repository and asking an official developer to merge from this repository

(*pull request* or *merge request*)

15developers who are not allowed to push to the official repository

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Explicit pull/push

push/pull can work on any arbitrary repository identified by its url

git push *url ref* [*ref*...]

git push *url local ref*:*remote ref* ... *(push as a different name)*

*→* push the local *ref* (a branch or a tag) to repository *url*

git pull *url ref* [*ref*...]

*→* merge the remote *ref* (a branch or a tag) from repository *url*

into the current local branch

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Decentralised workflow

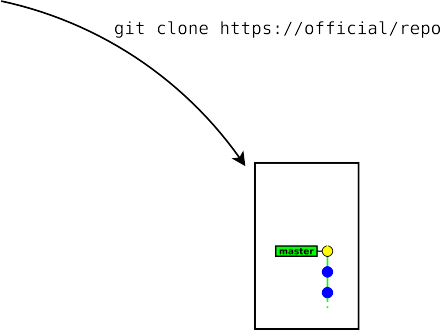




[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Decentralised workflow



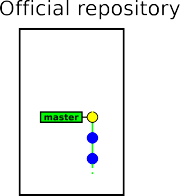


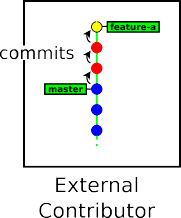
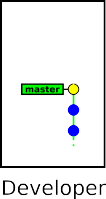


[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Decentralised workflow

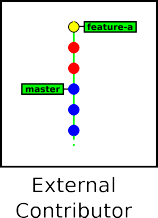
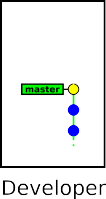
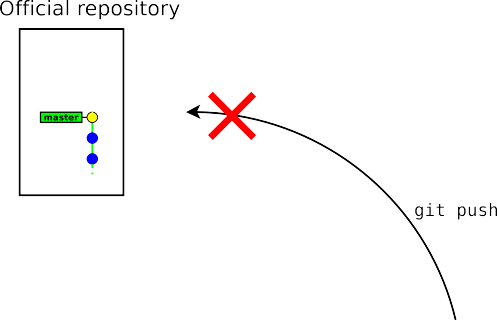




[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



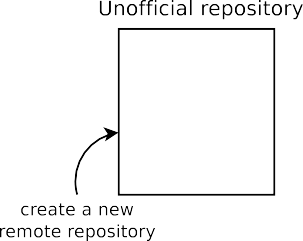
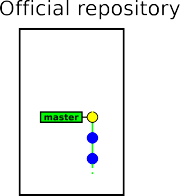
## Decentralised workflow

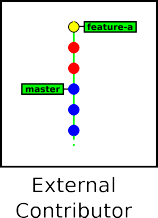
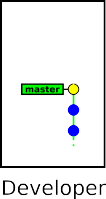


[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Decentralised workflow

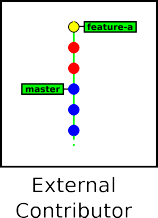
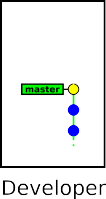
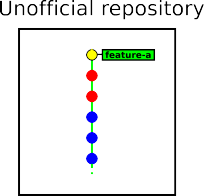
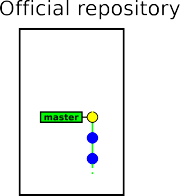




[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



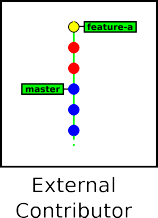
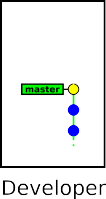
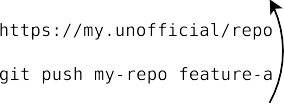
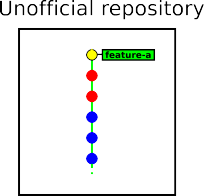
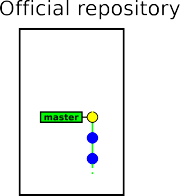
## Decentralised workflow



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Decentralised workflow

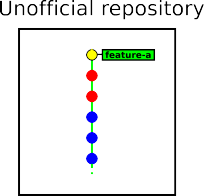
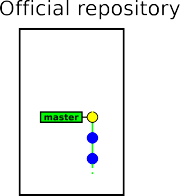


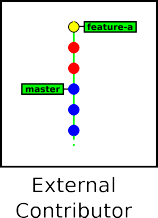
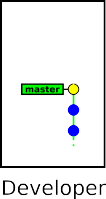


[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Decentralised workflow



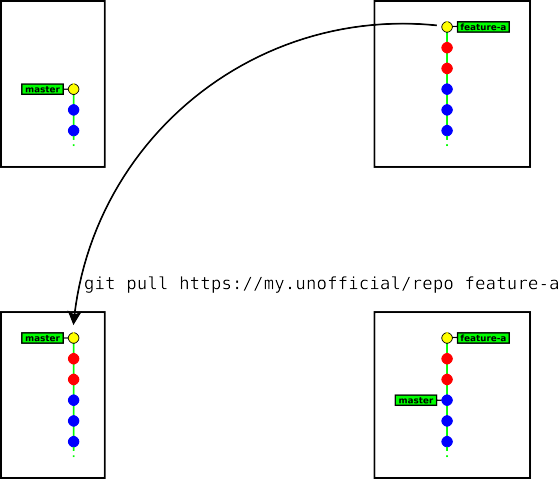


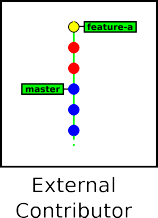


[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



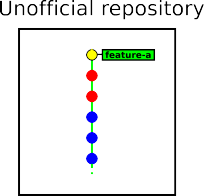
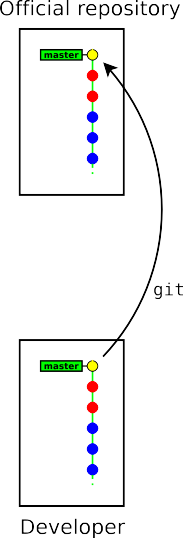
## Decentralised workflow



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



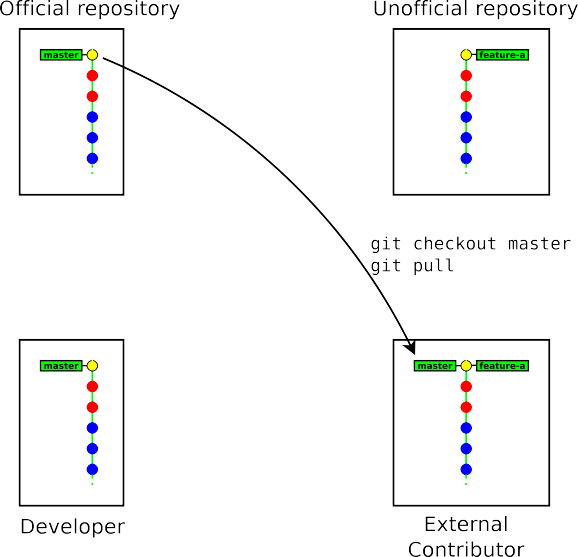
## Decentralised workflow



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



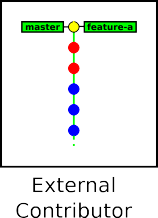
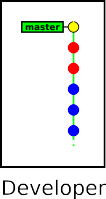
## Decentralised workflow

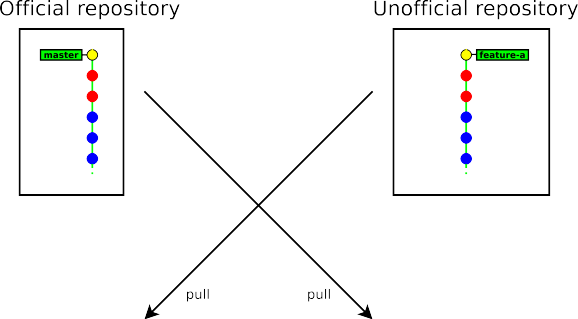


[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Decentralised workflow





[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Reviewing a remote branch

git pull merges **immediately** the remote branch into the current local branch.

In practice you may prefer to review it before merging.

*→* fetch the branch *branch* from the repository *url* and store it temporarily16 as FETCH HEAD

git fetch *url branch*

*, ,*

### $ git fetch git://git.raoul-duke.org/helloworld.git master

From git://git.raoul-duke.org/helloworld.git

\* branch master -> FETCH\_HEAD

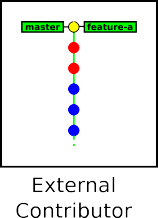
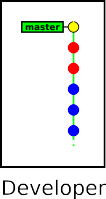
### $ gitk FETCH\_HEAD

...review the commits ...

**$ git merge FETCH\_HEAD**

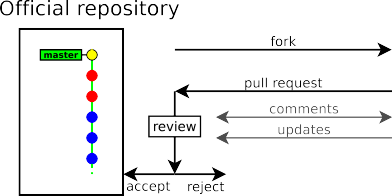
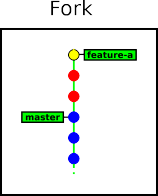
*z r*

16the FETCH HEAD ref remains valid until the next time git fetch is run

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Decentralised workflow (GIT-centric forges)



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Generating patches

* git diff

The basic (legacy) way: use git diff

* git format-patch

The modern way: git format-patch converts you history (commits) into a series of patches (on file per commit) and it **records the metadata (author name, commit message)** 17

17**Note:** git format-patch does not preserve merge history & conflicts resolution. You should only use it when your history is linear.

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)

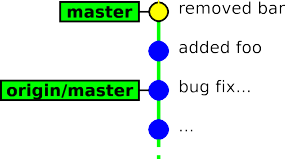


## Generating patches

git format-patch generates patches from revision *rev origin* to

git format-patch *rev origin*[..*rev final* ]

*rev final* (or to the current version if not given)

Example:

*, ,*

### $ git format-patch origin/master

0001-added-foo.patch 0002-removed-bar.patch

*z r*

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Applying patches

git am *file1* [ *file2* ...]

* git am18 applies a series of patches generated by git format-patch into the local repository (each patch produces one commit)
* the authorship of the submitter is preserved19

*, ,*

### $ git am 0001-added-foo.patch 0002-removed-bar.patch

Applying: added foo Applying: removed bar

*z r*

18am originally stands for “apply mailbox”

19actually GIT distinguishes between the **author** and the **committer** of a revision (usually they refer to the same person, but not when running git am)

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Exercises

In this part two separate repositories will be used :

* https://allgo.inria.fr/git/userDD *→* developer (official repository)
* https://allgo.inria.fr/git/userCC *→* external contributor

1. *(reminder: use* gitk --all*)*
2. associate with your neighbour and distribute roles: one is the **developer** and one is the **external contributor**
3. **(developer)** clone your repository (https://allgo.inria.fr/git/userDD) on your local machine, make some commits and push them
4. **(contributor)** clone the developer’s repository (https://allgo.inria.fr/git/userDD) and make some commits (but do not push them, you are not supposed to have the rights)
5. **(contributor)** push your new commits to your own public repository at

https://allgo.inria.fr/git/userCC

1. **(developer)** pull the commits from the contributor’s repository and push them to your own repository
2. **(contributor)** pull from the official repository and check that your commits were merged in the upstream branch
3. **(contributor)** make new commits (but do not push them)
4. **(contributor)** convert your new commits into patches and send them to the developer
5. **(developer)** apply the contributor’s patches and push them
6. **(contributor)** pull the latest changes and check that your patches were effectively applied upstream

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



# Part 8. Extras

* Some advices
* Common traps
* Documentation
* Next tutorial

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Some advices (1/2)

* commit as often as you can (keep independent changes in separate commits)
* run git diff before preparing a commit
* in commit messages, describe the rationale behind of your changes (it is often more important than the change itself)
* do not forget to run git push
* use a .gitignore file to ignore generated files (\*.o, \*.a, . . . )

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Some advices (2/2)

* don’t be fully desynchronised *→* run git pull enough often to avoid accumulating conflicts
* idem for feature branches

(merge from the mainstream branch enough often)

* when creating complex patches (as an external contributor) prefer using one branch per patch
* keep a gitk instance open when doing fancy things

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Common traps (1/2)

* git diff without arguments shows the difference with the index *→* run git diff HEAD to show the differences with the last commit
* git reset reverts the index, but keeps the working copy unchanged

*→* do git reset --hard if you need to revert the working copy too

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Common traps (2/2)

* GIT is not forgiving, do not ignore its warnings and do not use

--force unless you have a clear idea of what you are doing

* GIT’s history is not immutable
* git checkout on an arbitrary commit or a tag (anything that is not a branch) puts your in “detached HEAD” state. You can commit, but your history be lost if you don’t create any branch (or tag) to reference them.

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



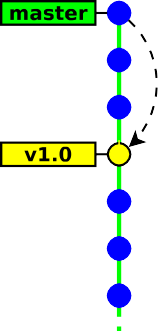
## Detached head state



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Detached head state





[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)

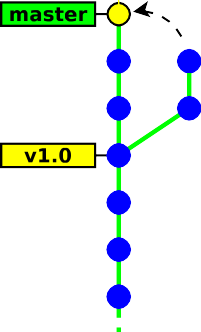


## Detached head state

[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Detached head state

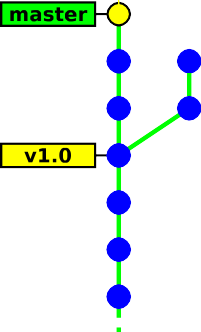




[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Detached head state



[Version Control](#_bookmark0) [GIT Intro](#_bookmark18) [Local GIT](#_bookmark27) [Branches](#_bookmark44) [Remote GIT](#_bookmark61) [Server](#_bookmark78) [Bazar](#_bookmark85) [Extras](#_bookmark98)



## Other useful utility commands

* git gc *→* garbage collector (run it when the /.git/

directory takes too much space)

* git stash *→* save/restore the state of the working copy and index (useful when in need to commit an urgent fix)
* git clean *→* clean the working tree (you must ensure that all your code is committed)
* git bisect *→* locating which commit introduced a bug
* git cherry-pick *→* merging a single commit
* git revert *→* cancelling a previous commit

THANK YOU