Software Engineering at MPRI - Tutorial on the version control system git, and its extensions

Amélie Ledein
ledein@lsv.fr

October 9, 2020
Don't forget ...
Don’t forget …

- **Rigor**
  - Code, methodology, specifications and tests
Don’t forget ... 

- **Rigor**
  - Code, methodology, specifications and tests

- **Adaptability**
  - Components could be re-used in a (slightly different) context.
Introduction

*Don’t forget ...*

- **Rigor**
  - Code, methodology, specifications and tests

- **Adaptability**
  - Components could be re-used in a (slightly different) context.

- **Modularity**
  - Segment project in *modules* with clearly defined *interfaces*.
  - Develop and test independently, facilitate re-use and change.
Don’t forget ...

- **Rigor**
  - Code, methodology, specifications and tests

- **Adaptability**
  - Components could be re-used in a (slightly different) context.

- **Modularity**
  - Segment project in **modules** with clearly defined **interfaces**.
  - Develop and test independently, facilitate re-use and change.

- **Abstraction**
  - Do not specify implementation **details**, i.e. things that can easily change.
Introduction

Don’t forget ...

- **Rigor**
  → Code, methodology, specifications and tests

- **Adaptability**
  → Components could be re-used in a (slightly different) context.

- **Modularity**
  → Segment project in *modules* with clearly defined *interfaces*.
  → Develop and test independently, facilitate re-use and change.

- **Abstraction**
  → Do not specify implementation *details*, i.e. things that can easily change.

OK, but how can you share your code?
Introduction

*Don’t forget …*

- **Rigor**
  - Code, methodology, specifications and tests

- **Adaptability**
  - Components could be re-used in a (slightly different) context.

- **Modularity**
  - Segment project in modules with clearly defined interfaces.
  - Develop and test independently, facilitate re-use and change.

- **Abstraction**
  - Do not specify implementation details, i.e. things that can easily change.

*OK, but how can you share your code?*

- USB-key
Don’t forget ...

- **Rigor**
  - Code, methodology, specifications and tests

- **Adaptability**
  - Components could be re-used in a (slightly different) context.

- **Modularity**
  - Segment project in modules with clearly defined interfaces.
  - Develop and test independently, facilitate re-use and change.

- **Abstraction**
  - Do not specify implementation details, i.e. things that can easily change.

OK, but how can you share your code?

- **USB-key**
Introduction

Don’t forget …

- **Rigor**
  - Code, methodology, specifications and tests

- **Adaptability**
  - Components could be re-used in a (slightly different) context.

- **Modularity**
  - Segment project in modules with clearly defined interfaces.
  - Develop and test independently, facilitate re-use and change.

- **Abstraction**
  - Do not specify implementation details, i.e. things that can easily change.

OK, but how can you share your code?

- USB-key
- Email
Introduction

_Don’t forget …_

- **Rigor**
  - Code, methodology, specifications and tests

- **Adaptability**
  - Components could be re-used in a (slightly different) context.

- **Modularity**
  - Segment project in _modules_ with clearly defined _interfaces_.
  - Develop and test independently, facilitate re-use and change.

- **Abstraction**
  - Do not specify implementation _details_, i.e. things that can easily change.

_OK, but how can you share your code?_

- **USB-key**
- **Email**
Introduction

Don’t forget …

- **Rigor**
  - Code, methodology, specifications and tests

- **Adaptability**
  - Components could be re-used in a (slightly different) context.

- **Modularity**
  - Segment project in *modules* with clearly defined *interfaces*.
  - Develop and test independently, facilitate re-use and change.

- **Abstraction**
  - Do not specify implementation *details*, i.e. things that can easily change.

*OK, but how can you share your code?*

- USB-key
- Email
- Dropbox, Google Drive, etc.
Introduction

Don’t forget …

- **Rigor**
  - Code, methodology, specifications and tests

- **Adaptability**
  - Components could be re-used in a (slightly different) context.

- **Modularity**
  - Segment project in *modules* with clearly defined *interfaces*.
  - Develop and test independently, facilitate re-use and change.

- **Abstraction**
  - Do not specify implementation *details*, i.e. things that can easily change.

**OK, but how can you share your code?**

- USB-key
- Email
- Dropbox, Google Drive, etc.
The answer is …
or any version control systems (VCS).
(systèmes de gestion de version, in French)
1 Git
- Git in a nutshell
- Basic commands
- Branching
- Data structures

2 GitHub
1 Git
   - Git in a nutshell
   - Basic commands
   - Branching
   - Data structures

2 GitHub
What is a version control system?

Software system that allows you to maintain and manage all versions of a set of files.

Why a version management system?
- Revolve easily to a previous version.
- Follow the evolution of the project over time.
- Allow parallel work on disjointed parts of the project and manage the competing modifications.
- Facilitate the detection and correction of errors.

An example:→ A free and open source distributed version control system (DVCS)→ Designed to handle everything from small to very large projects
What is a version control system?

- Software system that allows you to maintain and manage all versions of a set of files.

Why a version management system?
- Revolve easily to a previous version.
- Follow the evolution of the project over time.
- Allow parallel work on disjointed parts of the project and manage the competing modifications.
- Facilitate the detection and correction of errors.
- etc.

An example:
→ A free and open source distributed version control system (DVCS)
→ Designed to handle everything from small to very large projects with speed and efficiency.
What is a version control system?

- Software system that allows you to maintain and manage all versions of a set of files.

Why a version management system?

- Revolve easily to a previous version.
- Follow the evolution of the project over time.
- Allow parallel work on disjointed parts of the project and manage the competing modifications.
- Facilitate the detection and correction of errors.
- etc.
What is a version control system?

- Software system that allows you to maintain and manage all versions of a set of files.

Why a version management system?
- Revolve easily to a previous version.
- Follow the evolution of the project over time.
- Allow parallel work on disjointed parts of the project and manage the competing modifications.
- Facilitate the detection and correction of errors.
- etc.

An example:

→ A free and open source distributed version control system (DVCS)
→ Designed to handle everything from small to very large projects with speed and efficiency
Git in detail

Stash (Remise): A place to hide modifications while you work on something else.

Workspace (Espace de travail): Local checkout.

Index (Zone d’index): staging area, staged files or (current directory) cache: Files you want to commit. Before you “commit” (checkin) files, you need to first add them to the index.

Local repository (Dépôt local): A subdirectory named .git that contains all of your necessary repository files - a Git repository skeleton.

Remote/upstream repository (Dépôt distant): Versions of your project that are hosted on the Internet or network, ensuring all your changes are available for other developers.

The default name is origin.
● **Stash (remise):** A place to hide modifications while you work on something else.
- **Stash (remise):** A place to hide modifications while you work on something else.
- **Workspace (espace de travail):** Local checkout.
- **Stash (remise):** A place to hide modifications while you work on something else.
- **Workspace (espace de travail):** Local checkout.
- **Index (zone d’index), staging area, staged files or (current directory) cache:** Files you want to commit. Before you ”commit” (checkin) files, you need to first add them to the index.
**Git in detail**

- **Stash** *(remise)*: A place to hide modifications while you work on something else.
- **Workspace** *(espace de travail)*: Local checkout.
- **Index** *(zone d’index)*, **staging area**, **staged files** or (current directory) cache: Files you want to commit. Before you ”commit” (checkin) files, you need to first add them to the index.
- **Local repository** *(dépôt local)*: A subdirectory named .git that contains all of your necessary repository files - a Git repository skeleton.
Git in detail

- **Stash (remise)**: A place to hide modifications while you work on something else.
- **Workspace (espace de travail)**: Local checkout.
- **Index (zone d’index), staging area, staged files** or (current directory) cache: Files you want to commit. Before you ”commit” (checkin) files, you need to first add them to the index.
- **Local repository (dépôt local)**: A subdirectory named .git that contains all of your necessary repository files - a Git repository skeleton.
- **Remote/upstream repository (dépôt distant)**: Versions of your project that are hosted on the Internet or network, ensuring all your changes are available for other developers. The default name is origin.
Sommaire

1 Git
   - Git in a nutshell
   - Basic commands
   - Branching
   - Data structures

2 GitHub
Synchronization with the remote repository

$ git clone <url>
Retrieve an entire repository from hosted location via URL.

$ git fetch <alias>
Fetch down all the branches from that Git remote.

$ git merge <alias/<branch>
Merge a remote branch into your current branch to bring it up to date.

$ git pull
Fetch and merge any commits from the tracking remote branch.
Commit

98ca9
- commit size
- tree 92ec2
- parent
- author Scott
- committer Scott
  The initial commit of my project

34ac2
- commit size
- tree 184ca
- parent 98ca9
- author Scott
- committer Scott
  Fixed bug #1328 - stack overflow under certain conditions

f30ab
- commit size
- tree 0de24
- parent 34ac2
- author Scott
- committer Scott
  Add feature #32 - ability to add new formats to the central interface

Snapshot A
Snapshot B
Snapshot C
Make modifications

$ git add <file(s)>
Add a file (or several) as it looks now to your next commit (stage).

$ git rm <file(s)>
Delete the file (or several) from the project and stage the removal for commit.

$ git mv <old-name-file> <new-name-file>
Rename the file and stage the renaming.

$ git mv <existing-path> <new-path>
Change an existing file path and stage the move.

$ git reset <file(s)>
Unstage a file (or several) while retaining the changes in working directory.

$ git commit [-m "<descriptive message>"]
Commit your staged content as a new commit snapshot.

$ git push <alias> <branch>
Transmit local branch commits to the remote repository branch.
Exercise 1

1. Clone the repository from
   https://github.com/amelieled/SE_GIT_MPRI.git
2. Add at least 5 new items in the grocery list.
3. Fix the 5 errors.
4. Add a new section.
Informative commands

- **Setup:**
  Configuring user information used across all local repositories.
  
  ```
  $ git config --global user.name "[firstname lastname]"
  Set a name that is identifiable for credit when review version history.
  
  $ git config --global user.email "[valid-email]"
  Set a email address that will be associated with each history marker.
  ```

  **Note:** `export EDITOR=emacs` (or `vim`, etc.)
  To configure correctly your editor with Git.

- **To collect information:**
  
  ```
  $ git status
  Show modified files in working directory, staged for your next commit.
  
  $ git diff
  Diff of what is changed but not staged.
  
  $ git diff --staged
  Diff of what is staged but not yet committed.
  ```
Sommaire

1 Git
   - Git in a nutshell
   - Basic commands
   - Branching
   - Data structures

2 GitHub
## Branch

$ git branch
List your branches.
A star (*) will appear next to the currently active branch.

$ git branch <branch-name>
Create a new branch at the current commit.

$ git branch -d <branch-name>
Delete the specified branch.

$ git checkout <branch-name>
Switch to another branch and check it out into your working directory.

$ git merge <branch-name>
Merge the specified branch’s history into the current one.

$ git log
Show all commits in the current branch’s history.
Gitk - Graphical interface

Or if you prefer: git log --graph.
$ git log branchB..branchA
Show the commits on branchA that are not on branchB.

$ git log --follow <file>
Show the commits that changed file, even across renames.

$ git diff branchB...branchA
Show the diff of what is in branchA that is not in branchB.

$ git log --stat -M
Show all commit logs with indication of any paths that moved.

$ git show <SHA>
Show any object in Git in human-readable format.

→ Easier on Github (See later)
1 Git
- Git in a nutshell
- Basic commands
- Branching
- Data structures

2 GitHub
SHA1 is a hashing algorithm taking an input up to $2^{64}$ bits, and returns a unique sequence of 40 hexadecimal characters.

By hashing the contents of a file, Git obtains a series of unique digits symbolizing the file. Then, Git backs up only the files which are different hash (Git does not care about the names of the files, it only considers the content.).
There are four Git objects:

- **The Blob** (*Binary Large Object*): It more commonly represents a file.
- **The Tree**: It more commonly represents a directory or folder of your application. Its content is the list of SHA1s of Blobs or other Trees that it can contain. What gives a tree structure of files.
- **The Commit**: This is the complete state of your project at a given moment, i.e. a snapshot. Its content is the SHA1 of the source Tree, and various information such as the name of the commit, the name of the author, the date, etc.
- **The Tag**: This is an object used to qualify a particular commit by giving it a comment.
Each element has a unique SHA1.
How Git stores its information?

Only thanks to the directory `.git` at the root of your project.

- **config**: file relating to the configuration of the Git environment, such as information about the developer (name, email, etc.);
- **description**: contains information about your project;
- **objects/**: it is in this directory that all Git objects are stored (commits, tags, trees, blobs);
- **refs/**: contains information on local branches of the repository;
- **logs/**: contains log messages;
- **index**: file containing information about the status of the next commit.
- **HEAD**: pointer to current branch;
- **hooks/**: folder containing "hooks" or "triggers", i.e. actions/scripts that can be executed in pre or post condition.
Some questions

- How find a particular object?

Thanks to SHA1, in particular: .git/objects/

Example:
- How find the blob "4A558..."

See on /.git/objects/4A/558...

This structure is too heavy?

The answer is no thanks to the "Zlib" compression system that Git uses to compress the data and maintain the correct weight.

Demonstration:
- $ sudo apt install qpdf
- $ zlib-flate -uncompress < FILE

Amélie Ledein
Tutorial
October 9, 2020 24 / 31
Some questions

- How find a particular object?

Thanks to SHA1, in particular: .git/objects/

Example:

How find the blob "4A558..."?

See on /.git/objects/4A/558...

This structure is too heavy?

The answer is no thanks to the “Zlib” compression system that Git uses to compress the data and maintain the correct weight.

Demonstration:

$ sudo apt install qpdf
$ zlib-flate -uncompress < FILE
Some questions

- **How find a particular object?**
  Thanks to SHA1, in particular: .git/objects/
Some questions

- **How find a particular object?**
  Thanks to SHA1, in particular: `.git/objects/
  Example: **How find the blob ”4A558...”?**
How find a particular object?
Thanks to SHA1, in particular: .git/objects/
Example: **How find the blob ”4A558...”?**
See on / .git/objects/4A/558...
Some questions

- **How find a particular object?**
  Thanks to SHA1, in particular: `.git/objects/
  Example: **How find the blob ”4A558...”?**
  See on `.git/objects/4A/558...

- **This structure is too heavy?**
Some questions

- **How find a particular object?**
  Thanks to SHA1, in particular: `.git/objects/`
  Example: **How find the blob ”4A558...”?**
  See on `.git/objects/4A/558...`

- **This structure is too heavy?**
  The answer is no thanks to the ”Zlib” compression system that Git
  uses to compress the data and maintain the correct weight.
Some questions

- **How find a particular object?**
  Thanks to SHA1, in particular: .git/objects/
  Example: **How find the blob ”4A558...”?**
  See on /.git/objects/4A/558...

- **This structure is too heavy?**
  The answer is no thanks to the ”Zlib” compression system that Git uses to compress the data and maintain the correct weight.
  Demonstration:
  $ sudo apt install qpdf
  $ zlib-flate -uncompress < FILE
Graphical interface

- See current code
- See each commit
- See each issue
- Do integration continuous
- etc.
Continuous integration - Testing

Travis CI
and it can get complicated to debug (Why3 users had issues with this when testing my server)

Does it really? It seemed to me that some language server can be used with any LSP client (I'm not clear on the architecture of LSP, so I might confuse things up)

If we develop extensions, yep ;)

Gabriel Hondet @gabrielhd
Ah right right

Emilio Jesús Gallego Arias @ejgallego
I think that for now the only extension is this extra goals request
so it is not like super-complex

DIVERIO Diego @vycastor
When developing an emacs mode, you also have to provide some configuration to the LSP client you’re using underneath, but it’s pretty basic

Emilio Jesús Gallego Arias @ejgallego
yup, that was one of the advantages of eglot, it kinda worked "config free"

DIVERIO Diego @vycastor
So to support two different clients, we’ll have to provide two emacs configuration, even if short

Click here to type a chat message. Supports GitHub flavoured markdown.
References

- Interactive tutorial:
  - learngitbranching.js.org

- Cheat sheet:
  - https://ndpsoftware.com/git-cheatsheet.html (Interactive one - English, French, Chinese, Spanish, German, Korean)