Rebase is base

git basics workshop

Jiri Vlasak

Jul 2023

Introduction

Introduction

What?

- We know: that we don't know much
- We will: workshop

Table of content

I will trick you

See Pro Git or `git COMMAND --help` to avoid being tricked.

This is workshop

- I will be typing a lot, soon.
- You should be typing a lot, too.

Why (bother to) learn git?

- Craftsmanship
- Altruism

It's like . . .

- Typing all ten fingers
- Using tiling window manager
- Watching youtube videos speed 2x

. . . but you effect efficiency of others

- Writing nice and understandable code
- Clean history of changes (e.g., for code review)

Git is . . .

. . . program for tracking changes to the rows of files in a repository.

That is

- Repository is a directory.
- There are files in that directory.
- The rows of that files change.
- Git tracks those changes.
- A set of changes is called a commit.
- Commits are stacked one on the other.
- Commits form the Directed Acyclic Graph.

Git tracks *how* the current state of the repository is built.

The workshop goal (not really)

Create Python3 package called `args` ...

```
$ cd args ; tree
.
|-- args
|   |-- evaluator.py
|   `-- __init__.py
|-- LICENSE
|-- README
`-- tests
    |-- test_sum.py
    |-- test_prod.py
    |-- test_nonincr.py
    `-- test_nondecr.py

    2 directories, 8 files
```

The workshop goal (not really)

... that contains module `evaluator` with the `ev` procedure:

```
def ev(op, *args):
    """Return value computed from ``args`` based on ``op``.

    Procedure ``ev`` returns:
    - sum of arguments for ``op == "+"``,
    - product of arguments for ``op == "*"``,
    - true if arguments are non-descending for ``op == "<"``,
    - true if arguments are non-increasing for ``op == ">"``.

    :param op: A (string) operator to be applied to ``args``.
    :param args: The list (of numbers) to apply ``op`` to.
    """
    pass  # TODO
```

TODO

1. **Repository initialization**
2. Add license, readme
3. Add operator “+”
4. Add operator “*”
5. Add docstrings for “+” and “*” unit tests
6. **Alternative history**
7. Time travel
8. Branch and merge
9. **Conflicting history**
10. Branch and tests for “<”
11. Rewrite README
12. Branch and tests for “>”
13. Merge and solve conflicts
14. **Rebase way to solve conflicts**
15. Commands used

Baby steps

Baby steps

What?

- We know: what git is and what we'll do
- We will: initialize repository and add first files

Table of content

- Repository initialization
- Add license, readme
- Summary and questions

Repository initialization

Initialize repository, set name and email.

```
git init
git config user.email 'foo@bar.buzz'
git config --global user.name 'Foo Bar'
git config --global core.editor 'notepad'
```

Add license, readme

Add first files. Check the repository.

```
git status
git add LICENSE README
git diff --cached
git commit
```

Inspect repository.

```
git status
git log
git show
```

Set `git logg` command

```
git config --global --add alias.logg \
    'log --oneline --graph --decorate --all'
git logg
```

Summary and questions

Creating Python3 package

Creating Python3 package

What?

- We know: how to add files and see git history
- We will: calm down and practice a bit

Table of content

- Add operator “+”
- Summary and questions

Add operator “+”

Add Python3 package

```
git status
tree
git add args/
git commit
git show
```

Add evaluator with “+” operator.

```
cat args/evaluator.py
git add args/evaluator.py
```

Add unit tests.

```
python3 -m unittest discover tests
git show HEAD^
```

Add .gitignore.

```
git commit -m'...'
```

Summary and questions

Rewriting git history

Rewriting git history

What?

- We know: how to add files, see git history, and run tests
- We will: manipulate git history

Table of content

- Add operator “*”
- Add docstrings for “+” and “*” unit tests
- Summary and questions

Add operator “*”

Reorder `.gitignore` in git history.

```
git rebase -i
git logg
```

Rename test module.

```
git mv tests/test_sum.py tests/test_operators.py
git commit --amend
```

Add unit test for operator “*”, implement operator “*”.

```
...
python3 -m unittest discover tests
...
python3 -m unittest discover tests
```

Reorder git history (operator “+” unit tests) for TDD.

```
git rebase -i
git logg
python3 -m unittest discover tests
```

Add docstrings for “+” and “*” unit tests

Add just some lines and commit fix commits.

```
git add -p
git commit -m'F ...'
```

Rewrite/fix history.

```
git logg
git rebase -i ...
```

`git show` the changes in the history now

```
git logg
git show ...
```

Summary and questions

Alternative history

Alternative history

What?

- We know: how to create and manage git history
- We will: time travel, emphase where feature begins and ends

Table of content

Time travel

Change to different commits (revisions, changes).

```
git logg
git checkout ...
```

HEAD and change back to master branch.

```
git checkout master
```

Branch and merge

Create new branch.

```
git branch ...
```

Reset current branch. (WARNING! Changes will be lost!)

```
git reset --hard ...
```

Change between branches.

```
git checkout ...
```

Merge branch to the current one.

```
git merge --no-ff ...
git logg
```

Summary and questions

Conflicting history

Conflicting history

What?

- We know: (almost) all we need
- We will: calm down and solve conflicts

Table of content

- Branch and tests for “<”
- Rewrite README
- Branch and tests for “>”
- Do not forget to implement operator “<”
- Merge and solve conflicts
- Summary and questions

Branch and tests for “<”

Create branch for operator “<” and add tests.

```
git branch ...
git checkout ...
git add ...
git commit ...
```

Rewrite README

Change branch and update readme.

```
git checkout master
...
git add README
git commit -m'...'
```

Branch and tests for “>”

Create branch for operator “>” from older than master commit.

```
git branch ... ...
git checkout ...
```

TDD: Add tests, run tests, implement operator “>”, run tests.

```
...
```

Do not forget to implement operator “<”

Change to branch for operator “<”.

```
git logg
git checkout ...
```

Implement operator “<” and run tests.

```
...
```

Merge and solve conflicts

Change to master branch.

```
git checkout master
```

Merge branches for operators “<” and “>”.

```
git merge --no-ff ... ...
```

Solve conflicts. (Git does not know how to join alternative histories.)

```
...
git add ...
git merge --continue
git logg
git show ...
python3 -m unittest discover tests
```

Delete temporary branches.

```
git branch -d ... ...
```

Summary and questions

Rather rebase (those) conflicts

Rather rebase (those) conflicts

What?

- We know: how to resolve conflicts
- We will: learn the rebase way to do that

Table of content

Rebase way to solve conflicts

- Create branches for “<” and “>”
- Reset the master branch
- Merge “<” branch to master
- Rebase “>” branch to master (and solve conflicts)
- Merge “>” to master

You already know the commands.

```
git branch ...
git reset --hard ...
git merge ...
git checkout ...
git rebase ...
```

Summary and questions

Commands used

Commands used

What?

- We know: everything now
- We will: try to not forget

Table of content

- Repository initialization
- Adding files, repository exploration
- Editor and alias setup
- Reordering and fixing the history
- Alternative history (branch)
- Conflicting history

Repository initialization

```
git init
git config user.email 'foo@bar.buzz'
git config --global user.name 'Foo Bar'
```

Adding files, repository exploration

```
git status
git add FILENAME
git diff --cached
git commit
git config --global core.editor 'vim'
git log
git show
git log --oneline --graph --decorate --all
```

Editor and alias setup

```
git config --global --add alias.logg \
    'log --oneline --graph --decorate --all'
git logg

python3 -m unittest discover tests
git show HEAD^
git commit -m'When applied, this commit will ... (max. 52)'
```

Reordering and fixing the history

```
git rebase -i COMMIT-IDENTIFIER
git mv OLD-FILENAME NEW-FILENAME
git commit --amend

git add -p FILENAME
git commit -m'F COMMIT-IDENTIFIER'
```

Alternative history (branch)

```
git checkout COMMIT-IDENTIFIER
git branch NEW-BRANCH-NAME
git reset --hard COMMIT-IDENTIFIER
git checkout BRANCH-NAME
git merge --no-ff BRANCH-NAME
git branch -d BRANCH-NAME
```

Conflicting history

```
git branch NEW-BRANCH-NAME COMMIT-IDENTIFIER
git add -p
git merge --no-ff FIRST-BRANCH-NAME SECOND-BRANCH-NAME
git merge --continue
git branch -d FIRST-BRANCH-NAME SECOND-BRANCH-NAME
```

Summary and questions

- Repository initialization
- Adding files, repository exploration
- Editor and alias setup
- Reordering and fixing the history
- Alternative history (branch)
- Conflicting history

- Why to be pedant about a patchset?