MAD-X on GitHub

How to use Git(Hub) and the MAD-X workflow

Thomas Gläßle, Kyrre Sjobak

MAD-X meeting, June 22, 2017

Outline

Outline

What is Git? Core philosophy and ideas.

- A version control system
- Originally developed by Linus Torvalds for the Linux kernel
- Distributed
- Everyone receives a full copy of the whole repository
- Data integrity and safety against tempering
- Can reorder/edit/remove commits before publication

What is Git? Core philosophy and ideas.

- A version control system
- Originally developed by Linus Torvalds for the Linux kernel
- Distributed
- Everyone receives a full copy of the whole repository
- Data integrity and safety against tempering
- Can reorder/edit/remove commits before publication

What is Git? Core philosophy and ideas.

- A version control system
- Originally developed by Linus Torvalds for the Linux kernel
- Distributed
- Everyone receives a full copy of the whole repository
- Data integrity and safety against tempering
- Can reorder/edit/remove commits before publication

What is Git? Core philosophy and ideas.

- A version control system
- Originally developed by Linus Torvalds for the Linux kernel
- Distributed
- Everyone receives a full copy of the whole repository
- Data integrity and safety against tempering
- Can reorder/edit/remove commits before publication

What is GitHub?

- Company which offers hosting git repositories + tools (issue tracker, wiki, web interface,. . .)
- Free for open-source software
- Recommended for CERN projects which have outside collaborators
  - For closed projects, CERN hosted GitLab is recommended[a].
- About 26 million users (March 2017)
- No risk in using an external provider due to git's decentralized model.

https://github.com

a) https://cern.service-now.com/service-portal/article.do?n=KB0003132&s=gitlab

Moving from SVN to git: Why

- Simplify branch centric development
- Commit without server access
- Accessible for outside collaborators
- GitHub: code-review and pre-merge discussion
- History edits:
  - bug fixes can be squashed before merging
  - cleaner history, with ability to bisect
- Fast and flexible
- Easy to use for personal projects
- Many users!

Outline

Using Git: The .git repository

Create repository in current folder:

```
$ cd toy-project
$ git init
Initialized empty Git repository (...)
```

Using Git: The .git repository

Create repository in current folder:

```
$ cd toy-project
$ git init
Initialized empty Git repository (...)
```

Creates a .git subdirectory:

```
$ tree -aL 2
.
|-- .git
    |-- branches/
    |-- config
    |-- description
    |-- HEAD
    |-- hooks/
    |-- info/
    |-- objects/
    |-- refs/

6 directories, 3 files
```

Repository:

- **config:** personal config, remotes, ...
- **HEAD:** reference to current commit.
- **objects:** versioned files, trees, commits, tags.
- **refs:** references to objects (branches, tags)

Using Git: The .git repository

Create repository in current folder:

```
$ cd toy-project
$ git init
Initialized empty Git repository (...)
```

Creates a .git subdirectory:

```
$ tree -aL 2
.
|-- .git
    |-- branches/
    |-- config
    |-- description
    |-- HEAD
    |-- hooks/
    |-- info/
    |-- objects/
    |-- refs/

6 directories, 3 files
```

Repository:

- **config:** personal config, remotes, ...
- **HEAD:** reference to current commit.
- **objects:** versioned files, trees, commits, tags.
- **refs:** references to objects (branches, tags)

Using Git: The .git repository

Create repository in current folder:

```
$ cd toy-project
$ git init
Initialized empty Git repository (...)
```

Creates a .git subdirectory:

```
$ tree -aL 2
.
|-- .git
    |-- branches/
    |-- config
    |-- description
    |-- HEAD
    |-- hooks/
    |-- info/
    |-- objects/
    |-- refs/

6 directories, 3 files
```

Repository:

- **config:** personal config, remotes, ...
- **HEAD:** reference to current commit.
- **objects:** versioned files, trees, commits, tags.
- **refs:** references to objects (branches, tags)

Using Git: The .git repository

Create repository in current folder:

```
$ cd toy-project
$ git init
Initialized empty Git repository (...)
```

Creates a .git subdirectory:

```
$ tree -aL 2
.
|-- .git
    |-- branches/
    |-- config
    |-- description
    |-- HEAD
    |-- hooks/
    |-- info/
    |-- objects/
    |-- refs/

6 directories, 3 files
```

Repository:

- **config:** personal config, remotes, ...
- **HEAD:** reference to current commit.
- **objects:** versioned files, trees, commits, tags.
- **refs:** references to objects (branches, tags)

Using Git: The .git repository

Create repository in current folder:

```
$ cd toy-project
$ git init
Initialized empty Git repository (...)
```

Creates a .git subdirectory:

```
$ tree -aL 2
.
|-- .git
    |-- branches/
    |-- config
    |-- description
    |-- HEAD
    |-- hooks/
    |-- info/
    |-- objects/
    |-- refs/

6 directories, 3 files
```

Repository:

- **config:** personal config, remotes, ...
- **HEAD:** reference to current commit.
- **objects:** versioned files, trees, commits, tags.
- **refs:** references to objects (branches, tags)

Using Git: Preparing commits

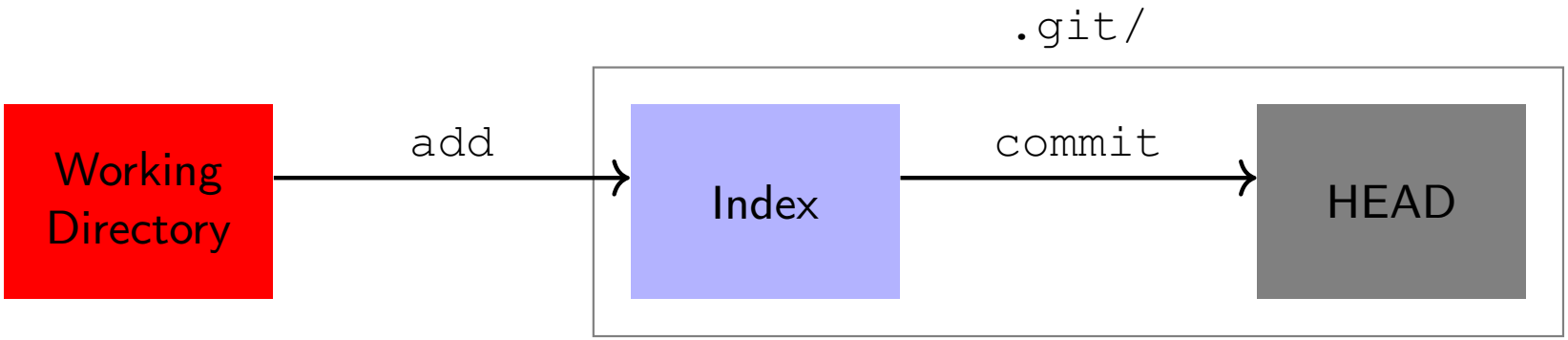
Add a new file under version control:

```
$ echo Foo > foo.txt
$ git add foo.txt
$ git commit -m "Add incredibly useful file"
```

Using Git: Preparing commits

Add a new file under version control:

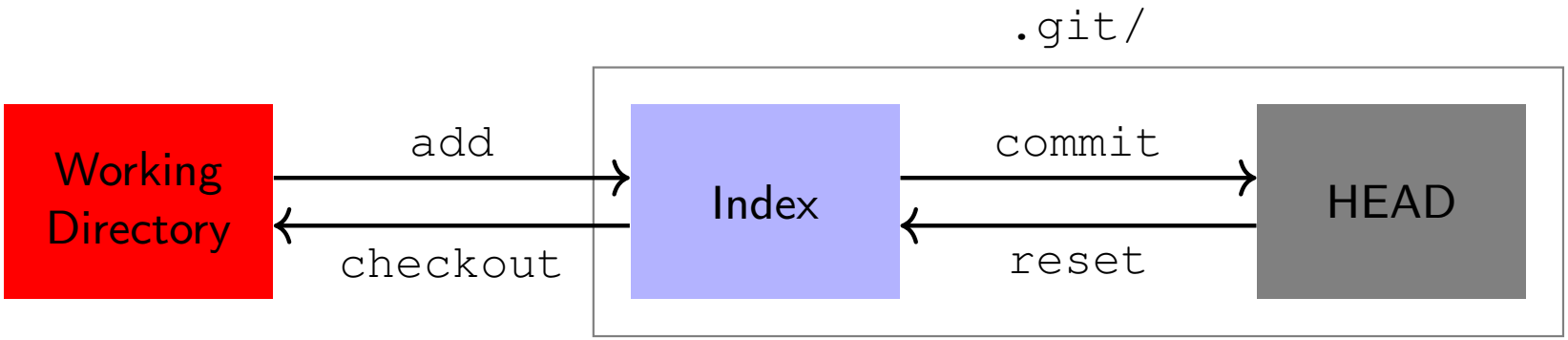
```
$ echo Foo > foo.txt
$ git add foo.txt
$ git commit -m "Add incredibly useful file"
```



Using Git: Preparing commits

Add a new file under version control:

```
$ echo Foo > foo.txt
$ git add foo.txt
$ git commit -m "Add incredibly useful file"
```




```
git add FILE          # Add existing file to the index
git rm FILE           # Remove file (FS and index)
git mv FROM DEST      # Move/rename file (FS and index)
git reset -- FILE     # Remove changes from index
git checkout FILE     # Checkout file from index
```

Using Git: Status summary

Let's modify things:

```
$ echo "another line" >> foo.txt
$ echo "another file"  > bar.txt
```

Using Git: Status summary

Let's modify things:

```
$ echo "another line" >> foo.txt
$ echo "another file"  > bar.txt
```

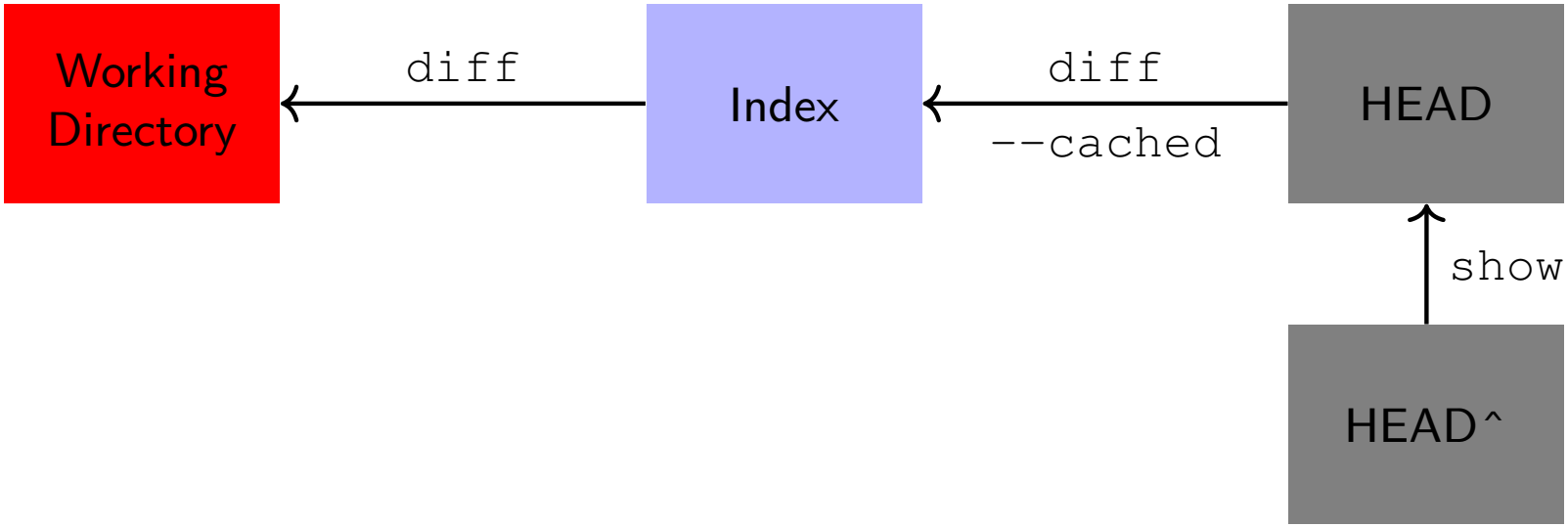
Summary of the current state:

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

        modified:   foo.txt

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

        bar.txt

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

Using Git: Diffs

Check for more details:

```
git diff                # between index and worktree
git diff --cached       # between HEAD and index
git show                # changes + message
```



Using Git: History

Append another commit to the current branch:

```
$ git add bar.txt
$ git commit -m "Add file bar.txt"
```

Using Git: History

Append another commit to the current branch:

```
$ git add bar.txt
$ git commit -m "Add file bar.txt"
```

We now have 2 commits:

```
$ git log
 (HEAD -> master)
Commit:  a77020c2c725e1b651667f6ceeee85885f6aa49a
Author:  Thomas Glaessle <t_glaessle@gmx.de>
Date:    Wed Jun 21 15:12:19 2017 +0200 (7 seconds ago)
Subject: Add file bar.txt


Commit:  917359f863e30c843760c539db94b4f6c50ed639
Author:  Thomas Glaessle <t_glaessle@gmx.de>
Date:    Wed Jun 21 15:04:05 2017 +0200 (8 minutes ago)
Subject: Add incredibly useful file
```

Using Git: History

Append another commit to the current branch:

```
$ git add bar.txt
$ git commit -m "Add file bar.txt"
```

We now have 2 commits:

```
$ git log
 (HEAD -> master)
Commit:  a77020c2c725e1b651667f6ceeee85885f6aa49a
Author:  Thomas Glaessle <t_glaessle@gmx.de>
Date:    Wed Jun 21 15:12:19 2017 +0200 (7 seconds ago)
Subject: Add file bar.txt


Commit:  917359f863e30c843760c539db94b4f6c50ed639
Author:  Thomas Glaessle <t_glaessle@gmx.de>
Date:    Wed Jun 21 15:04:05 2017 +0200 (8 minutes ago)
Subject: Add incredibly useful file
```

Using Git: History

Append another commit to the current branch:

```
$ git add bar.txt
$ git commit -m "Add file bar.txt"
```

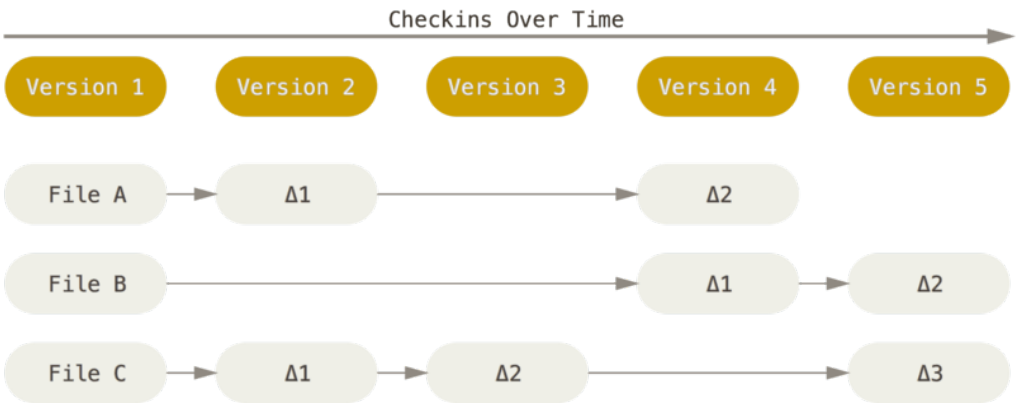
We now have 2 commits:

```
$ git log
 (HEAD -> master)
Commit: a77020c2c725e1b651667f6ceeee85885f6aa49a
Author:  Thomas Glaessle <t_glaessle@gmx.de>
Date:    Wed Jun 21 15:12:19 2017 +0200 (7 seconds ago)
Subject: Add file bar.txt


Commit: 917359f863e30c843760c539db94b4f6c50ed639
Author:  Thomas Glaessle <t_glaessle@gmx.de>
Date:    Wed Jun 21 15:04:05 2017 +0200 (8 minutes ago)
Subject: Add incredibly useful file
```

Objects are identified by SHA1 hash → immutable

Using Git: Revisions

Most VCS, history is stored as file based changes:



Git commits are snapshots:

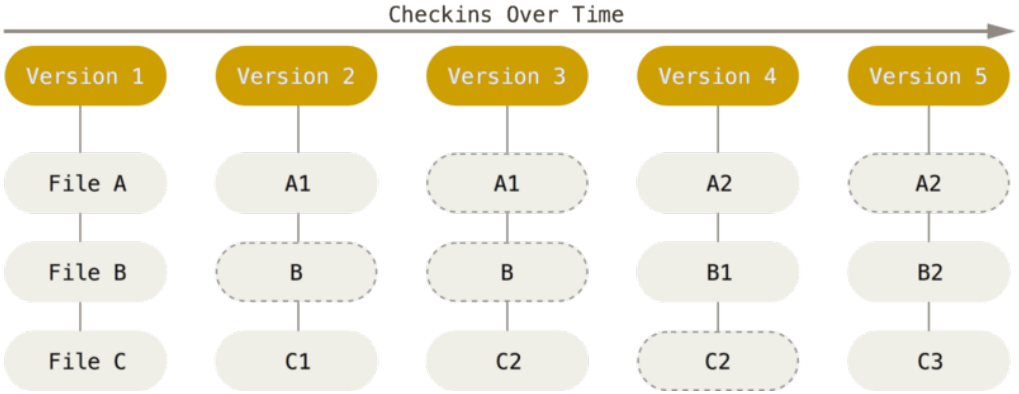
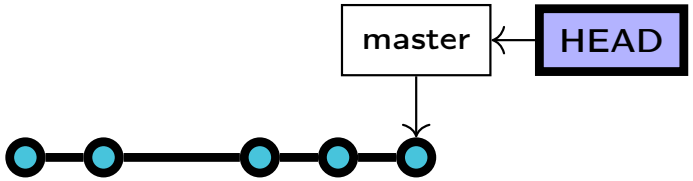


image source: `https://git-scm.com/book/en/v2/Getting-Started-Git-Basics`

Using Git: Branches

Switch to new branch:

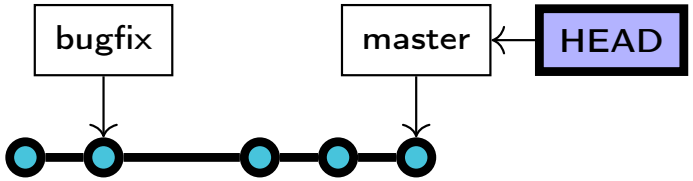
```
git branch bugfix HEAD~3    # Create branch
git checkout bugfix         # Switch to branch
```



Using Git: Branches

Switch to new branch:

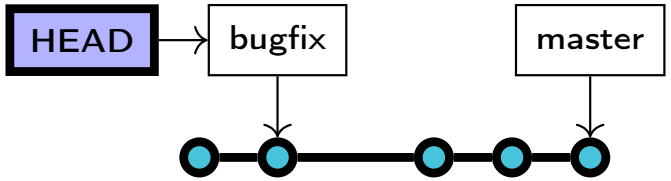
```
git branch bugfix HEAD~3    # Create branch
git checkout bugfix         # Switch to branch
```



Using Git: Branches

Switch to new branch:

```
git branch bugfix HEAD~3    # Create branch
git checkout bugfix         # Switch to branch
```

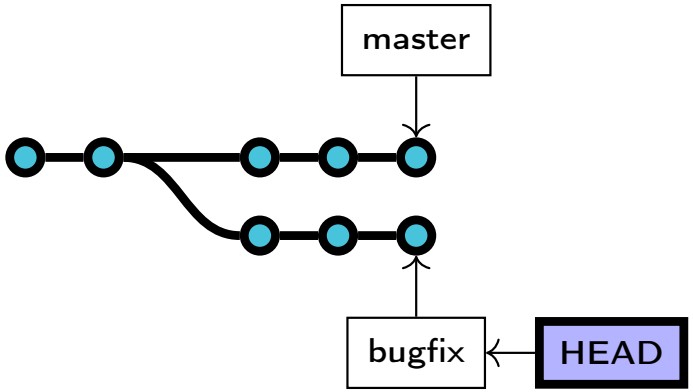


Using Git: Branches

Switch to new branch:

```
git branch bugfix HEAD~3    # Create branch
git checkout bugfix         # Switch to branch
```

(work, work. . .)



Using Git: Branches

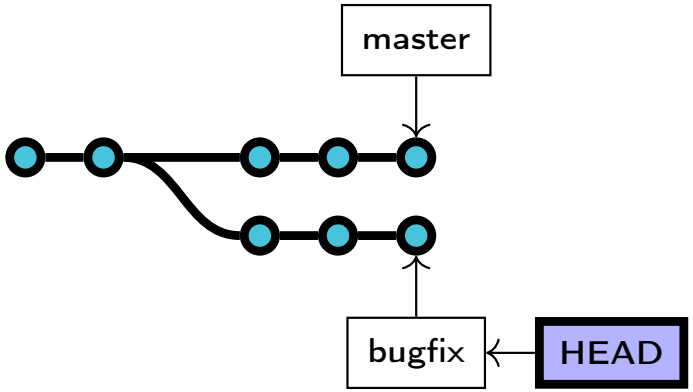
Switch to new branch:

```
git branch bugfix HEAD~3    # Create branch
git checkout bugfix         # Switch to branch
```

(work, work. . .)

Finally, merge branches:

```
git checkout master         # Switch back to master
git merge bugfix            # Merge branch into master
```



Using Git: Branches

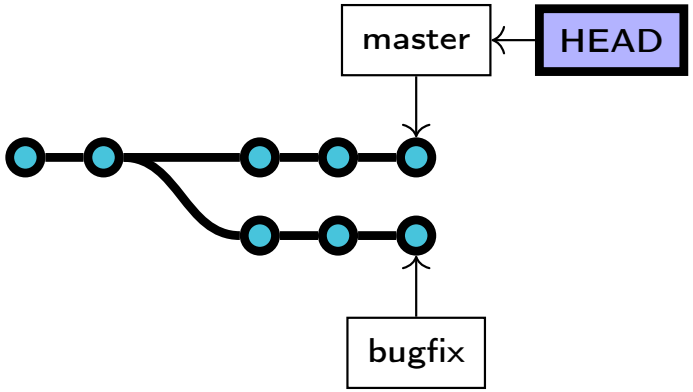
Switch to new branch:

```
git branch bugfix HEAD~3    # Create branch
git checkout bugfix         # Switch to branch
```

(work, work. . .)

Finally, merge branches:

```
git checkout master         # Switch back to master
git merge bugfix            # Merge branch into master
```



Using Git: Branches

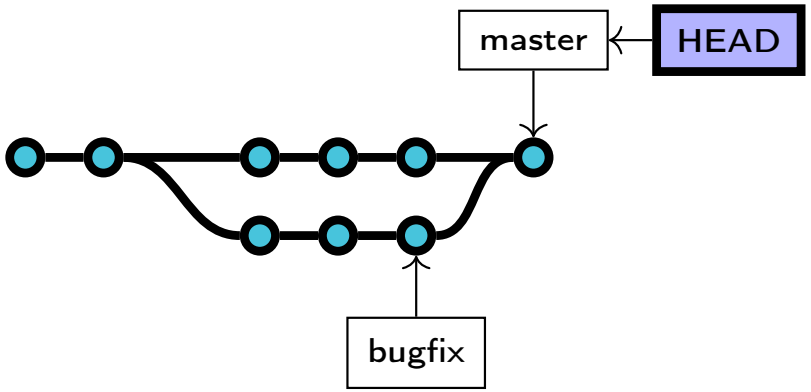
Switch to new branch:

```
git branch bugfix HEAD~3    # Create branch
git checkout bugfix         # Switch to branch
```

(work, work. . .)

Finally, merge branches:

```
git checkout master         # Switch back to master
git merge bugfix            # Merge branch into master
```



Using Git: References

Most git commands expect a *refspec*, i.e. a reference to an *object*.

Absolute refspecs, e.g.:

```
git revert   HEAD        # active commit
git diff     a2ff6d2c    # full or partial object hash
git checkout master      # tip commit of a branch
```

Using Git: References

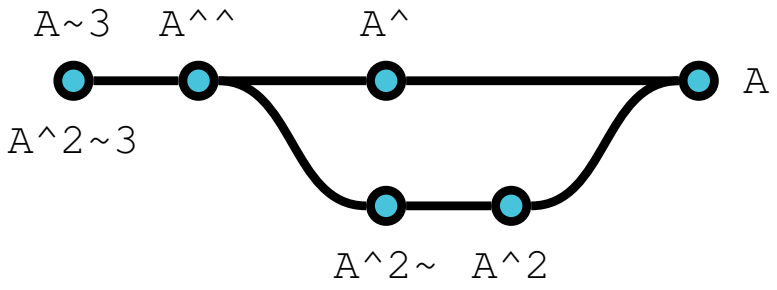
Most git commands expect a *refspec*, i.e. a reference to an *object*.

Absolute refspecs, e.g.:

```
git revert   HEAD        # active commit
git diff     a2ff6d2c    # full or partial object hash
git checkout master      # tip commit of a branch
```

Most important modifiers:

```
git show HEAD^2      # 2nd parent (for merges)
git show HEAD~3      # 3nd ancestor (via 1st parents)
```



Using Git: Remotes

A *remote* is a link to a related repository.

Fundamental commands:

```
git remote add origin URL   # Configure URL
git fetch origin            # Download objects, refs
git push                    # Upload objects, refs
```

Using Git: Remotes

A *remote* is a link to a related repository.

Fundamental commands:

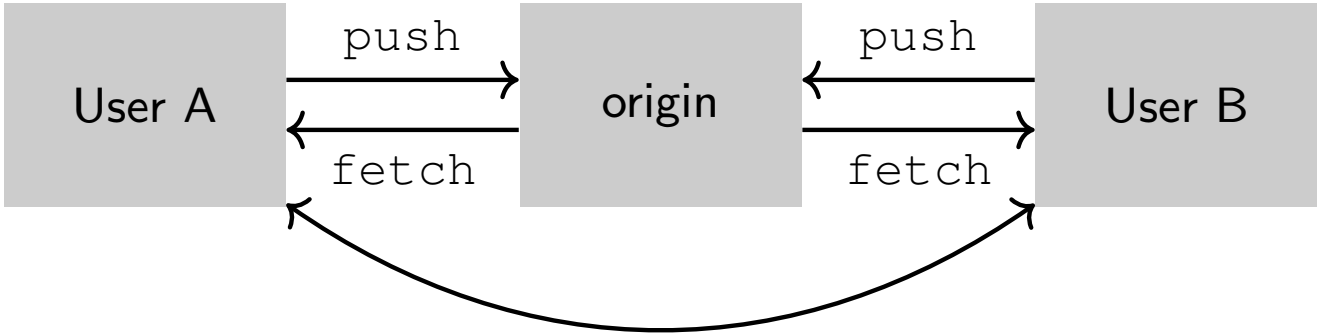
```
git remote add origin URL   # Configure URL
git fetch origin            # Download objects, refs
git push                    # Upload objects, refs
```

Using Git: Remotes

A *remote* is a link to a related repository.

Fundamental commands:

```
git remote add origin URL   # Configure URL
git fetch origin            # Download objects, refs
git push                    # Upload objects, refs
```

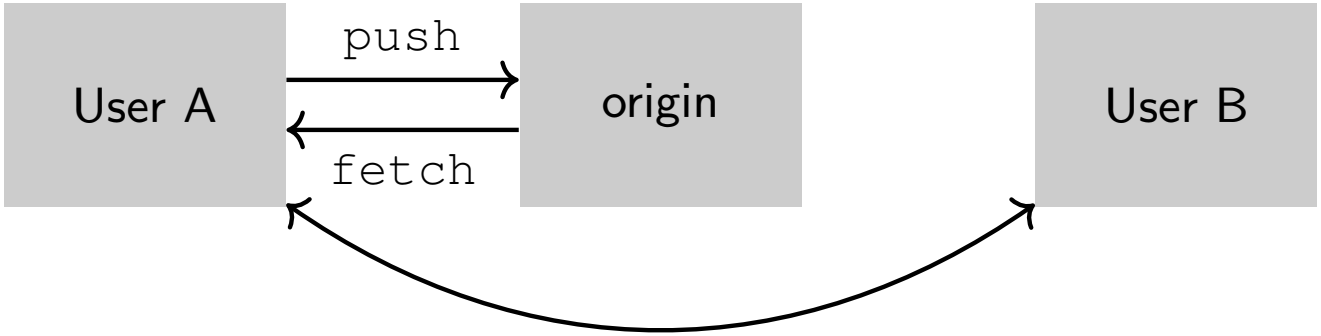


Using Git: Remotes

A *remote* is a link to a related repository.

Fundamental commands:

```
git remote add origin URL   # Configure URL
git fetch origin            # Download objects, refs
git push                    # Upload objects, refs
```

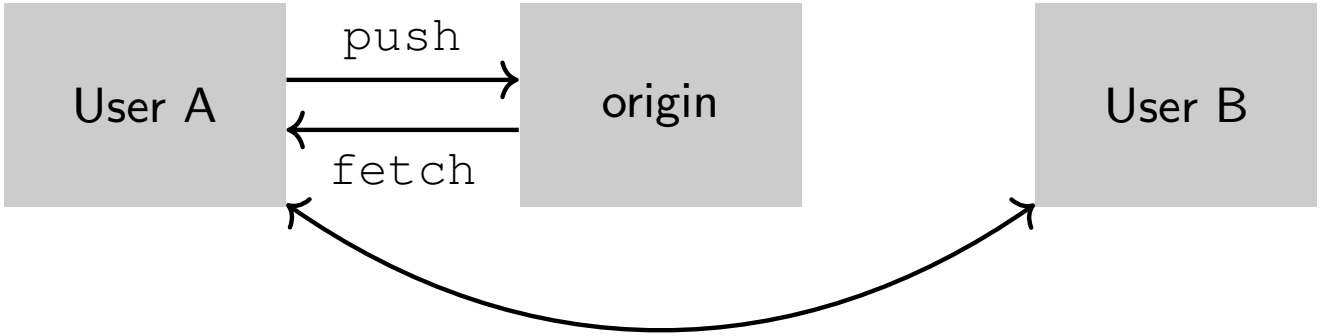


Using Git: Remotes

A *remote* is a link to a related repository.

Fundamental commands:

```
git remote add origin URL   # Configure URL
git fetch origin            # Download objects, refs
git push                    # Upload objects, refs
```



Remotes are defined in your personal `.git/config`
(→ not synced with collaborators.)

Using Git: More commands

More useful commands:

```
git clone              # init + remote add + fetch + checkout
git pull               # fetch + merge

git blame              # who changed what/when
git bisect             # find commit that introduced a bug

git submodule          # sub-repositories
git tag                # releases

git stash              # save workdir and reset to HEAD

git cherry-pick        # Apply commit from another branch
git revert             # Neutralize an earlier commit
git rebase             # change recent commits
git filter-branch      # automated history rewrites
```

and many more. . .

Using Git: Getting help

Commands have many useful options, find them:

```
git help TOPIC          # help about a subcommand
```

https://git-scm.com/documentation

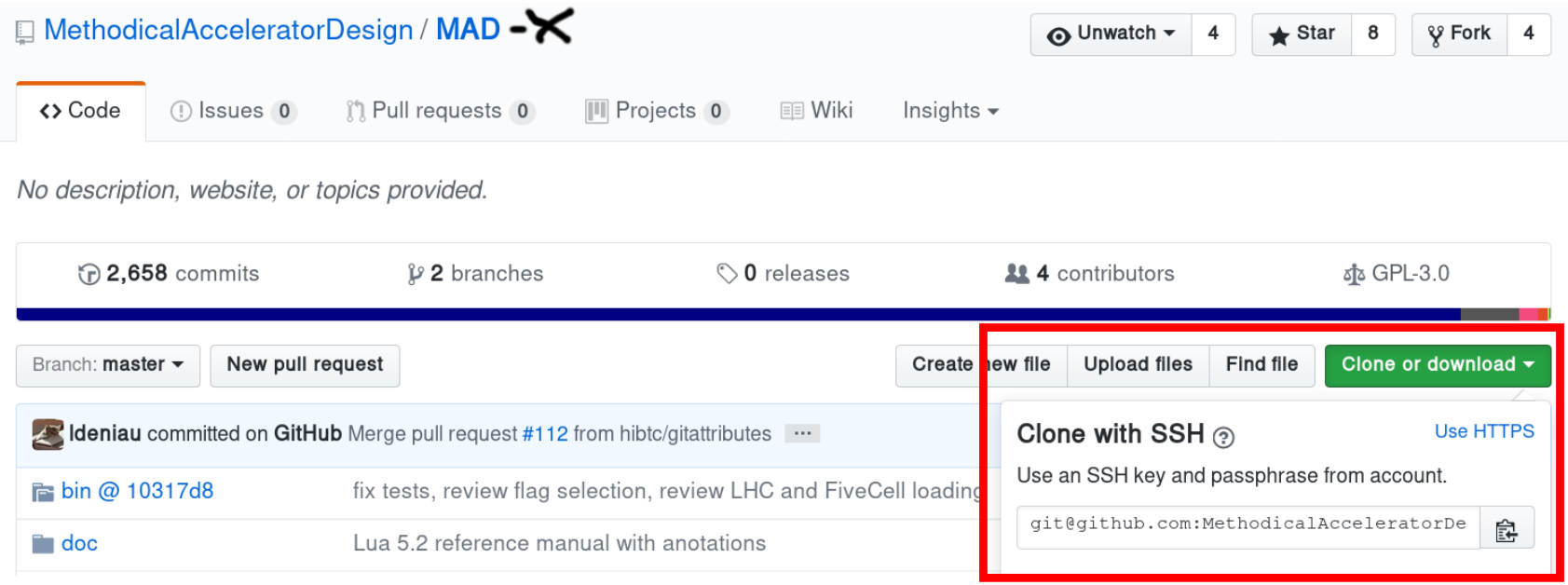
https://github.com/MethodicalAcceleratorDesign/MAD-X/wiki/Git

Using Git: General remarks

- Commit early and often!
- **Do not** blindly `commit -a`
- Use `diff --cached`, `diff`, `status` before committing
- Check which branch you have checked out: `git branch`
- Never commit large binary files!
- You can change your commits until merged to upstream
- Everything is local (and safe) until you push

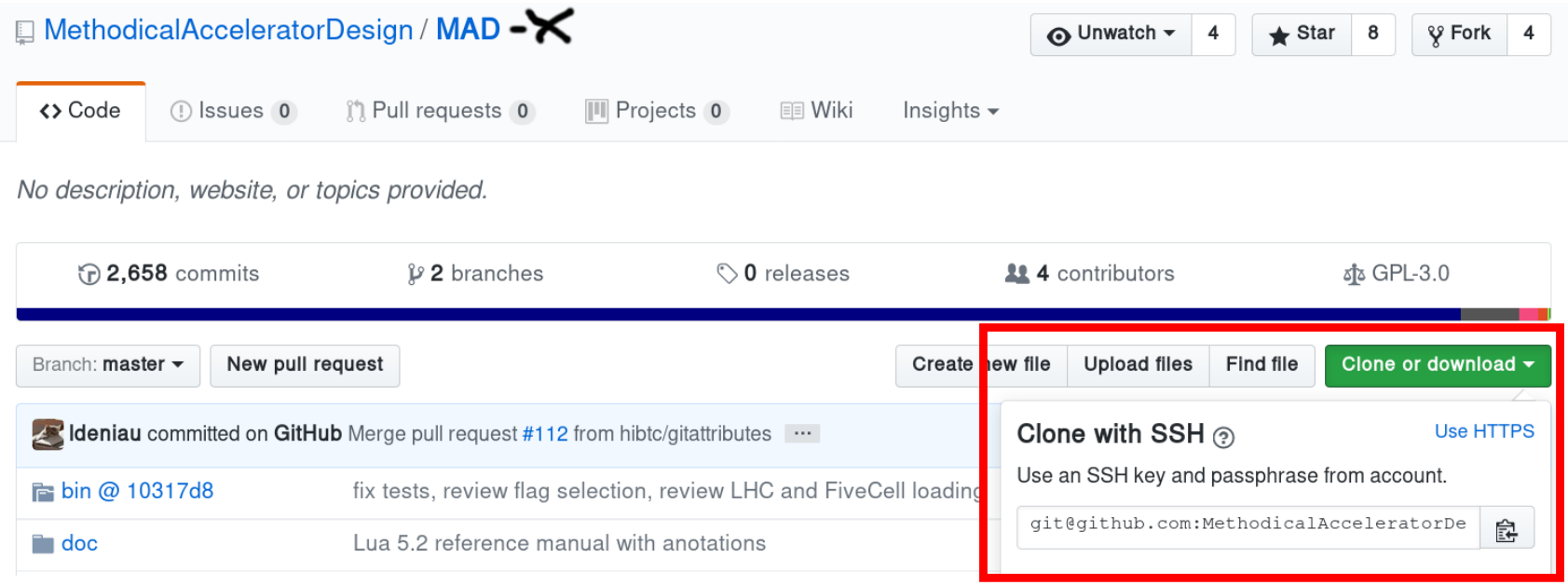
Outline

MAD-X Workflow: upstream




```
git clone https://github.com/MethodicalAcceleratorDesign/MAD-X
git clone     git@github.com:MethodicalAcceleratorDesign/MAD-X
```

MAD-X Workflow: upstream



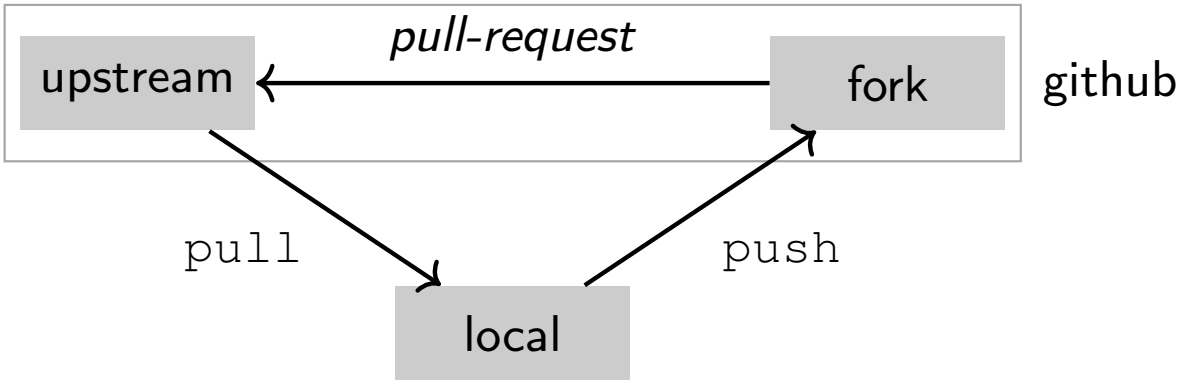

```
git clone https://github.com/MethodicalAcceleratorDesign/MAD-X
git clone     git@github.com:MethodicalAcceleratorDesign/MAD-X
```

Before running git on lxPlus:

```
scl enable rh-git29 bash
```

MAD-X Workflow: Overview

1. **Fork the official repo** to your own github username.
2. **Create a branch** for your feature/bugfix
3. **Work on your branch** until the feature is tested and ready for inclusion.
4. **Push your changes** to your github fork
5. **Create a pull-request** for inclusion in the official MAD-X repository.



No-one commits directly to upstream!

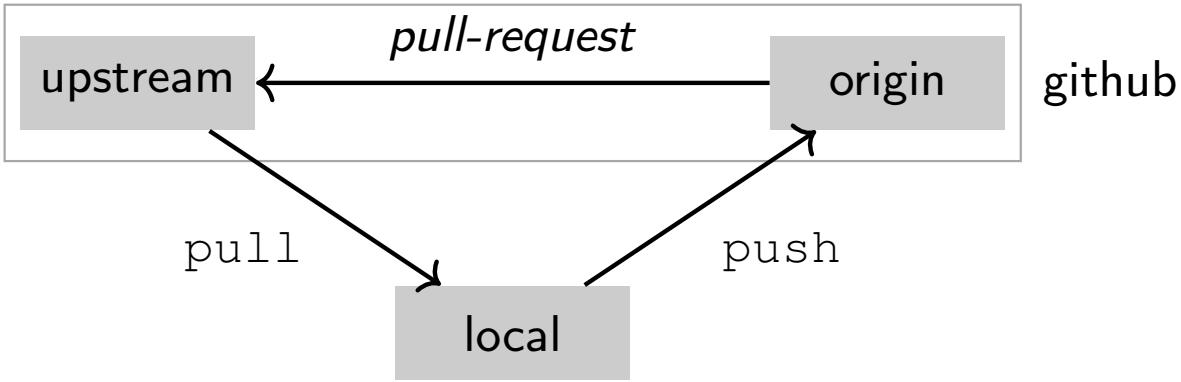
MAD-X Workflow: Setup your repository

- Fork the MAD-X repository to your github username
- Setup an SSH key
- Clone to your local workstation

```
git clone git@github.com/USERNAME/MAD-X
```

- Add the upstream:

```
git remote add upstream \
  git@github.com/MethodicalAcceleratorDesign/MAD-X
git fetch upstream
```



MAD-X Workflow: Start a feature branch

Create new feature branch:

```
git checkout -b fix-all-segfaults upstream/master
git push -u origin fix-all-segfaults
```

MAD-X Workflow: Start a feature branch

Create new feature branch:

```
git checkout -b fix-all-segfaults upstream/master
git push -u origin fix-all-segfaults
```

(work on branch. . .)

Commit and push to your fork:

```
git commit          # commit locally
git show            # check commit
git push            # push changes to fork
```

(repeat. . .)

MAD-X Workflow: Merging your changes

When your branch is ready for inclusion:

1. Create pull request on the github website
2. Wait for code review or invitation to coffee meeting
3. Implement requested fixes
4. Update your master (optional):

```
git checkout master
git pull upstream master
git push origin master
```

5. Delete your branch:

```
git branch -d fix-all-segfaults      # locally
git push origin :fix-all-segfaults   # on fork
```

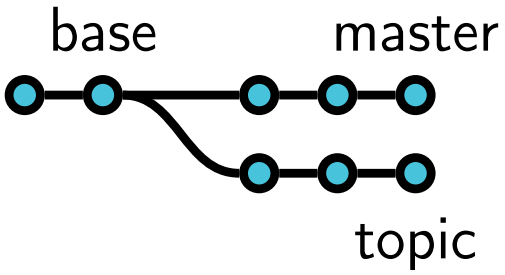
Outline

Merge conflicts (1)

Merging two branches that changed the same or adjacent lines is a **conflict** and requires manual intervention.

When submitting a pull-request that has a merge conflict, you will be asked to merge and resolve it manually:

```
git pull upstream/master
```

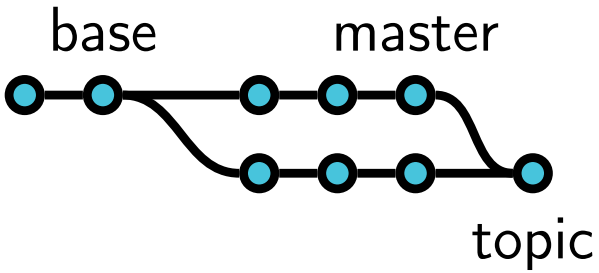


Merge conflicts (1)

Merging two branches that changed the same or adjacent lines is a **conflict** and requires manual intervention.

When submitting a pull-request that has a merge conflict, you will be asked to merge and resolve it manually:

```
git pull upstream/master
```

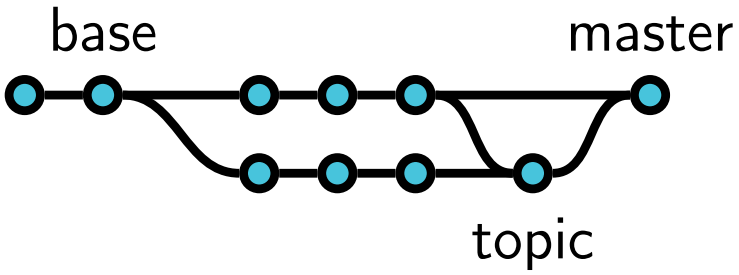


Merge conflicts (1)

Merging two branches that changed the same or adjacent lines is a **conflict** and requires manual intervention.

When submitting a pull-request that has a merge conflict, you will be asked to merge and resolve it manually:

```
git pull upstream/master
```



Merge conflicts (2)

Example: Merging master into branch JavierBB

```
$ git merge master
Auto-merging SixTrack/sixtrack.s
CONFLICT (content): Merge conflict in SixTrack/sixtrack.s
Auto-merging SixTrack/compareSVN
Automatic merge failed; fix conflicts and then commit the result.
```

Can't deal with this today? You can postpone:

```
git merge --abort      # go back to pre-merge-attempt
```

Merge conflicts (3)

```
$ git status
# On branch JavierBB
# You have unmerged paths.
#   (fix conflicts and run "git commit")
#
# Changes to be committed:
#
#   ...
#
# Unmerged paths:
#   (use "git add <file>..." to mark resolution)
#
# both modified:      sixtrack.s
#
```

Merge conflicts (4)

Look for blocks like:

```
<<<<<<< HEAD
How it looks like in the new branch
=======
How it looks like in master
>>>>>>> master
```

Merge conflicts (4)

Look for blocks like:

```
<<<<<<< HEAD
How it looks like in the new branch
=======
How it looks like in master
>>>>>>> master
```

Remove the blocks, leave only the valid version, test, then commit:

```
git add FILE        # add your conflict resolution
git commit          # create the merge commit
git show            # check your commit!
```

Rebasing commits

Before merging upstream, edit your branch with `git rebase`:

- change order of commits
- insert/drop/modify commits
- squash multiple commits together
- reattach branch to another base commit

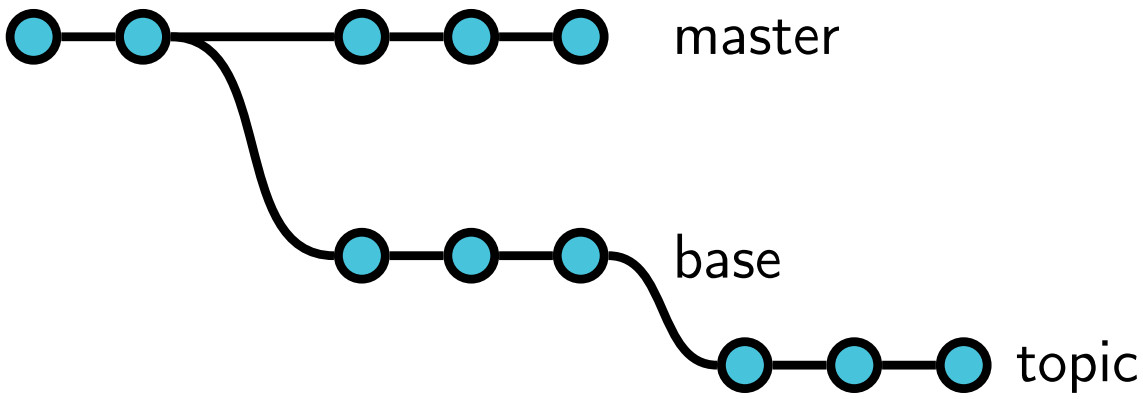
Basic usage:

```
git rebase -i HEAD~5     # edit/reorder/... last 5 commits
git rebase -i master     # edit all commits back to master
```

Rebasing commits

Reattach branch to another base commit:

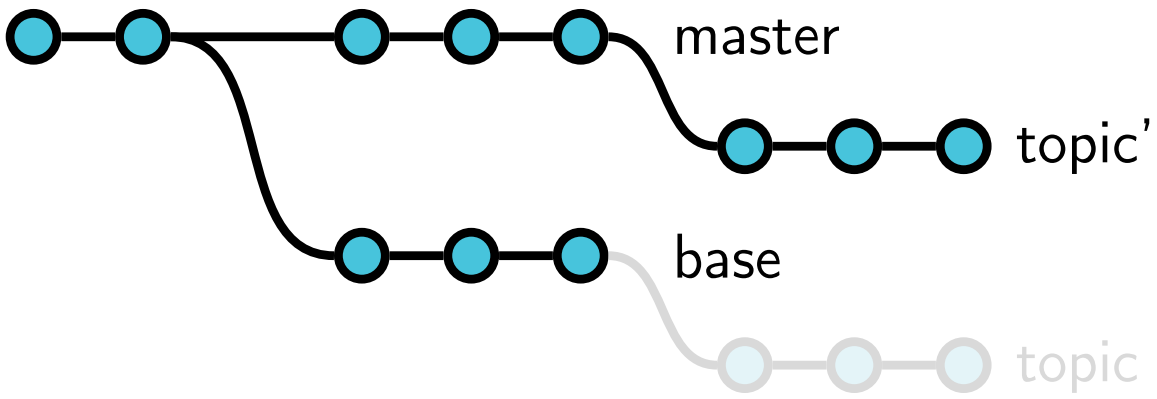
```
git rebase -i --onto master base topic
```



Rebasing commits

Reattach branch to another base commit:

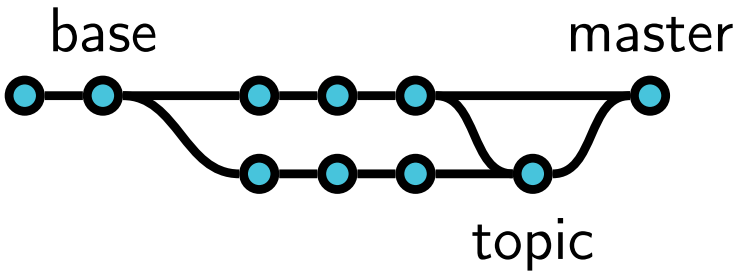
```
git rebase -i --onto master base topic
```



Rebasing commits

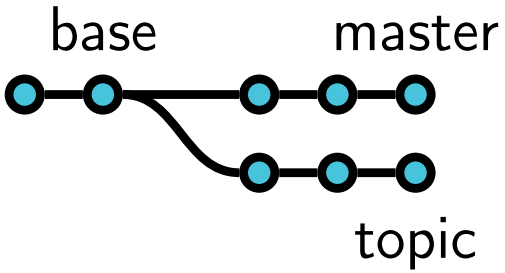
This can also be used as an often cleaner alternative for merging (but can also be more difficult in case of conflicts):

Merge:



Rebase:

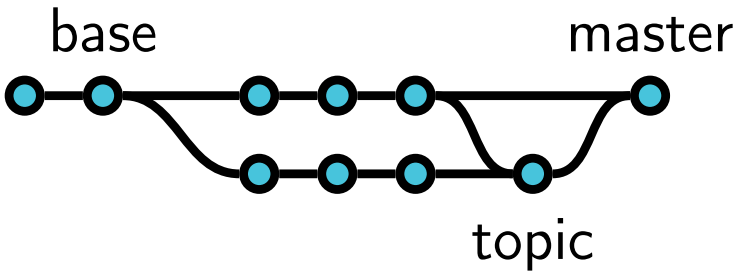
```
git rebase -i --onto master base topic
```



Rebasing commits

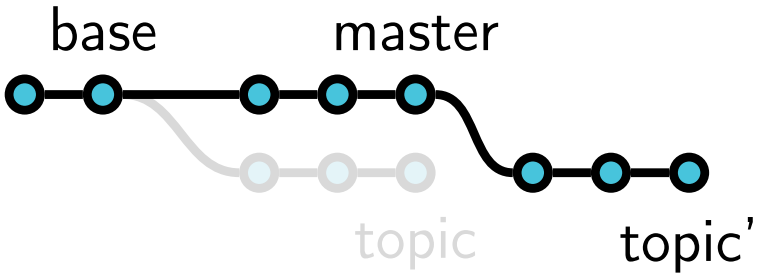
This can also be used as an often cleaner alternative for merging (but can also be more difficult in case of conflicts):

Merge:



Rebase:

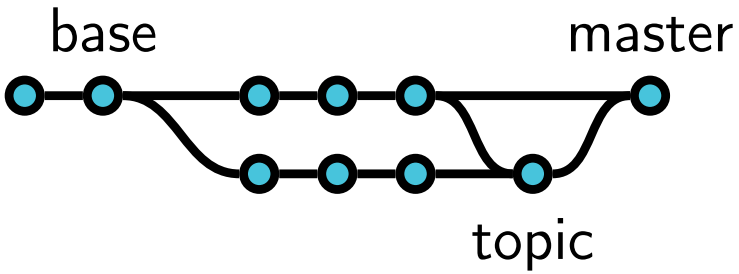
```
git rebase -i --onto master base topic
```



Rebasing commits

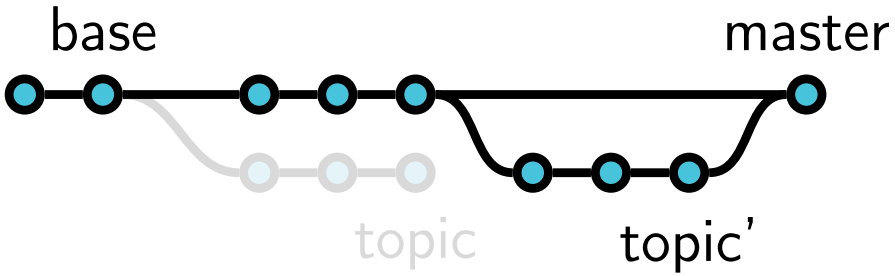
This can also be used as an often cleaner alternative for merging (but can also be more difficult in case of conflicts):

Merge:



Rebase:

```
git rebase -i --onto master base topic
```



Some tricks

- Abort commit: Leave commit message empty
- Use specific editor:

```
git config --global core.editor "emacs -nw"
```

- Use a GUI to explore repo:

```
git gui
```

- Fix a bad commit message or fixup changeset (if you haven't pushed yet):

```
git commit --amend
```

Summary and conclusions

- Moving MAD-X from SVN and to git
- Simplify contributions and parallel development
- Git(Hub) is a great tool for your own projects as well!

Appendix: Objects glossary

Contents of the `.git/` folder:

Repository

Objects, References, Hooks, ...

Object

Blob, Tree, Commit, Tag. An object is identified by the SHA-1 hash of its payload and therefore immutable!

Tree

Listing of filenames and file modes in one directory (flat).

Blob

Contents of a file, opaque to git.

Commit

Snapshot of the work tree, plus metadata: message, date, author, parents.

Tag

Static reference to a commit (plus signature, comment), signifies specific version/release.

Appendix: Advanced refspecs

You can specify commits by date, e.g.:

```
git show master^{yesterday}
git show master^{1 year 2 months ago}
```

Or search for commit message:

```
git show :/"fix nasty"
git show master^"{/fix nasty bug}"
```

and more. . . , see:

```
git help gitrevisions
```

or online at: `https://git-scm.com/docs/gitrevisions`

Appendix: Migration process

- Convert SVN repository using `subgit`
  - Link SVN usernames to real names
  - Branches and tags recognized
  - Add revision numbers to commit messages
- Cut the fat (340MiB → 106MiB download):
  - Removed some `.pdf` files
  - Extracted `.gz` files in history (→ packfiles!)
  - Moved `examples/` to a submodule
  - Aggressively compress objects
- Migrate issues from Trac to github
  - Replace revision numbers by links to commit IDs
  - Use github's REST API to create issues

For more information, contact me and see `https://coldfix.eu`