Git

SA² team
(Software Automation Architecture)

February 2019

AGENDA

Introduction	00’20
A little vocabulary	00’40
Basic usage	02’30
A bit further	00’30

Git

Introduction

Git

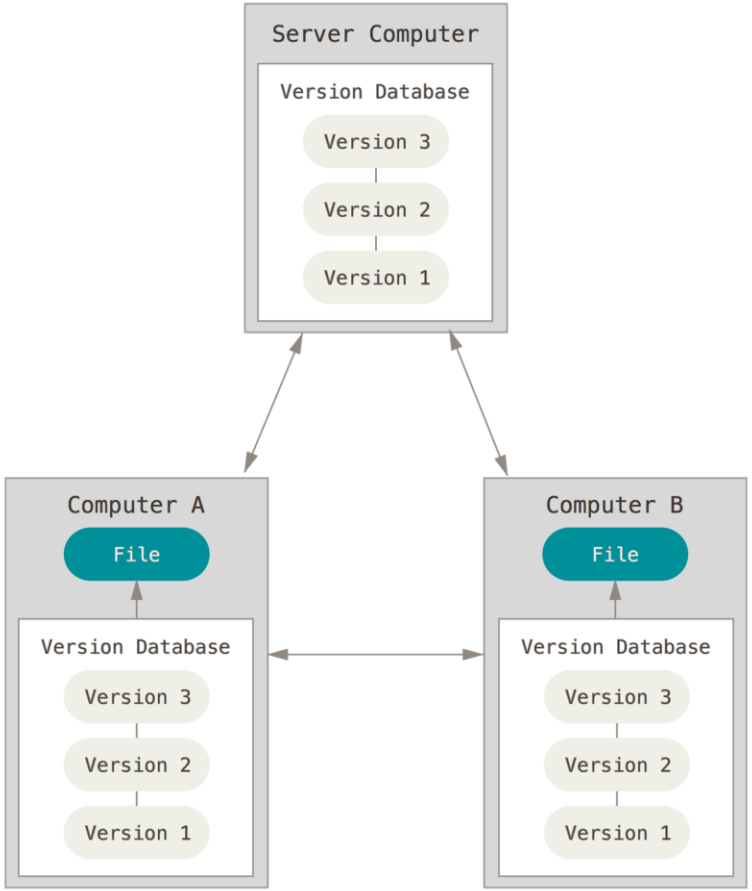
is a revision control system

- Created in 2005 by Linus Torvalds to develop the Linux kernel
- Free and open source
- Source code available at https://github.com/git/git
- Possible to contribute !

Git

is distributed

- You **clone** an entire repository
- Your local repo is a full copy of the original repo
- There is no single point of failure (as long as there is at least two copies of a repository)



Git

operations are mostly local

- Any repo is self-sufficient
- All the information, files and history are stored locally
- As a consequence operations are local… and fast

Commit and **checkout** are local too !

Git

branching is flexible

- Git encourages you to have multiple local branches
- Creation / merging / deletion of branches take seconds
- If you have a new idea, switch to a new branch and code

Git

is about snapshots

- Git stores state of the repo after each operation
- Git generally only adds data
- Almost any change on a branch can be recovered

Git

is not GitLab / GitHub / …

- Git is a **tool for version control**
- GitLab and GitHub are **hosting services** for Git repositories with additional collaboration features
- GitLab and GitHub concepts (Merge/Pull Request, fork, code review, etc) will be described in another presentation

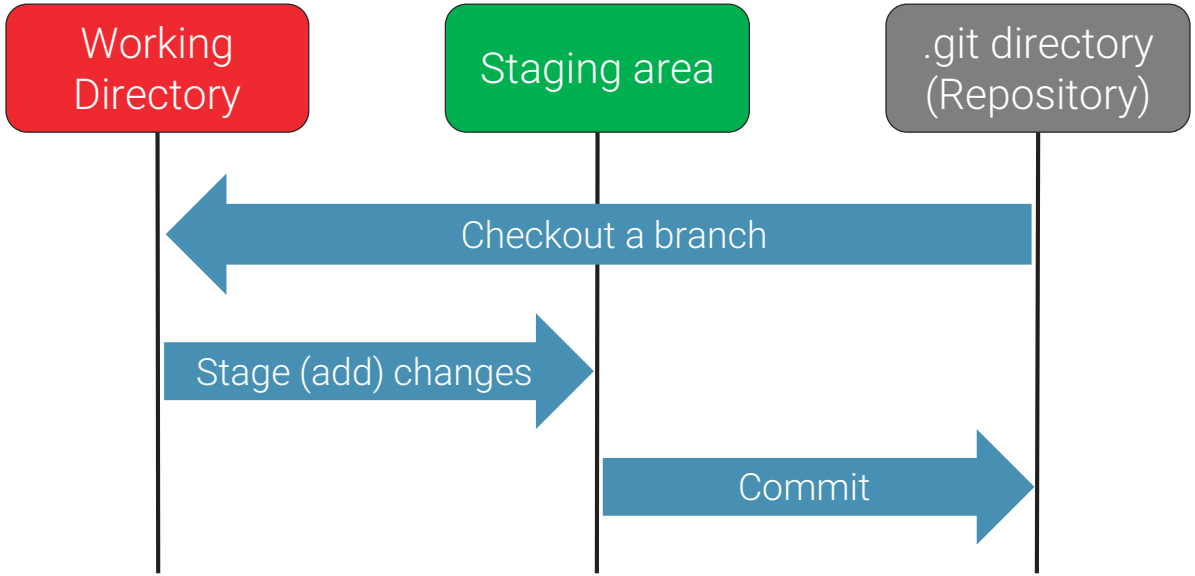
Git

A little vocabulary

sopra banking SOFTWARE

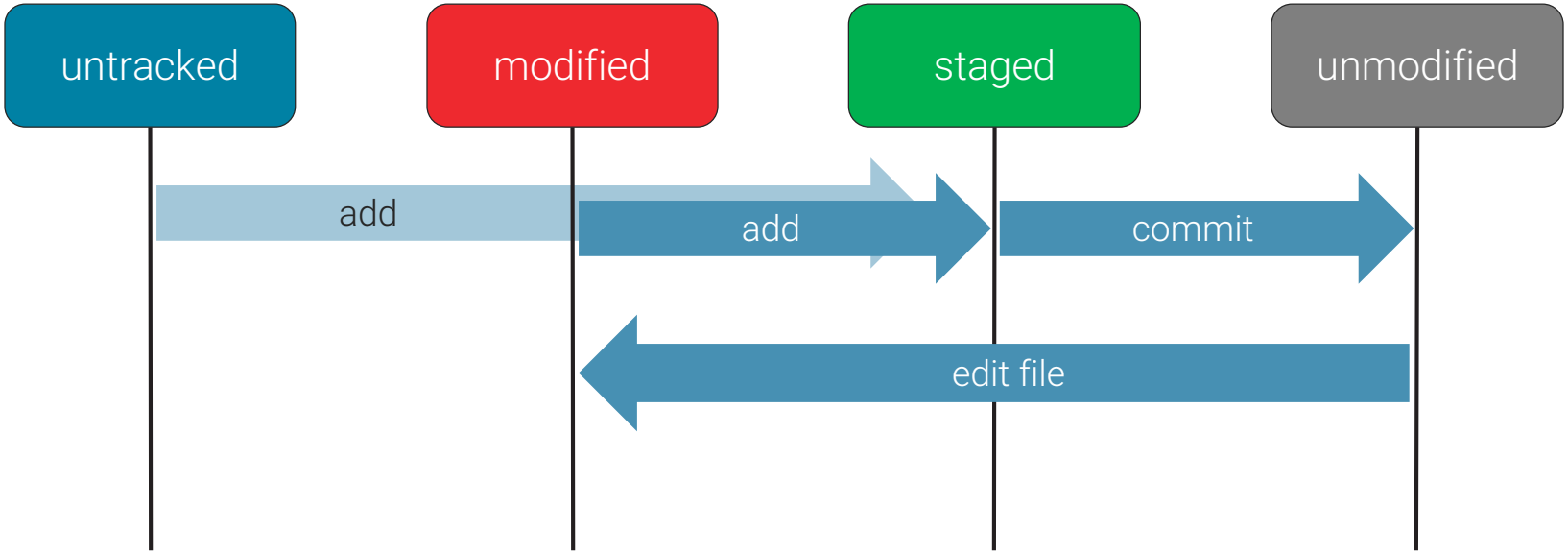
Git

The three zones



Git

The fourth states



Git

.gitignore

- Specifies intentionally untracked files that Git should ignore

```
# Eclipse
.classpath
.project
.settings/
target/
bin/
```

```
# Archives
*.7z
*.jar
*.rar
*.tar
*.zip
```

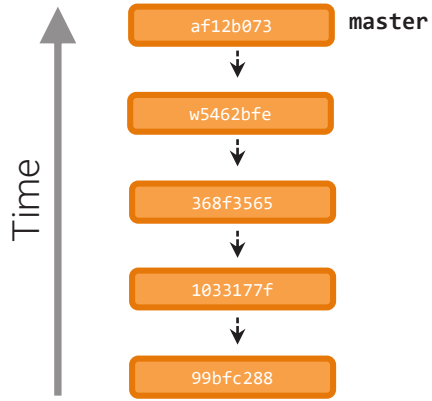
Useful .gitignore files for your project - https://www.gitignore.io/

Git

Commit

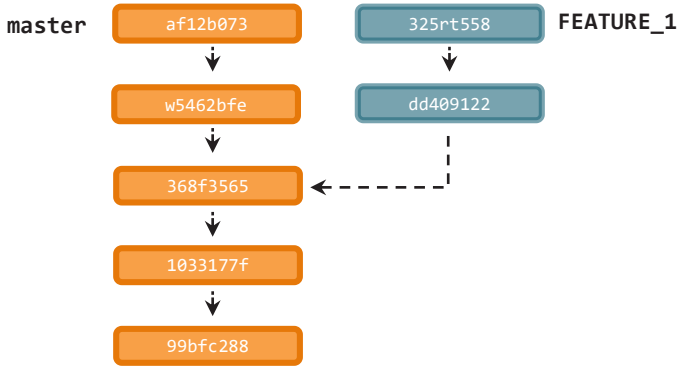
5d0c55cd6dbe1d1b779c7dd9d02910646e8fe988

- Holds one state of the repository
- Identified by a SHA1 hash like
- The SHA is globally unique
- Every commit has a parent commit
- A merge commit as two parent commits



Git

Branch



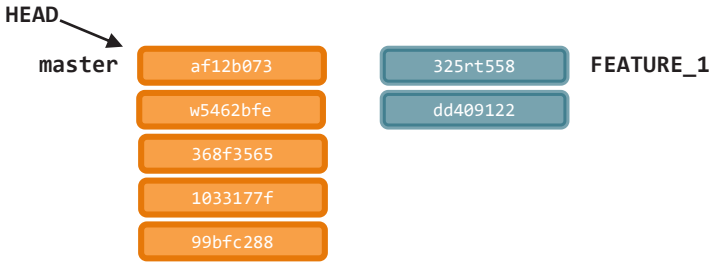
- A linked list of commits with a name
- Default branch is **master**
- Usually a branch is created to work on a new feature
- A branch is easy to create and delete

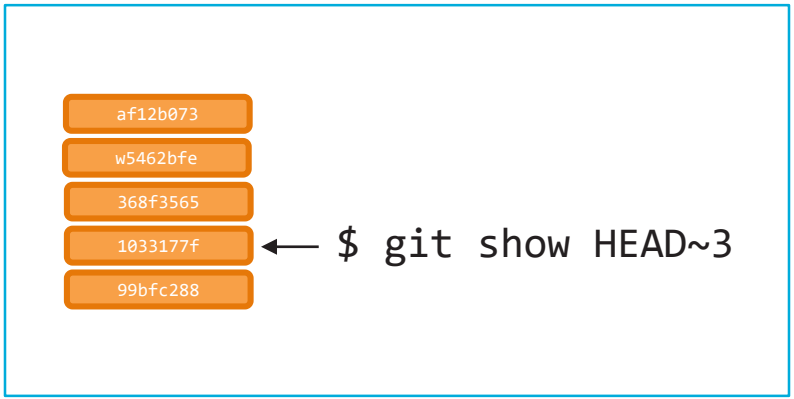
This is the basis of **Feature Branch** workflow

Git

HEAD

- Symbolic ref to the latest commit
- Only on currently checked out branch





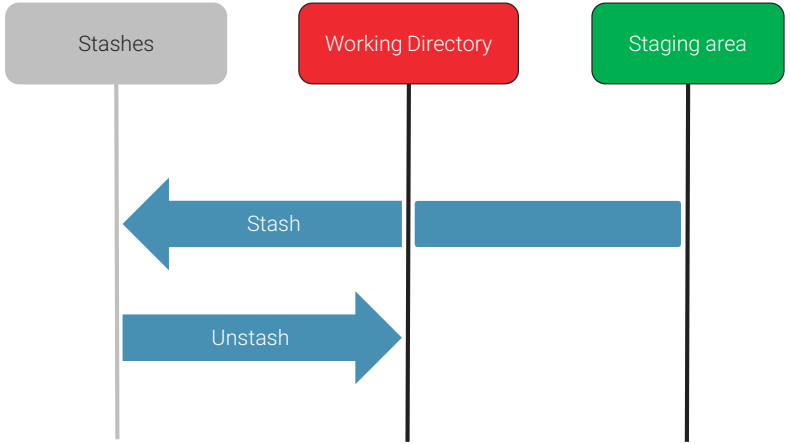
Git

Tag

- Same semantic as in SVN
- « branches move, tags don't »
- Usually created for releases

Git

Stash



- Like a « fourth zone »
- Save changes from your **working directory + staging area** on a stack away from any branch
- Number of stashes not limited
- You can reapply any of your stashes at any time on any of your local branches

Git

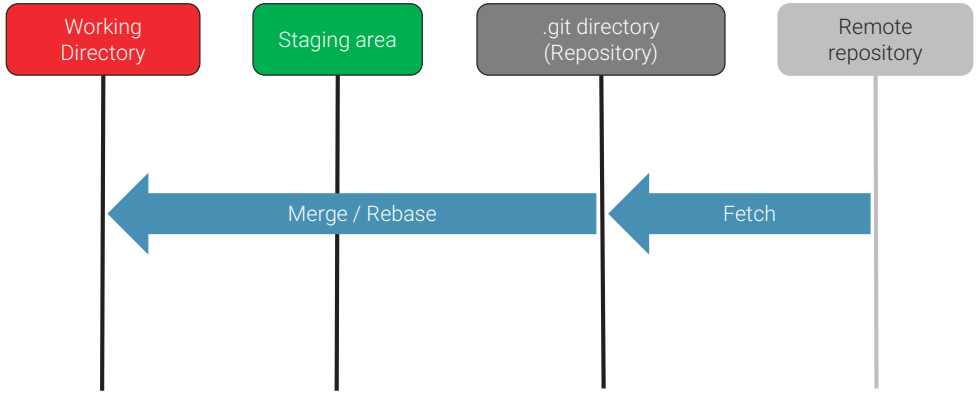
Remote

- Alias + URL that refers to another repository
- In URL protocol can be ssh / http(s) / git / local file
- **Several remotes** can be configured in a repository

```
$ git remote -v
origin  git@innersource.soprasteria.com:software-automation-architecture/git-training.git (fetch)
origin  git@innersource.soprasteria.com:software-automation-architecture/git-training.git (push)
evrignaud       https://innersource.soprasteria.com/etienne.vrignaud/git-training.git (fetch)
evrignaud       https://innersource.soprasteria.com/etienne.vrignaud/git-training.git (push)
```

Git

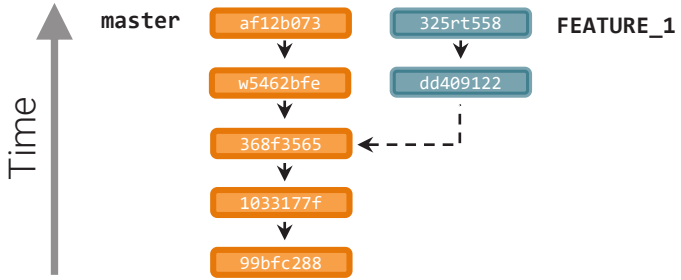
Pull



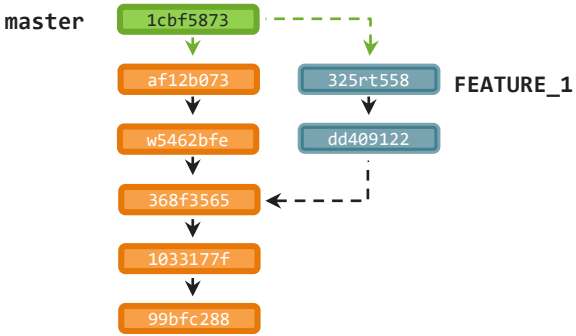
- Incorporates changes from a branch on a remote repo into the current local branch
- Shortcut for *fetch* + (***merge*** or ***rebase***)
- Pull will not work if you have unsaved local changes

--> `stash`, `commit` or `reset`

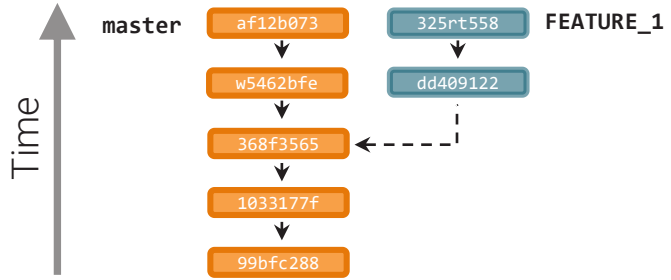
Git Merge



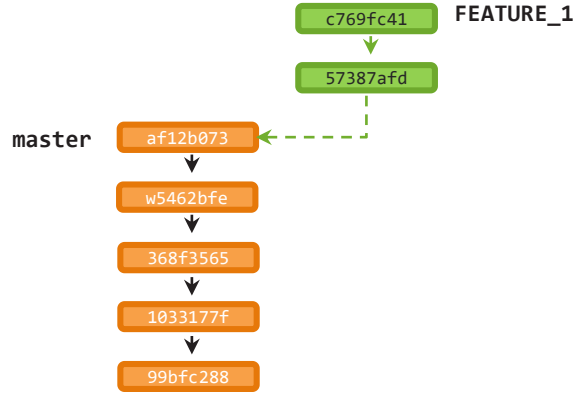

```
$ git checkout master
$ git merge FEATURE_1
```



Rebase

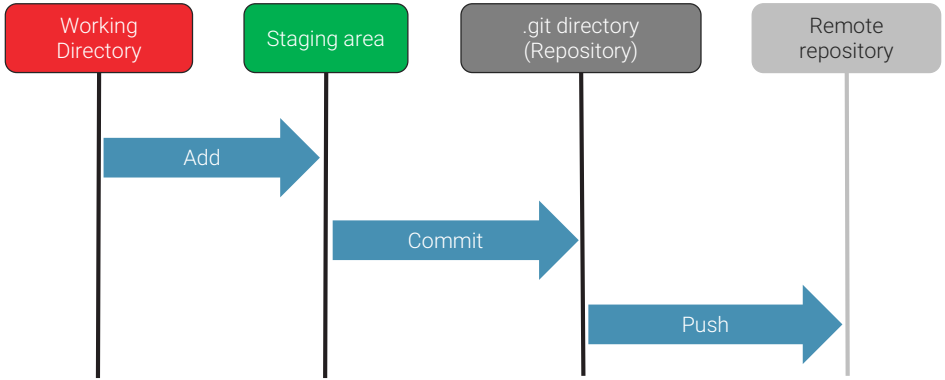



```
$ git checkout FEATURE_1
$ git rebase master
```



Git

Push



- Updates a remote branch from your local branch
- Sends objects (commits) necessary to complete the given branch

Git

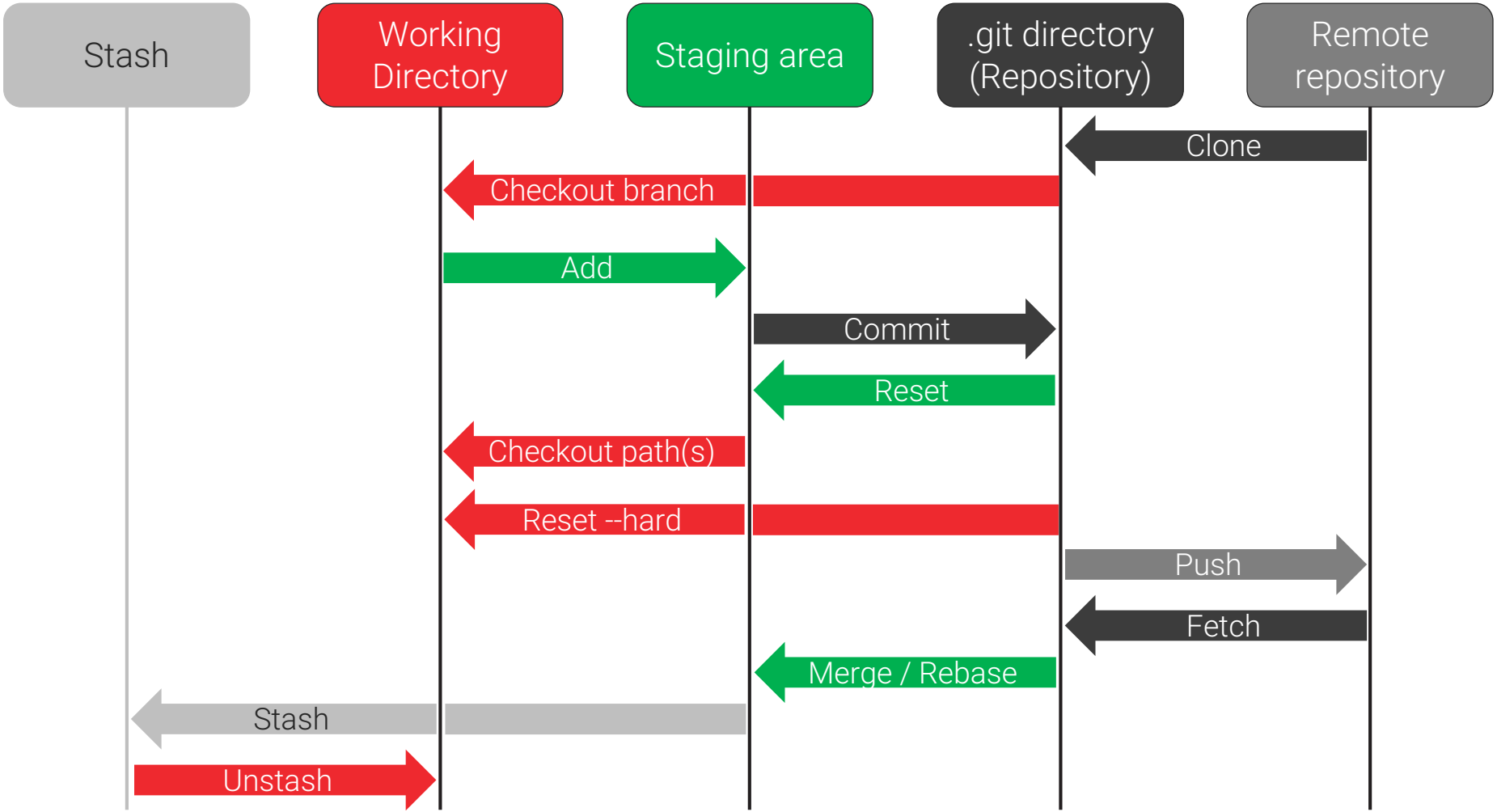
Basic usage

sopra banking SOFTWARE

Git

Notice

- Git offers tens of commands for all usage
- **Only the most common commands and options are presented in this section**
- For a comprehensive list of Git commands visit Git official page: https://git-scm.com/docs
- Git Cheat Sheet https://gitlab.com/gitlab-com/marketing/raw/master/design/print/git-cheatsheet/print-pdf/git-cheatsheet.pdf
- Stack **overflow** - https://stackoverflow.com/questions/tagged/git



Git

Install a Git client

- Create a HOME env variable in Windows with value `D:\Profiles\<user>` (HOME is the location where Git stores its configuration by default)
- Install **Git for Windows** https://gitforwindows.org/ (includes also **Git Bash**)
- You can then access Git Bash from Eclipse (right click on project > Show in > Git Bash)

Git

Configure SSH connection to Innersource

- Generate a key pair

```
$ ssh-keygen -t rsa
```

- Add the SSH key to your Innersource account

```
$ cat ~/.ssh/id_rsa.pub
```

Copy your SSH public key and **open** https://innersource.soprasteria.com/profile/keys

Paste your SSH public key in the ***Key*** **field**, fill the ***Title*** **field** & click on ***Add***

Git

Configure SSH connection to Innersource

- Active your SSH key when a new Git Bash window is opened

```
$ vi ~/.bash_profile

    eval "$(ssh-agent -s)"

    ssh-add ~/.ssh/id_rsa.pub

$ source ~/.bash_profile
```

Git

Test your connection to Innersource

```
$ git clone
git@innersource.soprasteria.com:Innersource-
meta/Welcome.git

$ rm -rf Welcome
```

see Innersource-meta wiki for more details

Git

config : manage global and repository configuration

- List properties

```
$ git config --list
```

- Configure your author name

```
$ git config --global user.name "Jean Dupont"
```

- Configure your email address

```
$ git config --global user.email jdu@sopra.com
```

Git

config : specific for Windows

- Allow to create a file or directory with a long path

```
$ git config --global core.longpaths true
```

- (for HTTPS connection) Store user / password inside the Git Credential Manager for Windows

```
$ git config --global credential.helper manager
```

(if needed go into the Windows credential manager to change your password: https://support.microsoft.com/en-us/help/4026814/windows-accessing-credential-manager)

Git

init : create a new **empty** repository

```
$ git init [<directory>]
```

Git

clone : copy an **existing** repository into a new directory

```
$ git clone [-b <branch>] <repo-url> [<directory>]
```

Git

Clone a repository

- Clone https://innersource.soprasteria.com/software-automation-architecture/git-training
- Tips:
  - Get the **SSH** URL of the repository to clone on Innersource

Git

Clone a repository

- Clone https://innersource.soprasteria.com/software-automation-architecture/git-training

```
$ cd /your/projects/folder

$ git clone
git@innersource.soprasteria.com:software-
automation-architecture/training/git-training.git

$ cd git-training
```

Git

branch : manage branches

- List branches

```
$ git branch
```

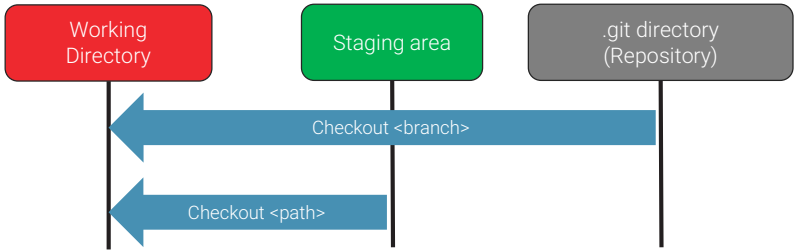
- Create a new branch

```
$ git branch <branch-name> [<start-point>]
```

- Delete a branch

```
$ git branch -d <branch-name>
```

Git



checkout : switch branches
or restore working tree files

- Switch to a branch

```
$ git checkout <branch>
```

- Create a new branch and switch to it

```
$ git checkout -b <new-branch> [<start-point>]
```

- Update given path in working tree to match the index

```
$ git checkout -- <path>   # Local modifications are lost
```

Git

Create a new feature branch

- Now you have a local Git repository which is a copy of *software-automation-architecture/git-training*
- Create a new local branch with a name that respects the convention on your project and switch to it

Git

Create a new feature branch

- Now you have a local Git repository which is a copy of *software-automation-architecture/git-training*
- Create a new local branch with a name that respects the convention on your project and switch to it

```
$ git checkout -b SA2_BLM1

(or $ git branch SA2_BLM1
    $ git checkout SA2_BLM1)
```

Git

Make some changes in working directory

- Make some local changes in existing files
- Create a new file
- Delete an existing file

Git

status : show the working tree status

```
$ git status
```

```
MINGW64 /d/projects/git-training (<branch-name>)
$ git status
On branch <branch-name>
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   src/main/java/com/example/Test.java
        deleted:    src/site/apt/index.apt
        modified:   src/test/java/com/example/TestGreeter.java

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:   src/test/java/com/example/TestGreeter.java
```

Git

diff : show changes between states or commits

- View the diff between working directory and index

```
$ git diff
```

- View the changes you added to the index (staging area)

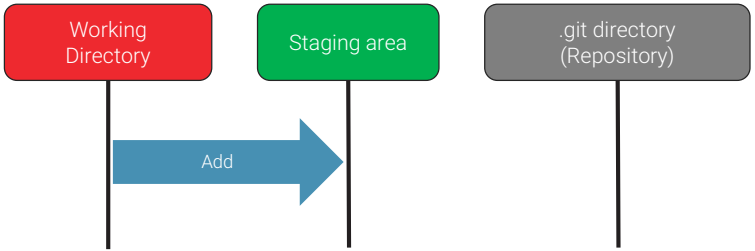
```
$ git diff --staged
```

- View the changes between two commits

```
$ git diff <commit> <commit>
```

Git

add : add file contents to the index



- Interactively choose changes to add to the index

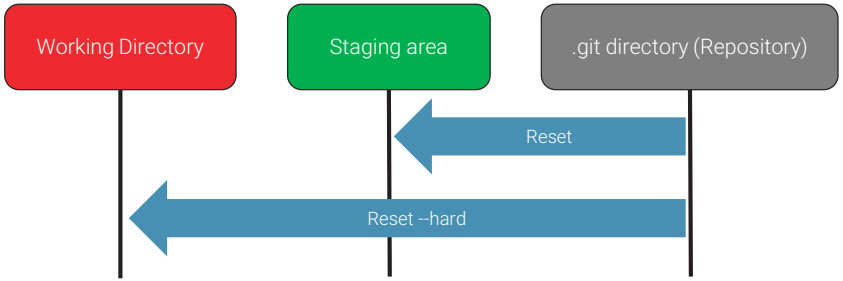
```
$ git add -p
```

- Add all changes at given path to the index

```
$ git add <path>
```

Git

reset : reset current HEAD
to the specified state



- Unstage changes

```
$ git reset HEAD
```

- Interactively choose changes to unstage

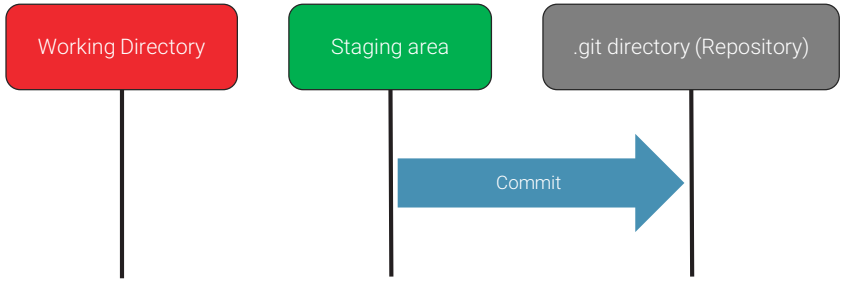
```
$ git reset -p HEAD
```

- Reset current branch head to a commit / branch / tag

```
$ git reset --hard <tree-ish>
```

Git

commit : record changes
to the local repository



- Stores the contents of the index in a new commit

```
$ git commit -m <message>
```

- Amend the previous commit with the contents of the index

```
$ git commit --amend
```

Git

log : show commit logs

- Show the commit logs

```
$ git log
```

- Shows commits and diffs that touch the given path

```
$ git log -p <path>
```

- Shows the commits difference between two branches

```
$ git log <branch1>..<branch2>
```

Git

log : show commit logs

- Visualize the branches, merges, etc.

```
$ git log --graph
```

- Visualize all branches decorated

```
$ git log --graph --oneline --all --decorate
```

- Filter by author

```
$ git log --author=...
```

Git

push : update remote refs along with associated objects

- Update given repository and branch using local branch

```
$ git push <remote> <branch>
```

Git

Add changes to index

- Add new file and deleted file to the index
- Interactively choose files content to add

Git

Add changes to index

- Add new file and deleted file to the index
- Interactively choose files content to add

```
$ git add <new file> <deleted file>
```

```
$ git add -p
```

Git

Unstage some changes

- Unstage **some** of the changes you just added to index

Git

Unstage some changes

- Unstage **some** of the changes you just added to index

```
$ git reset -p HEAD
```

Git

Commit changes

- Commit staged changes with a message

remark: You may have to respect a commit message convention on your project

Git

Commit changes

- Commit staged changes with a message

```
$ git commit -m "Doing something"
```

remark: You may have to respect a commit message convention on your project

Git

Push branch to a remote repository

- Push will create a branch with same name as your local branch on remote repository
- Tip:
  - Push operation takes two parameters **remote** and **branch**

Git

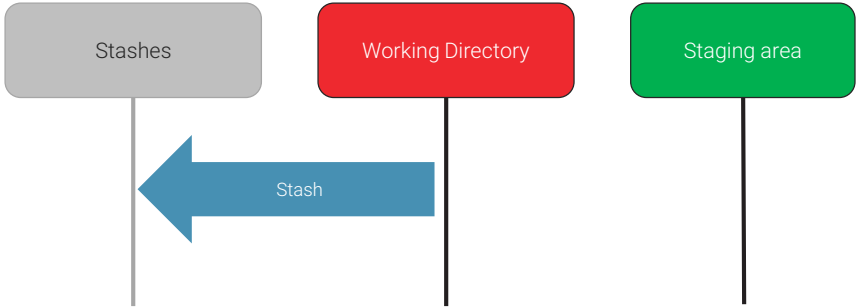
Push branch to a remote repository

- Push will create a branch with same name as your local branch on remote repository

```
$ git push origin <your branch name>
```

Git

stash : stash changes in
working directory away



- stash current state of the working directory and index

```
$ git stash [-p save <message>]
```

- List the stashes that you currently have

```
$ git stash list
```

https://dev.to/srebalaji/useful-tricks-you-might-not-know-about-git-stash-117e

- Apply a stash on top of the current working tree state

```
$ git stash pop [<stash>]
```

Git

Create another feature branch

- Create another branch **from master** (still respecting naming convention) + switch to it
- Tip:
  - Pay attention to use **master** as starting point and not your currently checked out branch

Git

Create another feature branch

- Create another branch **from master** (still respecting naming convention) + switch to it

```
$ git checkout -b SA2_BLM2 master
```

```
[error: Your local changes to the following files would
be overwritten by checkout:
        <your modified file>
Please commit your changes or stash them before you can
switch branches.]
```

Git

Stash your local changes

- Stash
- Check content of your stash

Git

Stash your local changes

- Stash
- Check content of your stash

```
$ git stash

$ git stash list

$ git stash show -p stash@{0}
```

Git

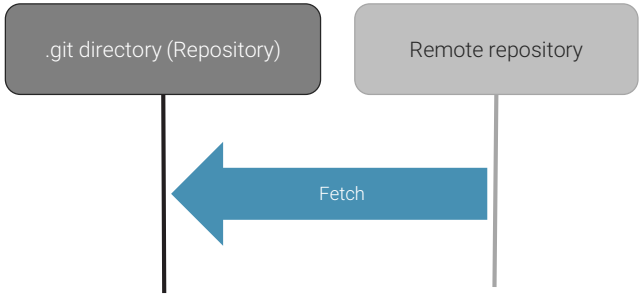
Retry to create your branch

- Now your working directory is clean ☺

```
$ git checkout -b SA2_BLM2 master
```

Git

fetch : download objects
and refs from another repository



- Fetch branches and tags from a given remote repo

```
$ git fetch <remote>
```

- Fetch from all configured remotes

```
$ git fetch --all
```

Git

rebase : apply commits on top of another branch

- Apply commits of current branch on top of given branch

```
$ git rebase <branch>
```

- Edit the list of commits which are about to be rebased

```
$ git rebase -i <branch>
```

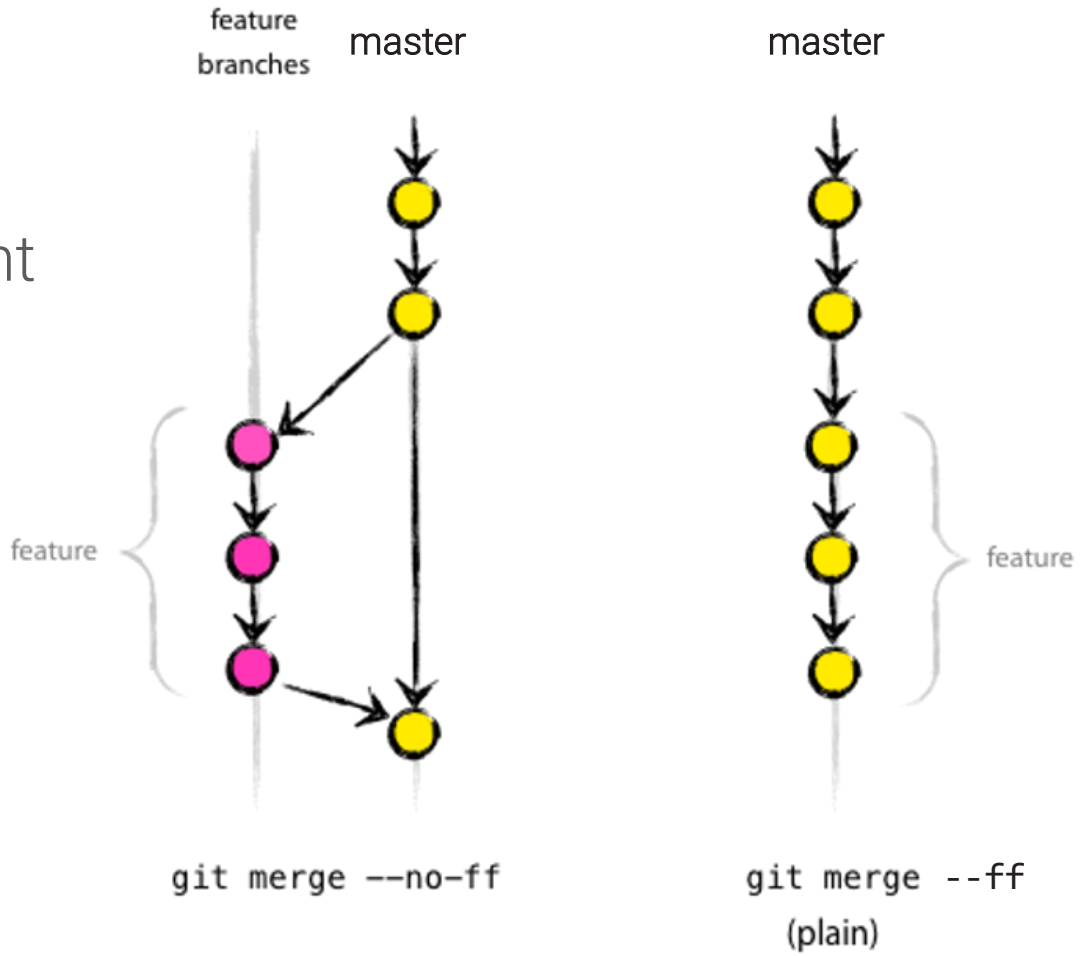
This operation may have consequences for other contributors on the project

Git

merge :
join several development histories together

Two strategies:

- Create a merge commit
- Fast forward



Git

merge : join several development histories together

- Merge branch FEATURE-1 with current branch

```
$ git merge FEATURE-1
```

- Abort the merge process and reconstruct the pre-merge state

```
$ git merge --abort
```

Git

pull : fetch from and integrate with another repository

- Shorthand for *`git fetch && git merge`*

```
$ git pull <remote> <branch>
```

- Shorthand for *`git fetch`* followed by *`git rebase`*

```
$ git pull --rebase <remote> <branch>
```

Recommended way of pulling

- Rebase by default when doing `git pull` without `--rebase`

```
$ git config --global pull.rebase true
```

Git

remote : manage remotes

- List remotes

```
$ git remote [-v]
```

- Add a new remote

```
$ git remote add <alias> <repository URL>
```

- Remove a given remote

```
$ git remote remove <alias>
```

Git

Apply changes from another branch

- Apply the changes from your first branch on the second branch using the **rebase** strategy
- Tip:
  - You can use either the local or the remote branch to get the changes

Git

Apply changes from another branch

- Apply the changes from your first branch on the second branch using the **rebase** strategy

```
$ git rebase <branch>
```

```
(or $ git pull --rebase origin <branch>)
```

Git

Unstash your local changes

- Unstash the local changes that we staged earlier

Git

Unstash your local changes

- Unstash the local changes that we staged earlier

```
$ git stash pop
```

```
or $ git stash apply [stash{0}]
```

Git

Clean your local changes

- Clean your local changes from the working directory
- Tip:
  - Revert changes on working directory = update given paths in the working tree from the index file

Git

Clean your local changes

- Clean your local changes from the working directory

```
$ git checkout -- <path>
```

Git

revert : Revert some existing commits

- Revert the changes that given commit(s) introduced

```
$ git revert <commit>..
```

- Edit the commit message prior to committing the revert

```
$ git revert -e <commit>..
```

Git

A bit further

sopra banking SOFTWARE

Git Aliases

Aliases are shortcuts to Git commands

- You can define some useful aliases

```
$ git config --global alias.st status
$ git config --global alias.ci commit
$ git config --global alias.lg "log --graph --oneline --
all --decorate"
$ git config --global alias.oops "commit --amend --no-
edit"
```

- Using one of them

```
$ git lg
```

Cherry picking

Just pick a single commit

- Pick a commit

```
$ git cherry-pick <commit-sha>
```

- If it goes wrong you can abort cherry pick

```
$ git cherry-pick --abort
```

Cherry picking

```
$ git log --graph --oneline --all --decorate
* 9a050ca (evrignaud) After some work
* 44e18fb (master) Modif #2
* b736445 Modified two files
* f46a1cd (HEAD -> dev) Add some content
* 1db511c Add readme

$ git cherry-pick b736445
[dev e87b4a5] Modified two files
 Date: Wed Jan 31 22:52:20 2018 +0100
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 src/site/apt/index.apt
 create mode 100644 src/test/java/com/example/TestGreeter.java

$ git log --graph --oneline --all --decorate
* e87b4a5 (HEAD -> dev) Modified two files
| * 9a050ca (evrignaud) After some work
| * 44e18fb (master) Modif #2
| * b736445 Modified two files
|/
* f46a1cd Add some content
* 1db511c Add readme
```

Cleanup your history

- Rewriting History
  - Changing Multiple Commit Messages
  - Reordering Commits
  - Squashing Commits
  - Splitting a Commit

For example using:

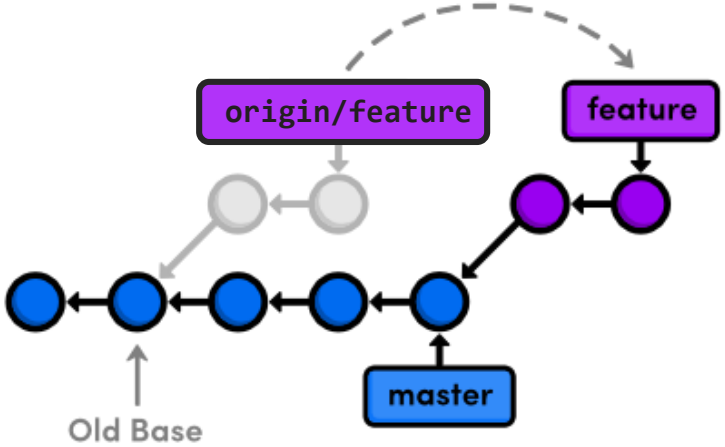
```
$ git rebase -i HEAD~3
```

- https://git-scm.com/book/en/v2/Git-Tools-Rewriting-History
- https://delicious-insights.com/en/posts/getting-solid-at-git-rebase-vs-merge/

Merge Conflicts

- Merge conflicts may occur if competing changes are made to the same line of a file or when a file is deleted that another person is attempting to edit.
  - https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging#_basic_merge_conflicts
  - https://help.github.com/articles/resolving-a-merge-conflict-using-the-command-line/
- Using GitLab's Merge Request you don't merge manually. To ease GitLab's Merge you need to rebase on the master's HEAD.

Force push needed after rebase



- After a rebase the remote branch cannot be « fast-forwarded » to your local branch
  - https://stackoverflow.com/a/8940299
  - https://stackoverflow.com/a/15144275

Git

More exercises

1. Display commits difference between a local branch and origin/master
2. Reset HEAD to previous commit
3. Apply **one** commit from another branch to your branch
4. Edit last commit (content, message and author)
5. Delete a branch both locally and on remote repository

Git

More exercises

1. `git log origin/master..<my-branch>`
2. `git reset [mode] HEAD~1`
3. `git cherry-pick <commit SHA>`
4. `git commit --amend [--author “Author <a@a.a>”]`
5. `git branch -d (or -D) <branch>`
   `&& git push <remote> :<branch>`

Going further

- Learn Git Branching
  https://learngitbranching.js.org/
- Git book available online for free
  - EN https://git-scm.com/book
  - FR https://git-scm.com/book/fr/v2
- Introduction to Git with Scott Chacon of GitHub
  https://www.youtube.com/watch?v=ZDR433b0HJY
- Git GUI Clients
  https://git-scm.com/download/gui/windows

.gitconfig minimal content

```
# File: ~/.gitconfig
[core]
        longpaths = true
        autocrlf = true
        excludesfile = D:/Profiles/<username>/.gitignore
        fscache = true
[push]
        default = matching
[user]
        name = <name>
        email = <email>
[pull]
        rebase = true
[merge]
        ff = false
[credential]
        helper = manager
[help]
        autocorrect = 1
```

Details here:
https://git-scm.com/docs/git-config

```
[alias]
        st = status
        ci = commit
        oops = commit --amend --no-edit
        lg = log --graph --oneline --all --decorate
```

THINKING AHEAD BEGINS NOW.

Thank you.