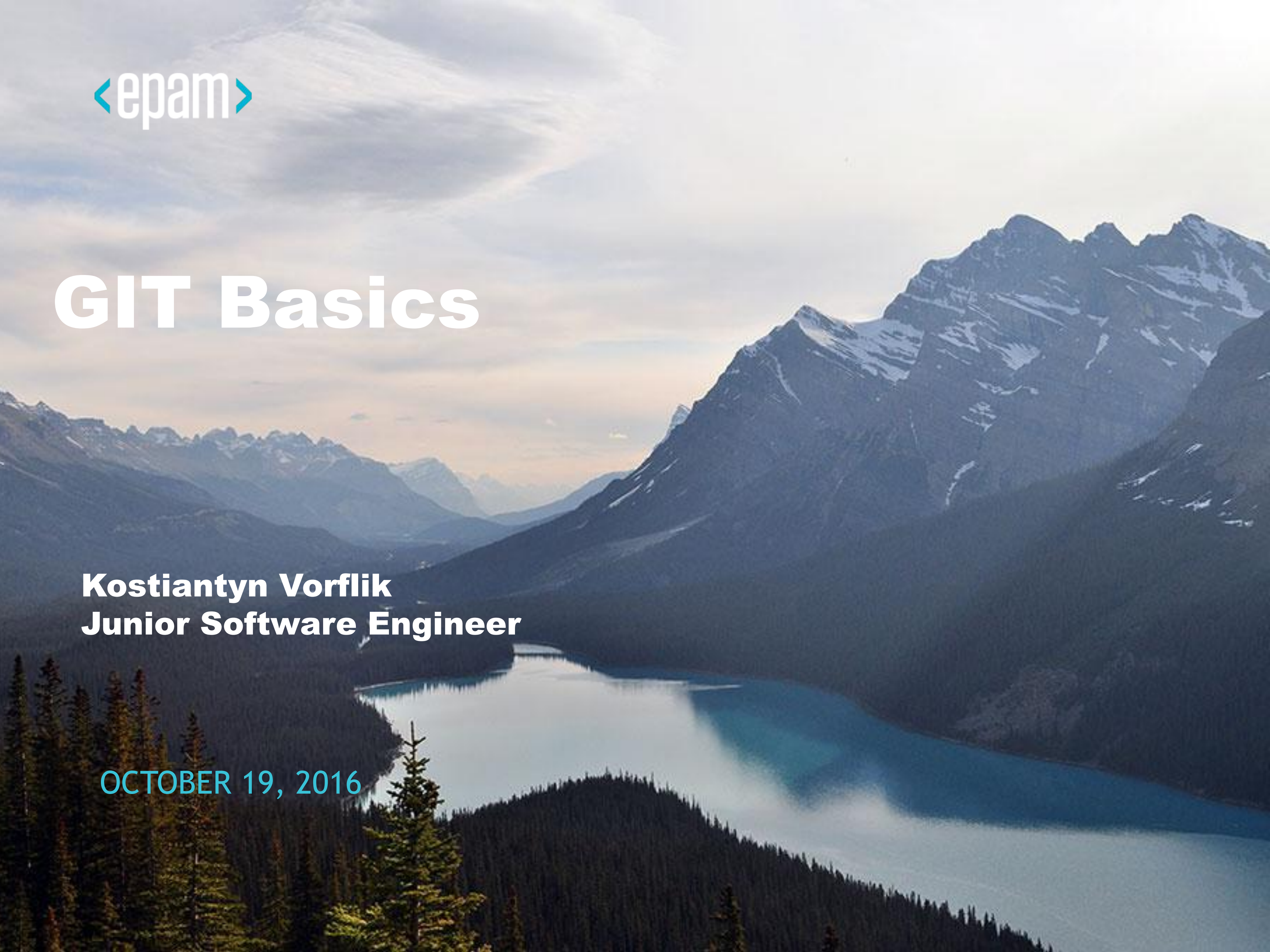


GIT Basics

Kostiantyn Vorflik
Junior Software Engineer

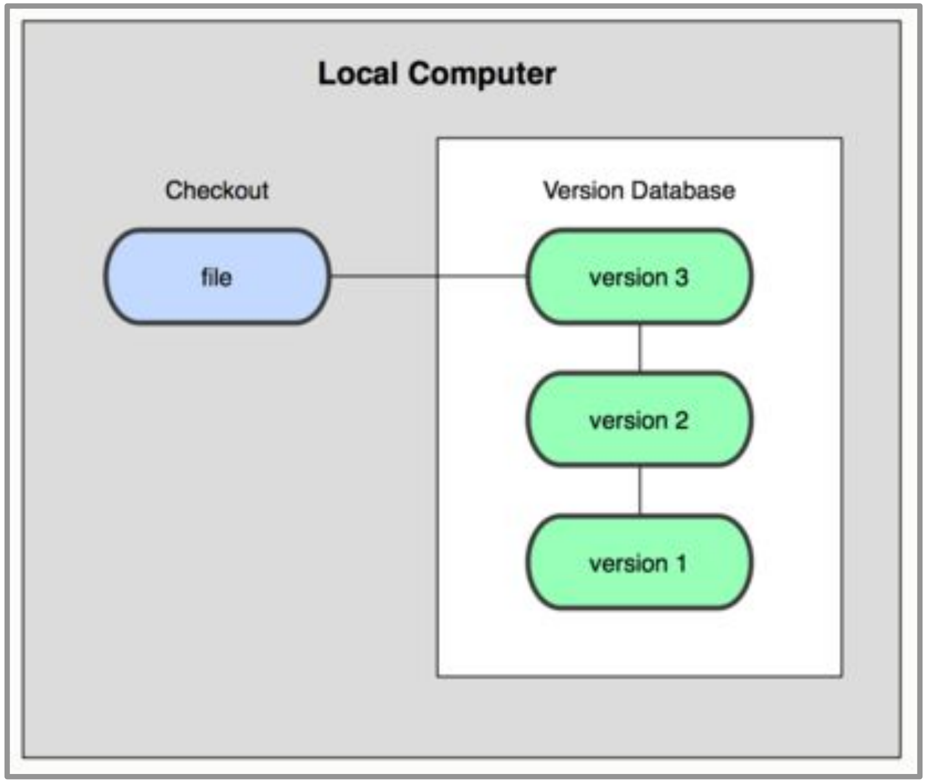
OCTOBER 19, 2016

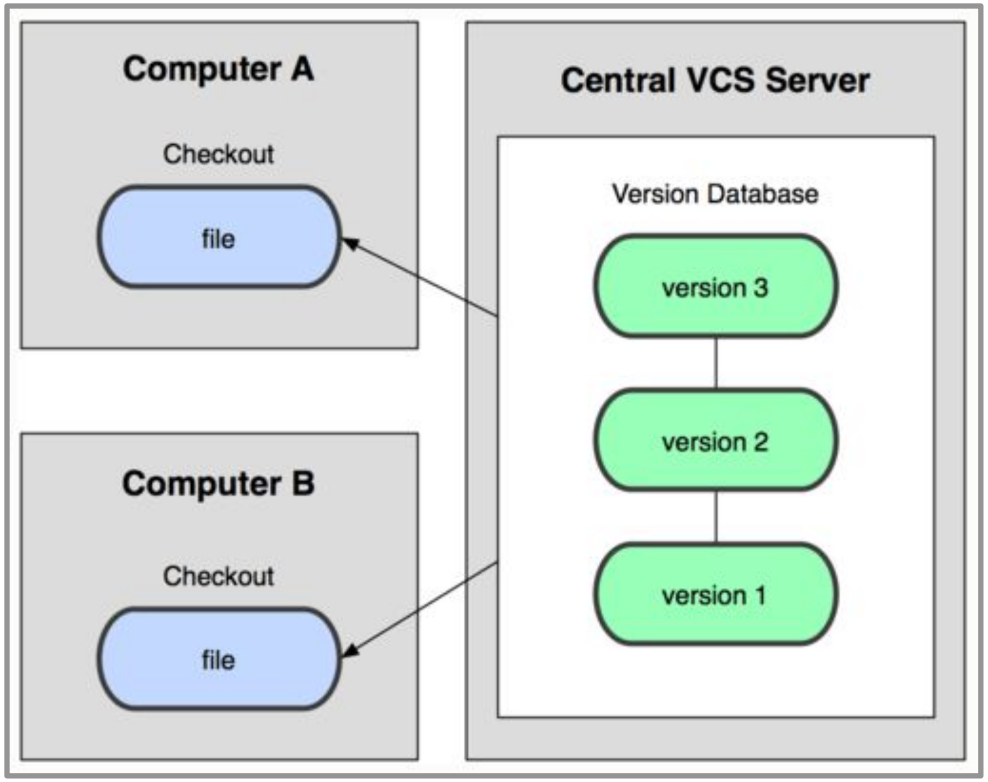
Agenda

1. What it VCS and why it is useful to use it?
2. Distributed VS Centralized VSC. Prof and cons.
3. Installing GIT
4. Gitlab
5. Git under the bonnet
6. Git basics

PART I
ABOUT GIT

What it VCS and why it is useful to use it?

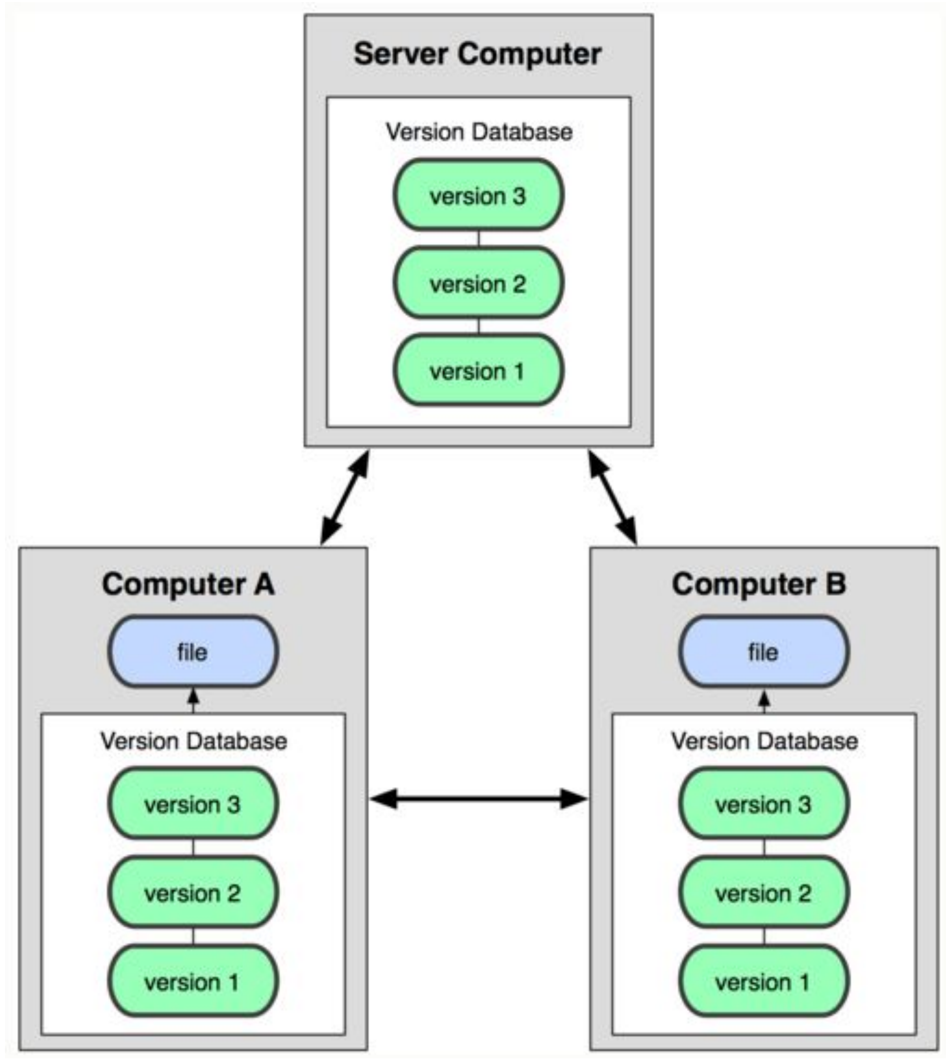


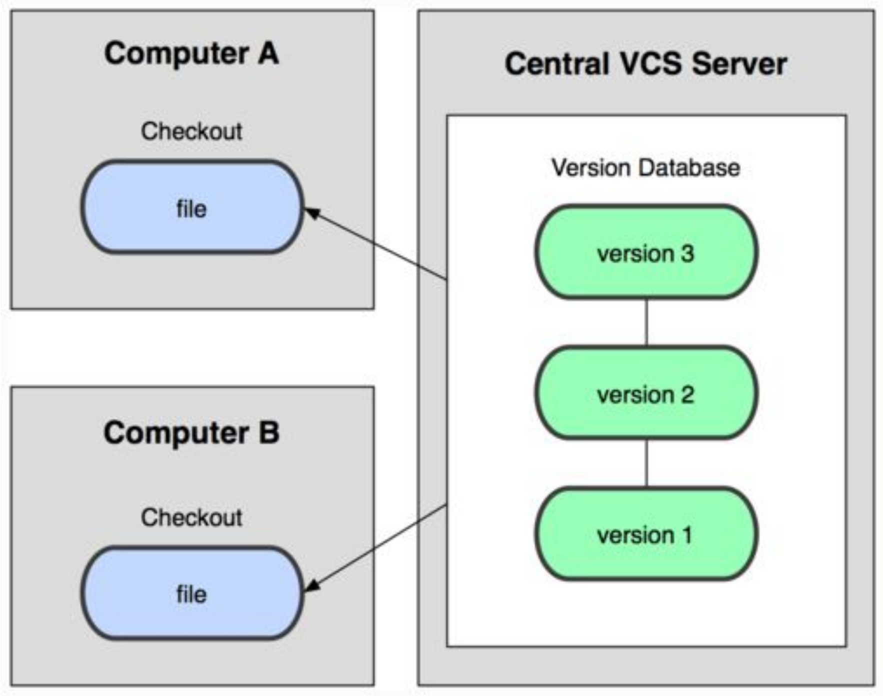


Advantages of using VCS

1. Collaboration
2. Storing Versions (Branching)
3. Restoring Previous Versions
4. Understanding What Happened
5. Backup

Distributed VS Centralized VSC





Distributed VS Centralized VSC

Advantages of distributed VCS

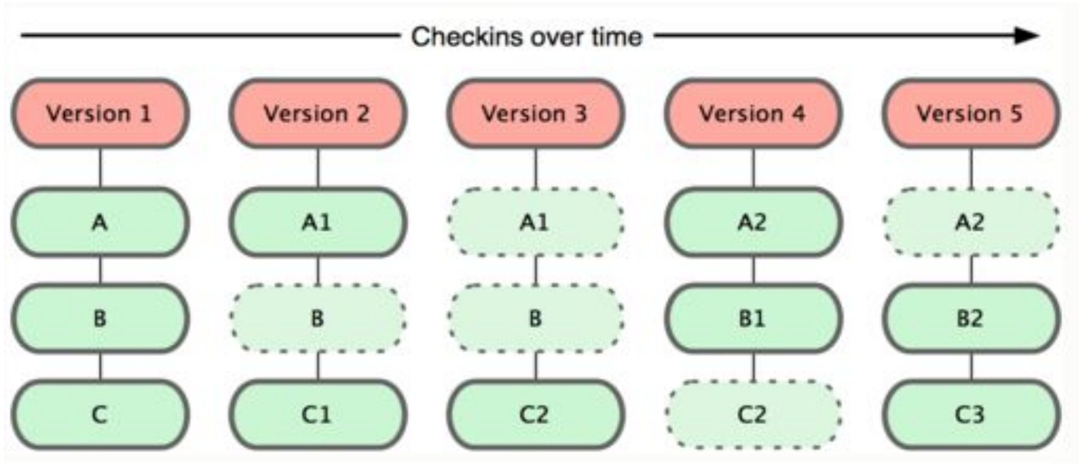
- Most of operations are local.
- Repository data and history available on each local copy, so you could do a lot of operation without internet connection.
- If central copy of data will be lost, any local copy could be used to restore central.
- Lightweight branching.
- Possibility of working with several remotes in one time.

Advantages of centralized VCS

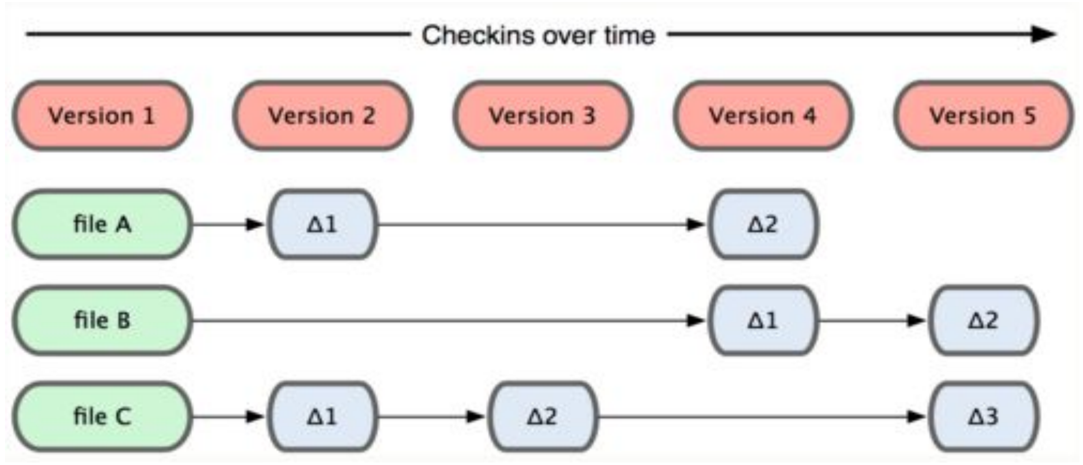
- Storing only current copy of data in a local repository could be an advantage.
- Easier workflow for novice users.

Distributed VS Centralized VSC

Distributed VCS stores patches



Centralized VCS stores stream of snapshots



Installing GIT

Linux

- Via binary installer:

```
$ sudo yum install git-all
```

- If you're on a Debian-based distribution like Ubuntu, try:

```
$ sudo apt-get install git-all
```

Windows

- Just go to the next link and the download will start automatically.

```
http://git-scm.com/download/win
```

Other

- To find more ways to download and install git visit:

```
https://git-scm.com/downloads
```

GIT configuration & help

`git config` Saves configuration for current repository

`--system` (Saves configuration for all system users)

`--global` (Saves configuration for current system user)

- `git config --global user.name “Ivan Ivanov"` (To set user name)
- `git config --global user.email ivan_ivanov@epam.com` (To set user email)
- Setup Notepad++ as core editor

```
git config --global core.editor "'C:/Program Files (x86)/Notepad++/notepad++.exe' -multiInst
-notabbar -nosession -noPlugin"
```

- `git config --list` (To get current configuration)

`git help`

- `git help <verb>`
- `git <verb> --help`
- `man git-<verb>`

GIT configuration & help

`git config` Saves configuration for current repository

`--system` (Saves configuration for all system users)

`--global` (Saves configuration for current system user)

- `git config --global user.name “Ivan Ivanov"` (To set user name)
- `git config --global user.email ivan_ivanov@epam.com` (To set user email)
- Setup Notepad++ as core editor

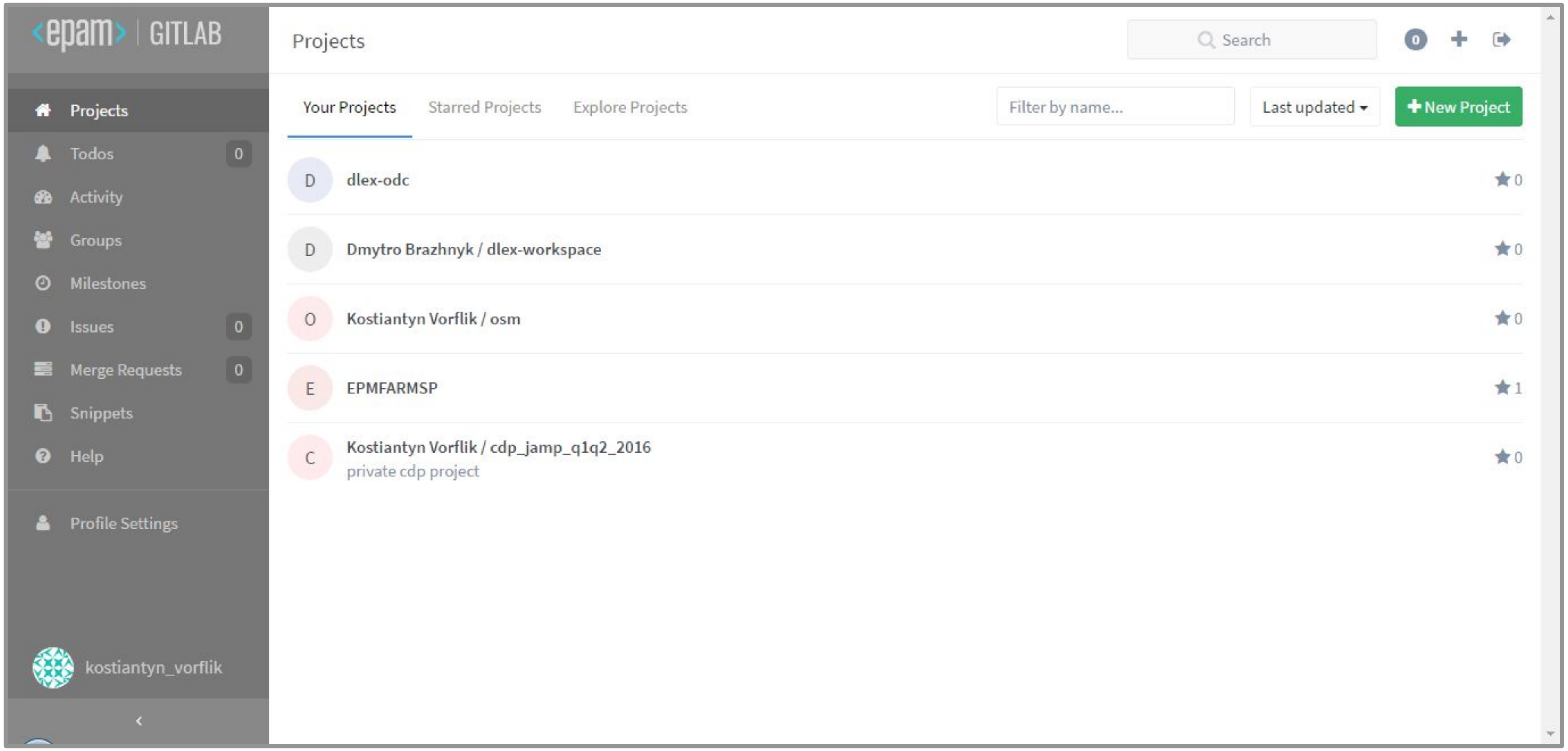
```
git config --global core.editor "'C:/Program Files (x86)/Notepad++/notepad++.exe' -multiInst
-notabbar -nosession -noPlugin"
```

- `git config --list` (To get current configuration)

`git help`

- `git help <verb>`
- `git <verb> --help`
- `man git-<verb>`

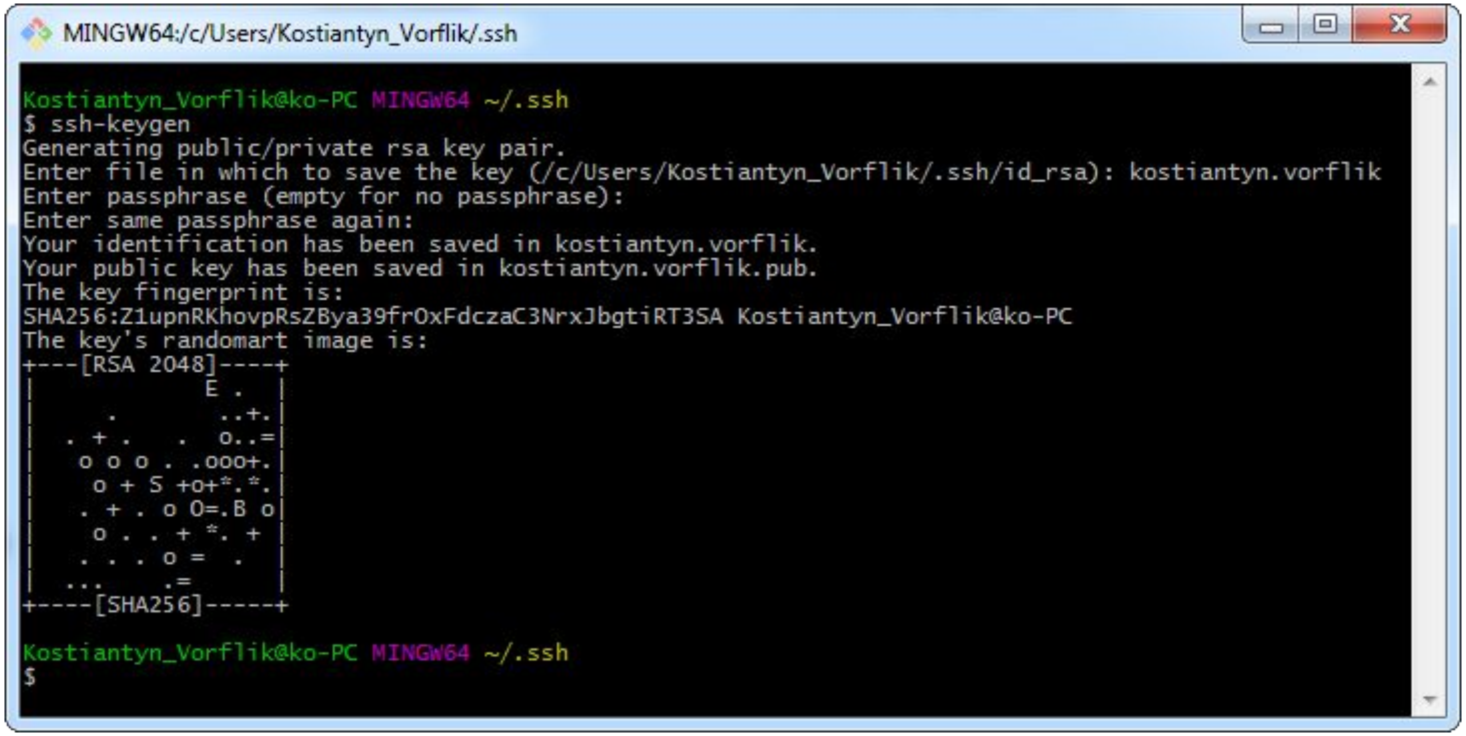
Gitlab – internal EPAM repository



D	dlex-odc	0
D	Dmytro Brazhnyk / dlex-workspace	0
O	Kostiantyn Vorflik / osm	0
E	EPMFARMSP	1
C	Kostiantyn Vorflik / cdp_jamp_q1q2_2016 private cdp project	0

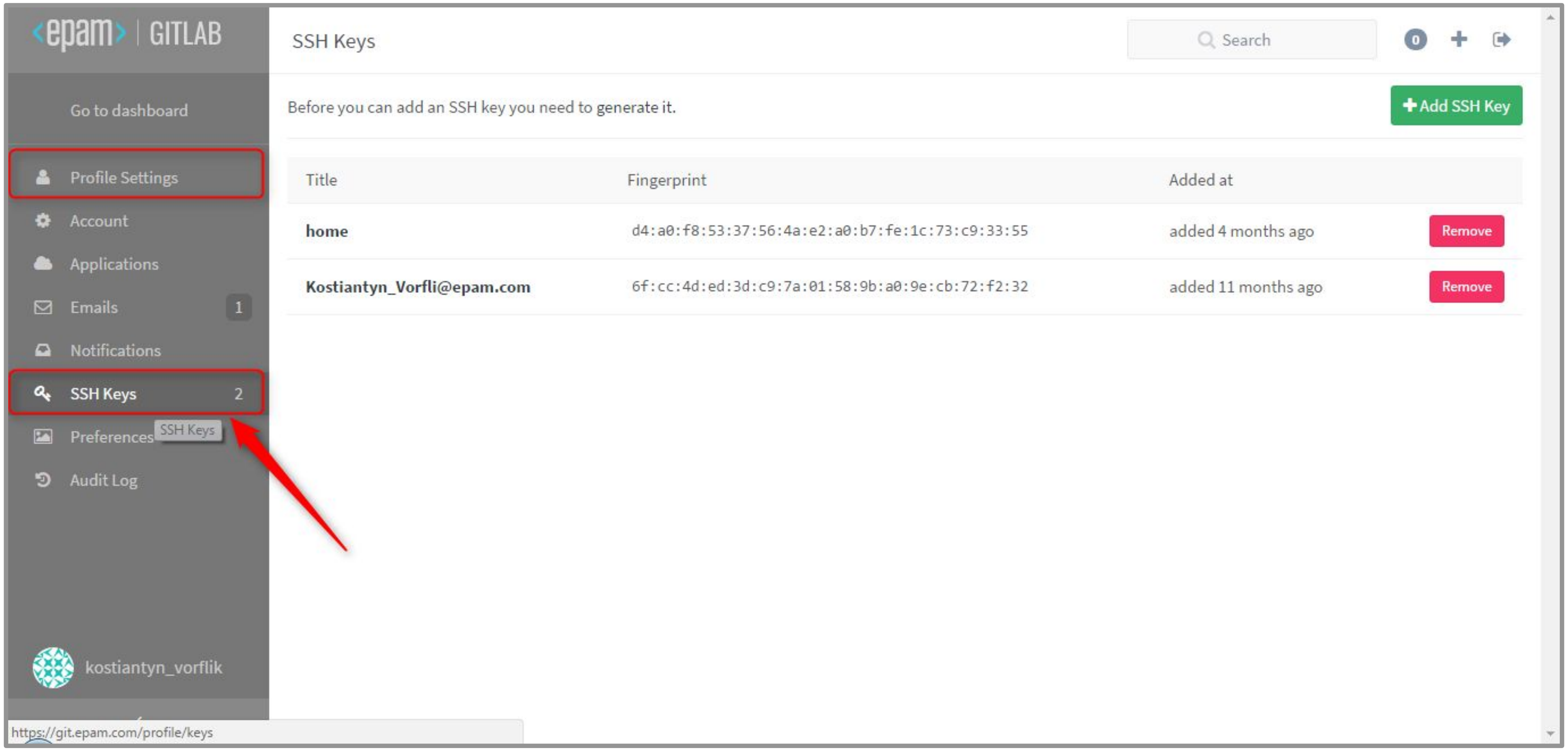
Generate new ssh key

1. Set your email and username in you Git client.
2. Generate a new SSH private/public key-set.
3. Add your public key to Gitlab


MINGW64:/c/Users/Kostiantyn_Vorflik/.ssh

```
Kostiantyn_Vorflik@ko-PC MINGW64 ~/.ssh
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/Kostiantyn_Vorflik/.ssh/id_rsa): kostiantyn.vorflik
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in kostiantyn.vorflik.
Your public key has been saved in kostiantyn.vorflik.pub.
The key fingerprint is:
SHA256:Z1upnRKhovpRsZBya39frOxFdczaC3NrxJbgtiRT3SA Kostiantyn_Vorflik@ko-PC
The key's randomart image is:
+---[RSA 2048]----+
|            E .  |
|      .      ..+.|
| . + .    .  o..=|
|   o o o . .ooo+.|
|    o + S +o+*.*.|
|   . + . o O=.B o|
|    o . . + *. + |
|   . . . o =  .  |
|  ...    .=      |
+----[SHA256]-----+

Kostiantyn_Vorflik@ko-PC MINGW64 ~/.ssh
$
```

Integrate new ssh key with Gitlab



SSH Keys

Before you can add an SSH key you need to generate it.

Title	Fingerprint	Added at	
home	d4:a0:f8:53:37:56:4a:e2:a0:b7:fe:1c:73:c9:33:55	added 4 months ago	Remove
Kostiantyn_Vorfli@epam.com	6f:cc:4d:ed:3d:c9:7a:01:58:9b:a0:9e:cb:72:f2:32	added 11 months ago	Remove

ssh-key sample

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCrLMjgTwIO/uFRom47o2oMWYiFxIRa+nrsjQ2n9W4Tft9hW0wDGXa
9AYN/MAWEMD6FzGxLvkHy9vwHChQbKPXAwwTGAmpp7RenJ8ukGczVEY00K8nlfZ6qS5unxcFtR4/C2NJGvx
OCYYJEac+1Lpxwk02ZXX4TwARKHgl+oNlE6KoAHG6tDBYdvxH981alxp+aqyhZs5RNRTECRJujwjNcjTwFayn
G5LlfRwUjI+UtWvD70fQj4u/TE7Rfi+sNyBblJTnJYjkzgppseF5vttQsBvLWISthmUDizfKh1FXJ+g7AjS3tLztBX1
8Qw3tLkck+1iz/Er5HbclsboBIH9tB Kostiantyn_Vorflik@ko-PC
```

PART II
GIT BASICS

.gitignore

This is a file, which you could create in the root of your repository. All files, which are match patterns from gitignore, would be untracked by default. This could be binary files; files, which are generated by IDE, logs, ect. So all of this files exist in you project directory, but you will never want to commit them to repository.

The rules for the patterns you can put in the .gitignore file are as follows:

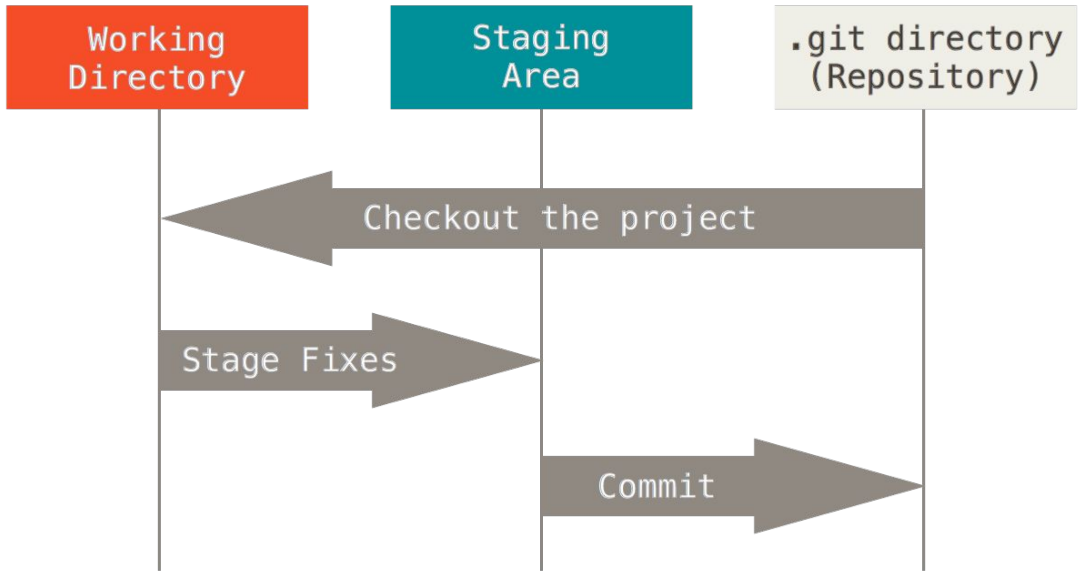
- Blank lines or lines starting with # are ignored.
- Standard glob patterns work.
- You can start patterns with a forward slash (/) to avoid recursivity.
- You can end patterns with a forward slash (/) to specify a directory.
- You can negate a pattern by starting it with an exclamation point (!).

```
# no .a files
*.a
# but do track lib.a, even though you're ignoring .a files above
!lib.a
# ignore all files in the build/ directory
build/
# ignore all .pdf files in the doc/ directory
doc/**/*.pdf
```

The three states. The basic GIT workflow

- **Modified:** you have changed the file but have not committed it to your local database
- **Staged:** you have marked a modified file in its current version to go into your next commit snapshot.
- **Committed:** the data is safely stored in your local database.

This leads us to the three main sections of a GIT project:



Creating GIT repository

Initialization

```
git init
```

This command is used for putting existing project under version control. Command should be executed in the root project directory. *Pay attention*! After invoking this command you files will be **untracked**. You should track them and do initial commit [illegible]

Clone

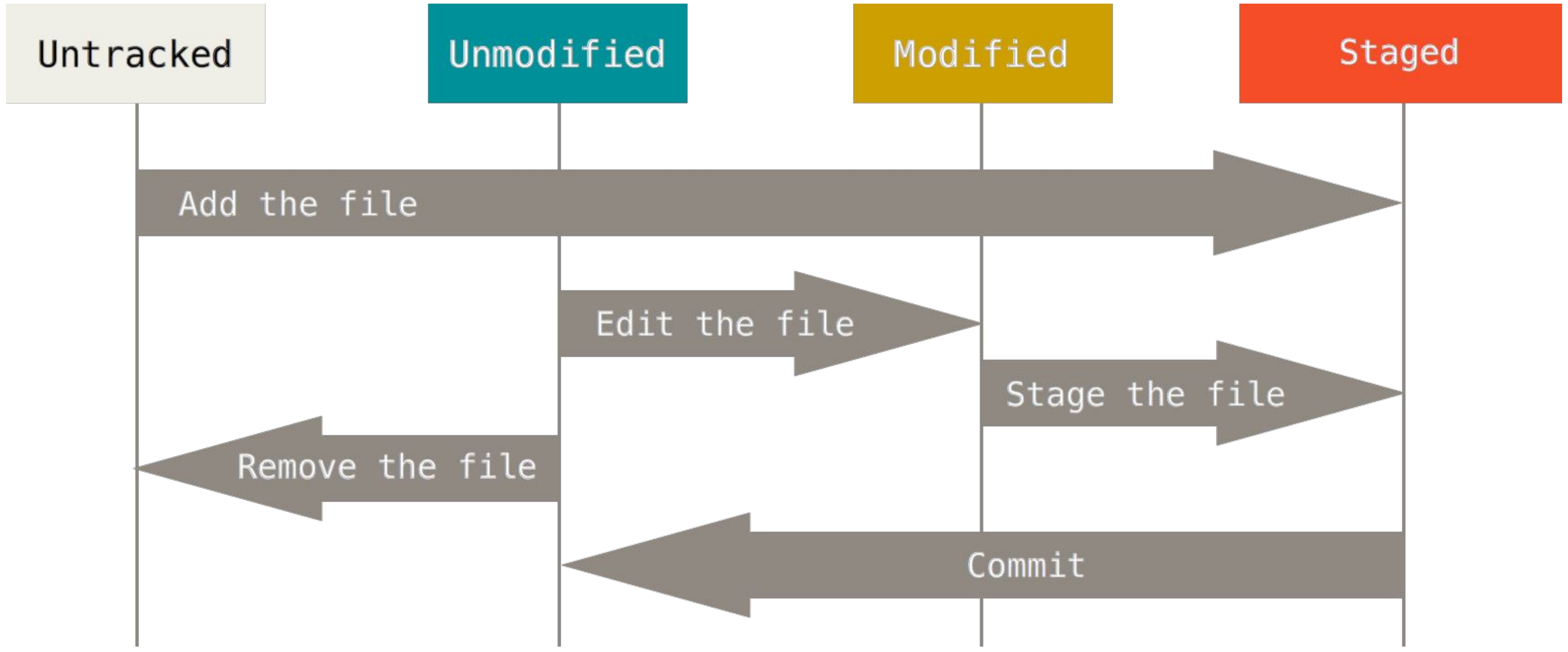
```
git clone [url]
```

This command is used to clone remote repository and create local copy for you. After cloning repository all files are in **unmodified** state.

For cloning repository you could use different transfer protocols. For example: https, ssh.

File state lifecycle. GIT status

Lifecycle



Status

```
git status
```

This command is used to find out in which states you repository files are.

GIT add

```
git add [file]
```

Command git add is used for the different proposes. Two of them are:

- Put untracked file under VCS, prepare them for commit. **[untracked -> staged]**

```
On branch master
Untracked files:
 (use "git add <file>..." to include in what will be committed)
 README
nothing added to commit but untracked files present (use "git add" to track)
```

- Prepare modified files for commit. **[modified -> staged]**

```
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: CONTRIBUTING.md
```

GIT add

After using

`git add *`

or

`git add README`

`git add CONTRIBUTING.md`

we will get the next result:

```
$ git status
On branch master
Changes to be committed:
 (use "git reset HEAD <file>..." to unstage)
 new file: README
 modified: CONTRIBUTING.md
```

GIT add

What will happened if we do some changes in README file?

```
vim CONTRIBUTING.md
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
    new file:   README
    modified:   CONTRIBUTING.md
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:   CONTRIBUTING.md
```

Git stages a file exactly as it is when you run the git add command.

Committing changes

The command

`git commit`

allows you to fix your **staged changes**.

```
$ git commit -m "Story 2: Extending readme files"
[master 463dc4f] Story 2: Extending readme files
 2 files changed, 2 insertions(+)
 create mode 100644 README
```

You could also use

`git commit –a`

to skip staging area.

Deleting & moving files

Deleting

```
$ rm PROJECTS.md
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working
directory)

        deleted:    PROJECTS.md

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

`git rm [file]` allows you to **stage** files, which should be deleted.

```
rm 'PROJECTS.md'
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD &lt;file&gt;..." to unstage)

    deleted:    PROJECTS.md
```

Deleting & moving files

Moving and renaming files

```
git mv [source][dest]
```

```
$ git mv README.md README
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    renamed:    README.md -> README
```

Reviewing commit history

`git log`

The command for reviewing commit history. By default shows SHA-1, commit name, author, email, date.

Some of the most popular options:

Option	Description
-p	Shows the difference between commits
-2	Limits number of commits
--pretty[value]	Changes the view of output. Possible values: oneline, short, full, fuller, format
-- graph	Shows the graph with current branch and merging history

```
$ git log --pretty=oneline -1
ca82a6dff817ec66f44342007202690a93763949 changed the version number
$ git log --pretty=format:"%h - %an, %ar : %s" -1
ca82a6d - Scott Chacon, 6 years ago : changed the version number
```

Reverting local changes

```
git commit --amend
```

This command allows you to make some changes in your last commit.

```
git reset HEAD [file]
```

To unstaging a staged file. Git status will help you:

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    renamed:    README.md -> README
    modified:   CONTRIBUTING.md
```

```
git checkout --[file]
```

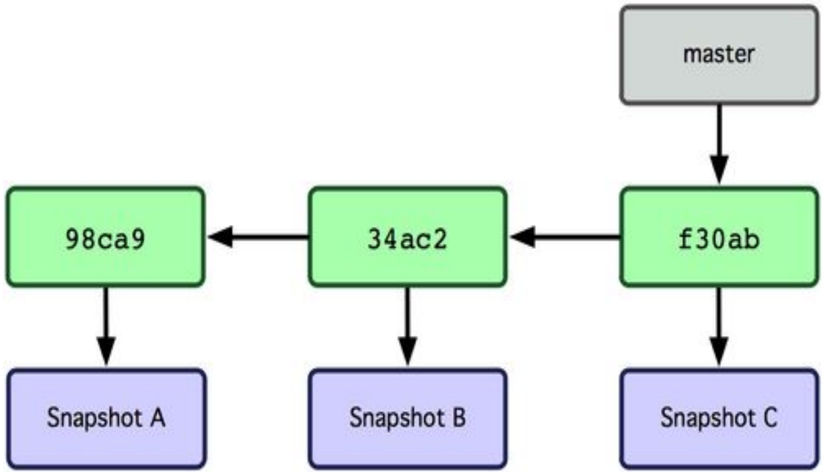
Unmodifying a modified file. Git status will help you again:

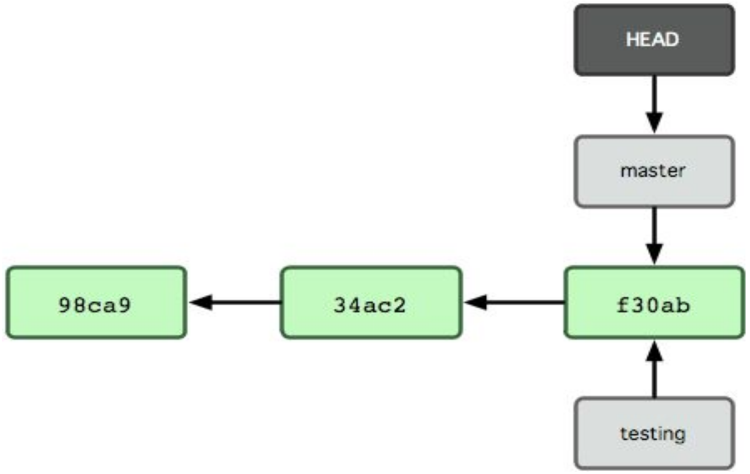
```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   CONTRIBUTING.md
```

Git Branching

What branch is?

A branch in Git is simply a lightweight movable pointer to one of commits.





Creating new branch

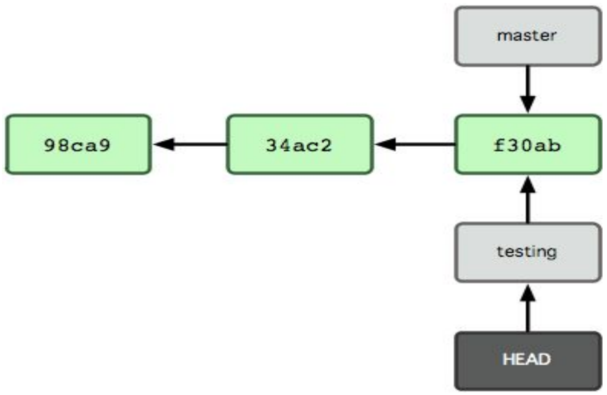
`git branch [name]` Only creates a branch, does not switch on it.

HEAD a special pointer, which allows GIT to know what branch you're currently on.

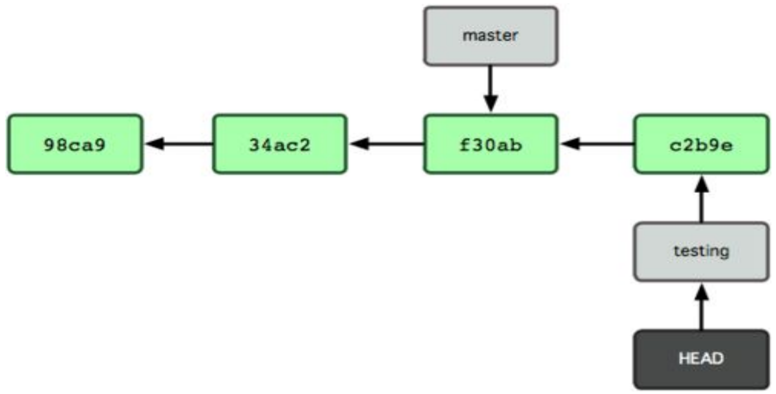
Git Branching: Example

Switch to another branch

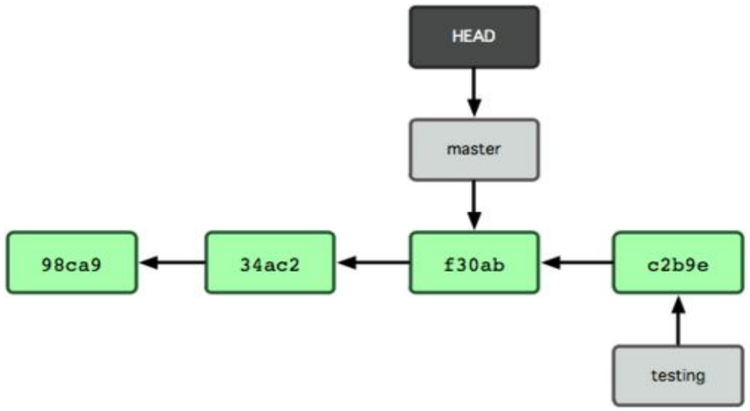
```
git checkout -b testing
```



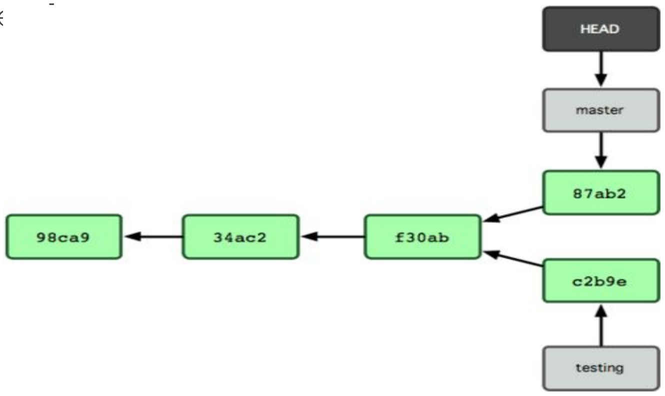
```
[change something]
git commit -a -m 'made a change'
```



```
git checkout master
```

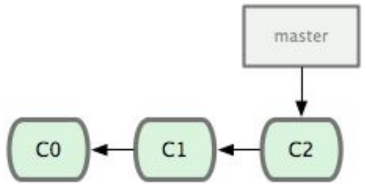


```
[made another changes]
git commit -a -m 'made other
chang
```

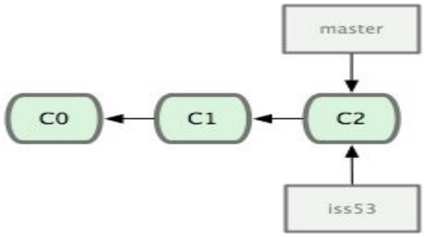


Branching & merging workflow

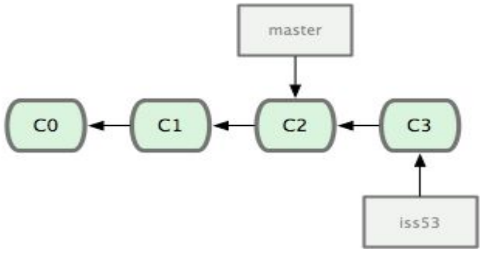
Possible git workflow



```
$ git checkout -b iss53
Switched to a new branch 'iss53'
```

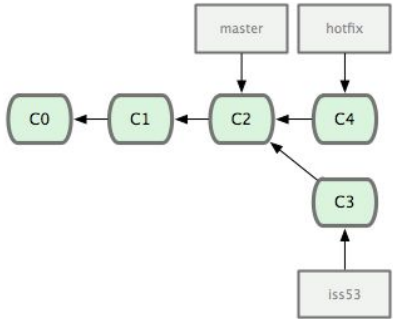


```
[working on iss53]
$ git commit -a -m ‘issue53 add footer'
```

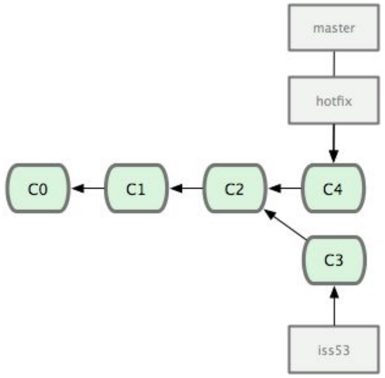


```
$ git checkout master
Switched to branch 'master'
```

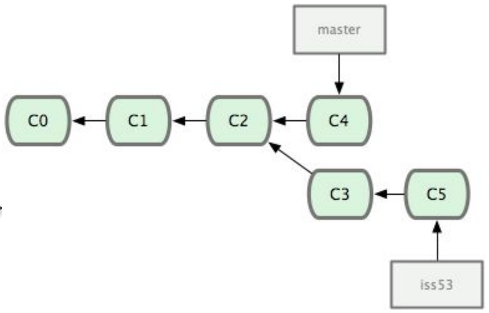
```
$ git checkout -b hotfix
Switched to a new branch 'hotfix‘
[do some fixes]
$ git commit -a -m 'fix something'
```



```
$ git checkout master
$ git merge hotfix
Updating f42c576..3a0874c
Fast-forward
```



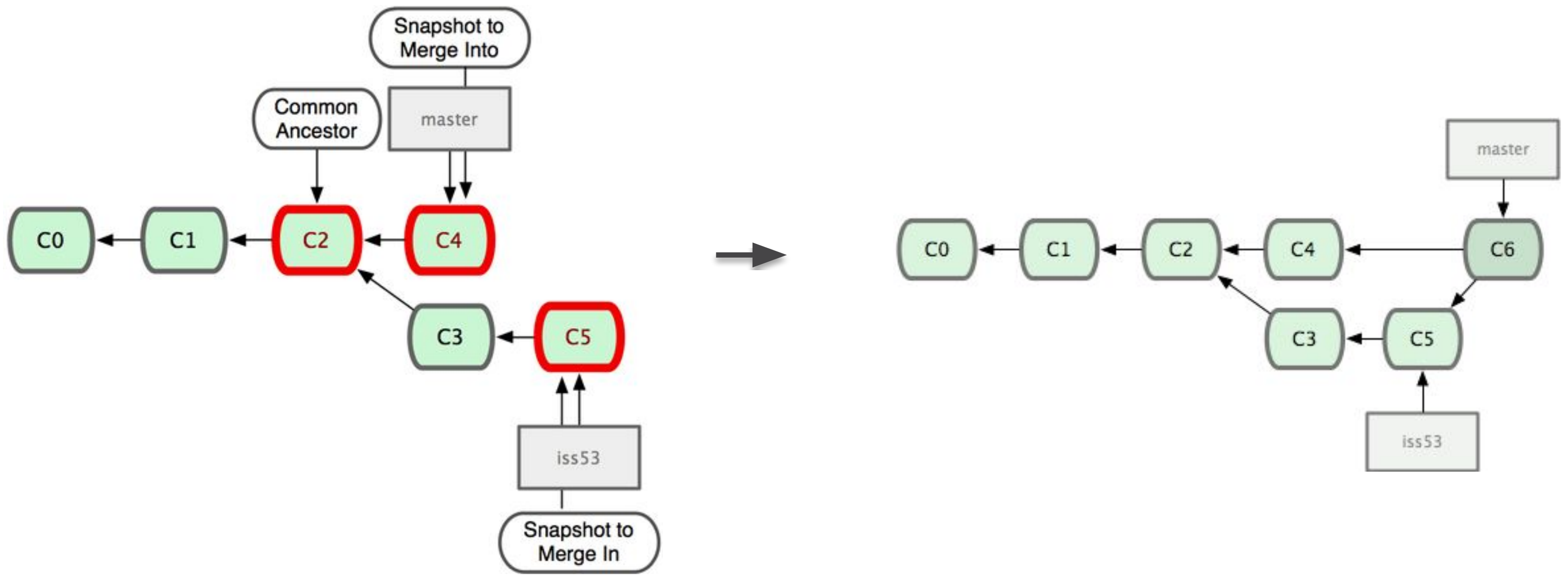
```
$ git branch -d hotfix
Deleted branch hotfix (was 3a0874c).
$ git checkout iss53
Switched to branch 'iss53'
[Finish working on iss53]
$ git commit -a -m 'finish [issue 53]'
```



git merge Join two or more development histories together

Basic merging

```
$ git checkout master
$ git merge iss53
Auto-merging README
Merge made by the 'recursive' strategy.
```



Merge conflicts

```
$ git merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

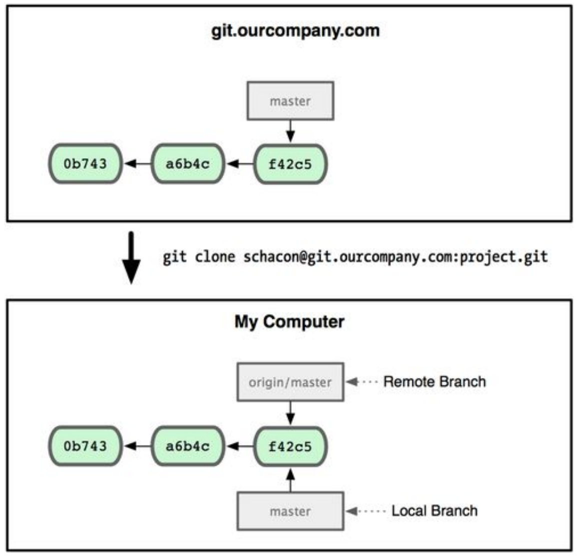
Git hasn't automatically created a new merge commit. It has paused the process while you resolve the conflict. If you want to see which files are unmerged at any point after a merge conflict, you can run `git status`:

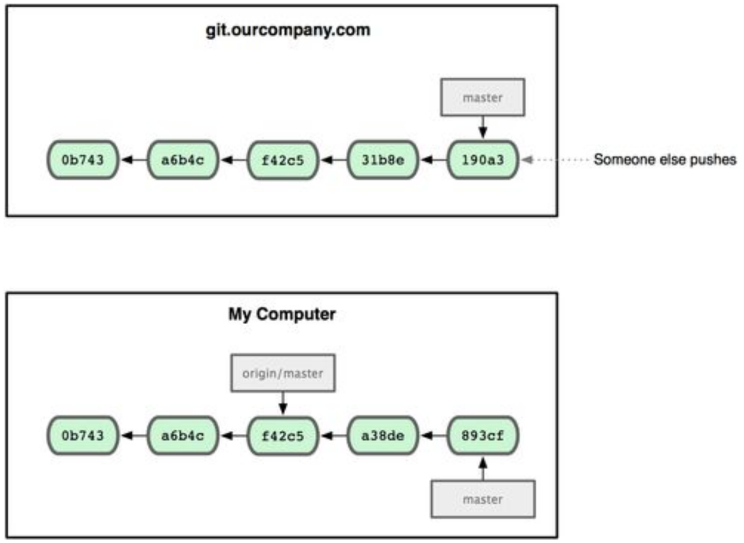
```
$ git status
On branch master
You have unmerged paths.
     (fix conflicts and run "git commit")
Unmerged paths:
     (use "git add <file>..." to mark resolution) both modified:
     index.html
no changes added to commit (use "git add" and/or "git commit -a")
```

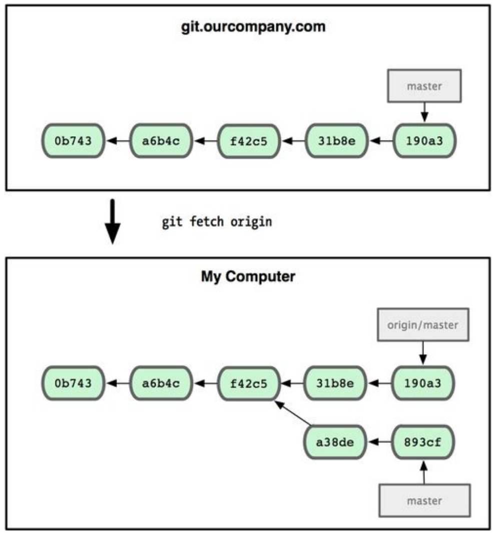
`git mergetool` Run an appropriate visual merge tool

After merging you should **add to index** and **commit** the changes.

Remote and local branches







Remote branches

Pushing branch to remote

```
git push (remote) (branch)
```

```
$ git push origin serverfix
...
* [new branch] serverfix -> serverfix
```

`git push origin serverfix:newname` to give remote branch another name

Fetching / pulling remote branches

Someone else do:

```
$ git fetch origin
...
* [new branch] serverfix ->
origin/serverfix
```

Local branch is not created.

`$ git checkout -b serverfix origin/serverfix` to get a local copy of remote branch

Deleting remote branch

```
git push [remotename] :[branch]
```

Git reflog

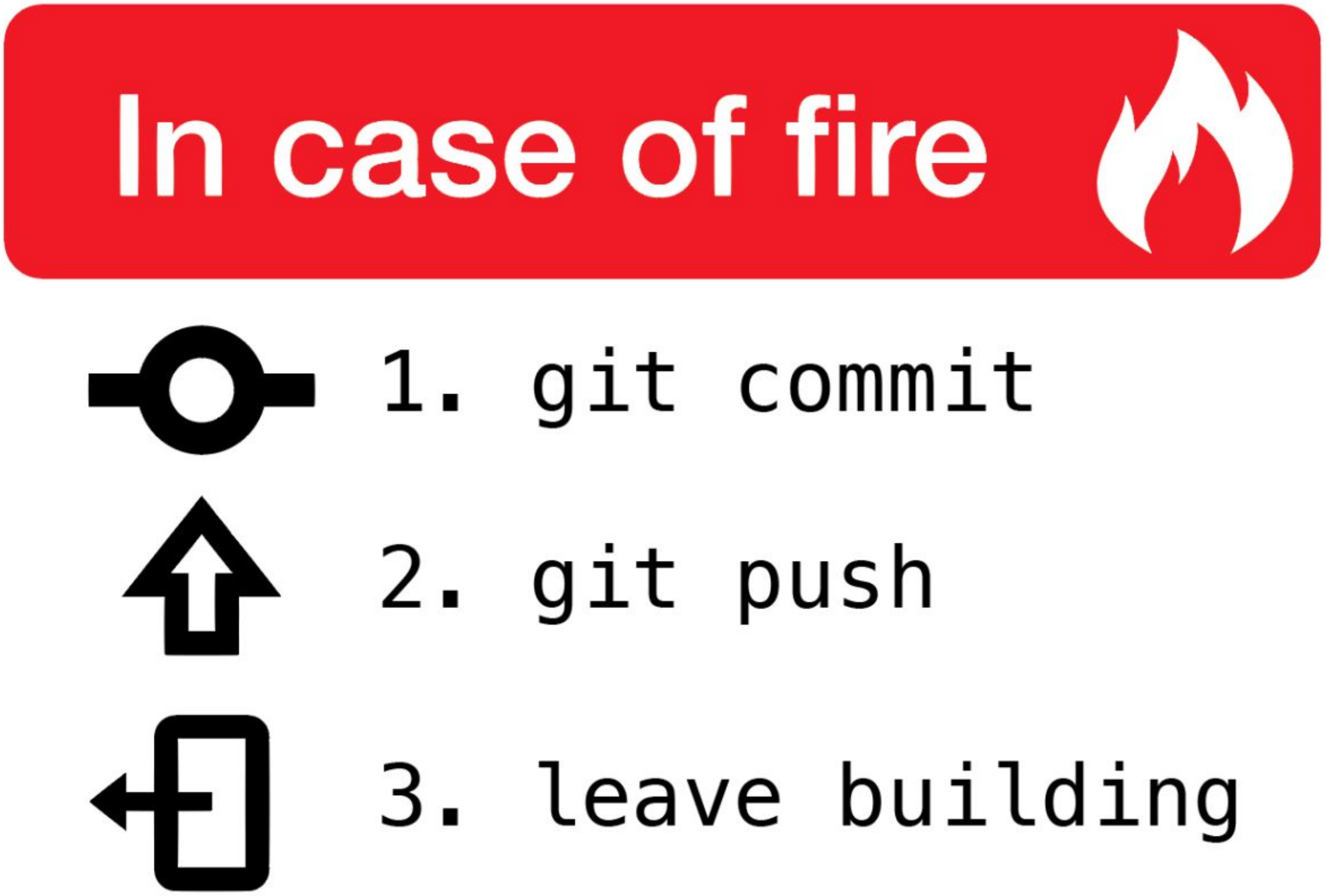
`git reflog`   get reference log

```
ad0096f HEAD@{10}: checkout: moving from new to master
d82a8e0 HEAD@{11}: commit: n3
2ae10cd HEAD@{12}: commit: n2
c1c51a3 HEAD@{13}: commit: n1
ad0096f HEAD@{14}: checkout: moving from master to new
ad0096f HEAD@{15}: commit: clean
```

Resources

1. About Git – short guide
   https://git-scm.com/book/en/v2
2. Git Reference Manual
   https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control
3. LearnGitBranching
   http://learngitbranching.js.org/
4. Git shell download page
   https://desktop.github.com/

In case of fire...



In case of fire

1. git commit
2. git push
3. leave building

Q&A

Do you have any questions?

Thank you!