

GIT – Version Control System

Łukasz Kurzaj

Support:

Piotr Koniarek, Maciej Krzesinski

As amended by: Marcel Rzepka

Agenda



■ Need for VCS

- Market solutions
- Why GIT?
- GIT assumptions
- Basic flow
- Working with branches
- Pull Requests
- Exercises
- Resources

Need for VCS and some context



Conventional project challenges

- Work on the same source code by whole team
- Tracking changes
- Ability to revert appropriate versions
- Requirements changing
- Priorities changing
- Software versioning
- Development on multiple operating systems
- Trace who changed what and why





Need for VCS and some context

Git was created by a Linus Torvalds (the Linux kernel creator) as a tool to support Linux kernel development. So its strongly influenced by a Linux.

. – symbol **for current directory**

.. – symbol **for parent directory**

1 - STDOUT

2 - STDERR

> - redirect

| - pipe

<command> --help #usually provides short command description

cd <path> #change directory

ls -la #list (files)

grep #command for filtering text with regex

mkdir <name> #make directory -p - to create parrents also

rmdir <path> #remove empty directory

rm <filename> #remove file

rm -r <dir> #remove -recursive - remove directory with its content (be cerfull if you delete your files there is not trashcan)

touch #create empty file

cat <plik> #print file

less <plik> #print file with more sophisticated software

History #



Need for VCS and some context

```
MARZEPKA@LCE31794 MINGW64 /tmp/test  
$ touch plik{1..100}.txt
```

```
MARZEPKA@LCE31794 MINGW64 /tmp/test  
$ ls  
plik1.txt plik24.txt plik4.txt plik55.txt plik70.txt plik86.txt  
plik10.txt plik25.txt plik40.txt plik56.txt plik71.txt plik87.txt  
plik100.txt plik26.txt plik41.txt plik57.txt plik72.txt plik88.txt  
plik11.txt plik27.txt plik42.txt plik58.txt plik73.txt plik89.txt  
plik12.txt plik28.txt plik43.txt plik59.txt plik74.txt plik9.txt  
plik13.txt plik29.txt plik44.txt plik6.txt plik75.txt plik90.txt  
plik14.txt plik3.txt plik45.txt plik60.txt plik76.txt plik91.txt  
plik15.txt plik30.txt plik46.txt plik61.txt plik77.txt plik92.txt  
plik16.txt plik31.txt plik47.txt plik62.txt plik78.txt plik93.txt  
plik17.txt plik32.txt plik48.txt plik63.txt plik79.txt plik94.txt  
plik18.txt plik33.txt plik49.txt plik64.txt plik8.txt plik95.txt  
plik19.txt plik34.txt plik5.txt plik65.txt plik80.txt plik96.txt  
plik2.txt plik35.txt plik50.txt plik66.txt plik81.txt plik97.txt  
plik20.txt plik36.txt plik51.txt plik67.txt plik82.txt plik98.txt  
plik21.txt plik37.txt plik52.txt plik68.txt plik83.txt plik99.txt  
plik22.txt plik38.txt plik53.txt plik69.txt plik84.txt  
plik23.txt plik39.txt plik54.txt plik7.txt plik85.txt
```

```
MARZEPKA@LCE31794 MINGW64 /tmp/test  
$ ls | grep 7  
plik17.txt  
plik27.txt  
plik37.txt  
plik47.txt  
plik57.txt  
plik67.txt  
plik7.txt  
plik70.txt  
plik71.txt  
plik72.txt  
plik73.txt  
plik74.txt  
plik75.txt  
plik76.txt  
plik77.txt  
plik78.txt  
plik79.txt  
plik87.txt  
plik97.txt
```

```
MARZEPKA@LCE31794 MINGW64 /tmp/test  
$ ls | grep 7 > wynik.txt
```

```
MARZEPKA@LCE31794 MINGW64 /tmp/test  
$ cat wynik.txt  
plik17.txt  
plik27.txt  
plik37.txt  
plik47.txt  
plik57.txt  
plik67.txt  
plik7.txt  
plik70.txt  
plik71.txt  
plik72.txt  
plik73.txt  
plik74.txt  
plik75.txt  
plik76.txt  
plik77.txt  
plik78.txt  
plik79.txt  
plik87.txt
```



Need for VCS and some context VIM

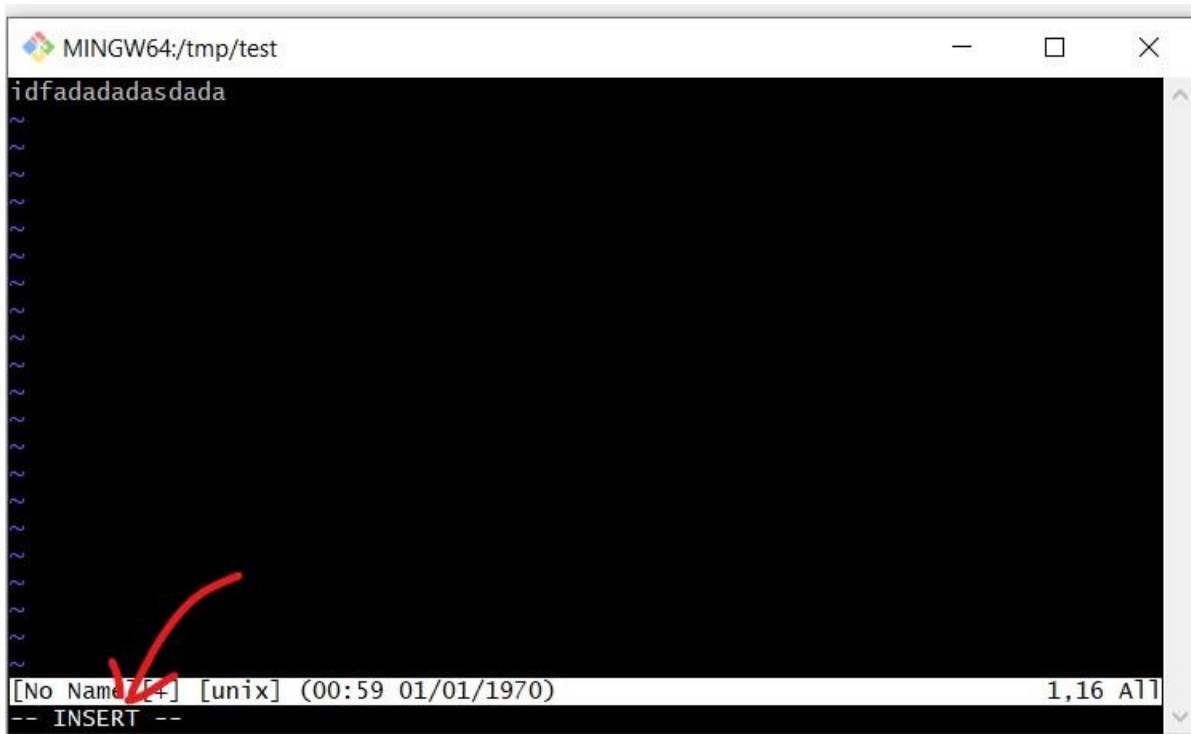
As the VIM is the default editor for GIT it might happen that you will end up with inside it.

Don't panic. We will practice exiting it together.

Go to git-bash

Type vim and hit enter

Start typing probably you will write i a s o or some other letter and enter insert mode.



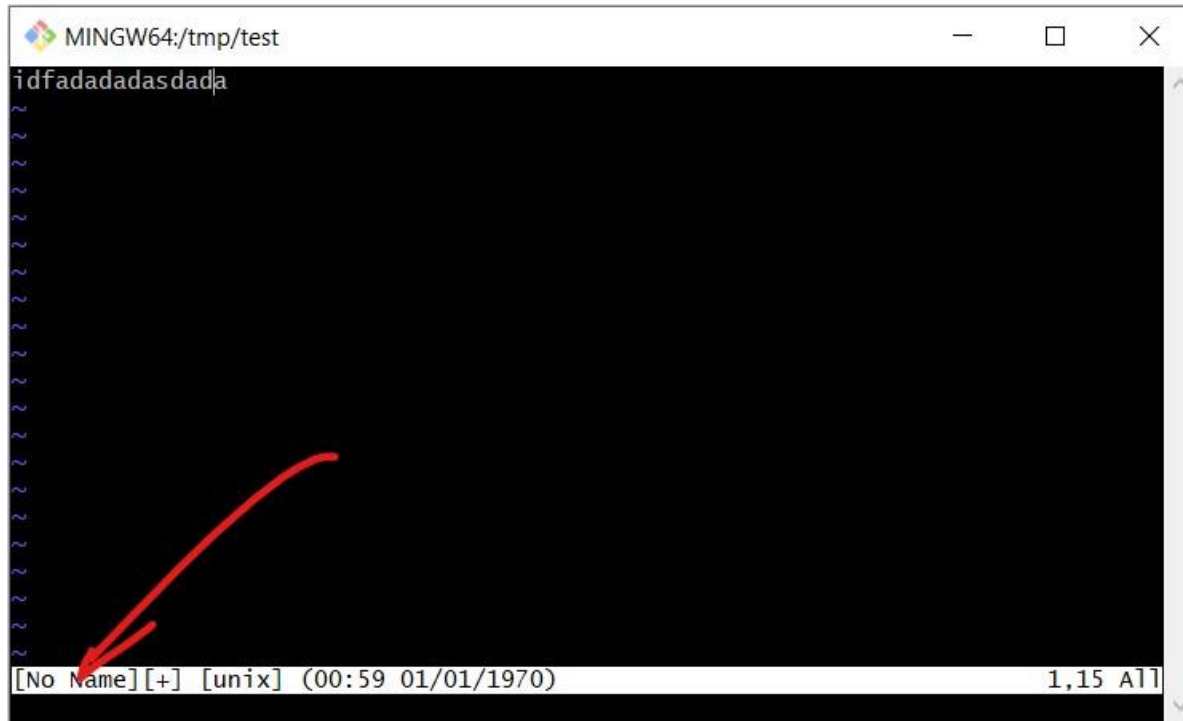
The screenshot shows a terminal window titled 'MINGW64:/tmp/test'. The main area is black with the text 'idfadadadasdada' in white. On the left side, there are several tilde (~) characters. At the bottom, a status bar shows '[No Name] [unix] (00:59 01/01/1970) 1,16 A |' and '-- INSERT --'. A red arrow points to the 'A' in the status bar.



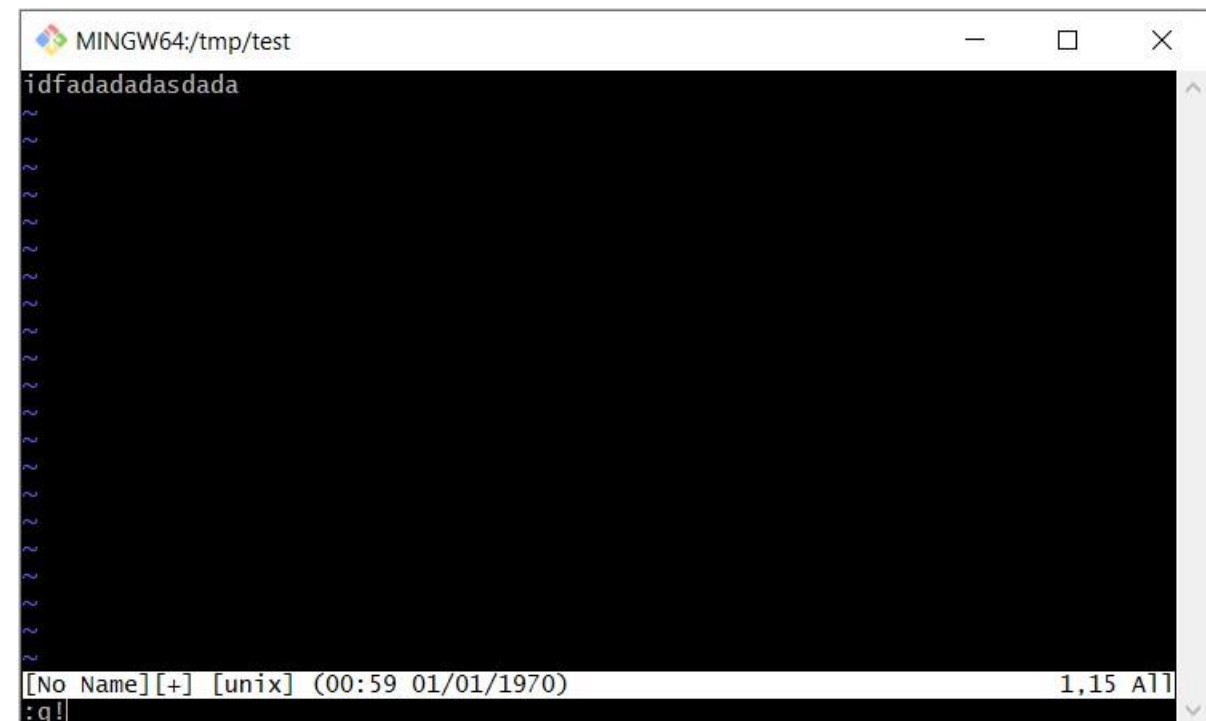


Need for VCS and some context VIM

First we need to exit insert mode – hit ESC
Insert mode label disappeared
Now type :q! and hit enter



A screenshot of a VIM editor window titled "MINGW64:/tmp/test". The main text area contains the string "idfadadasdada" on the first line. The status bar at the bottom displays "[No Name] [+] [unix] (00:59 01/01/1970) 1,15 All". A red arrow points from the bottom left towards the status bar.



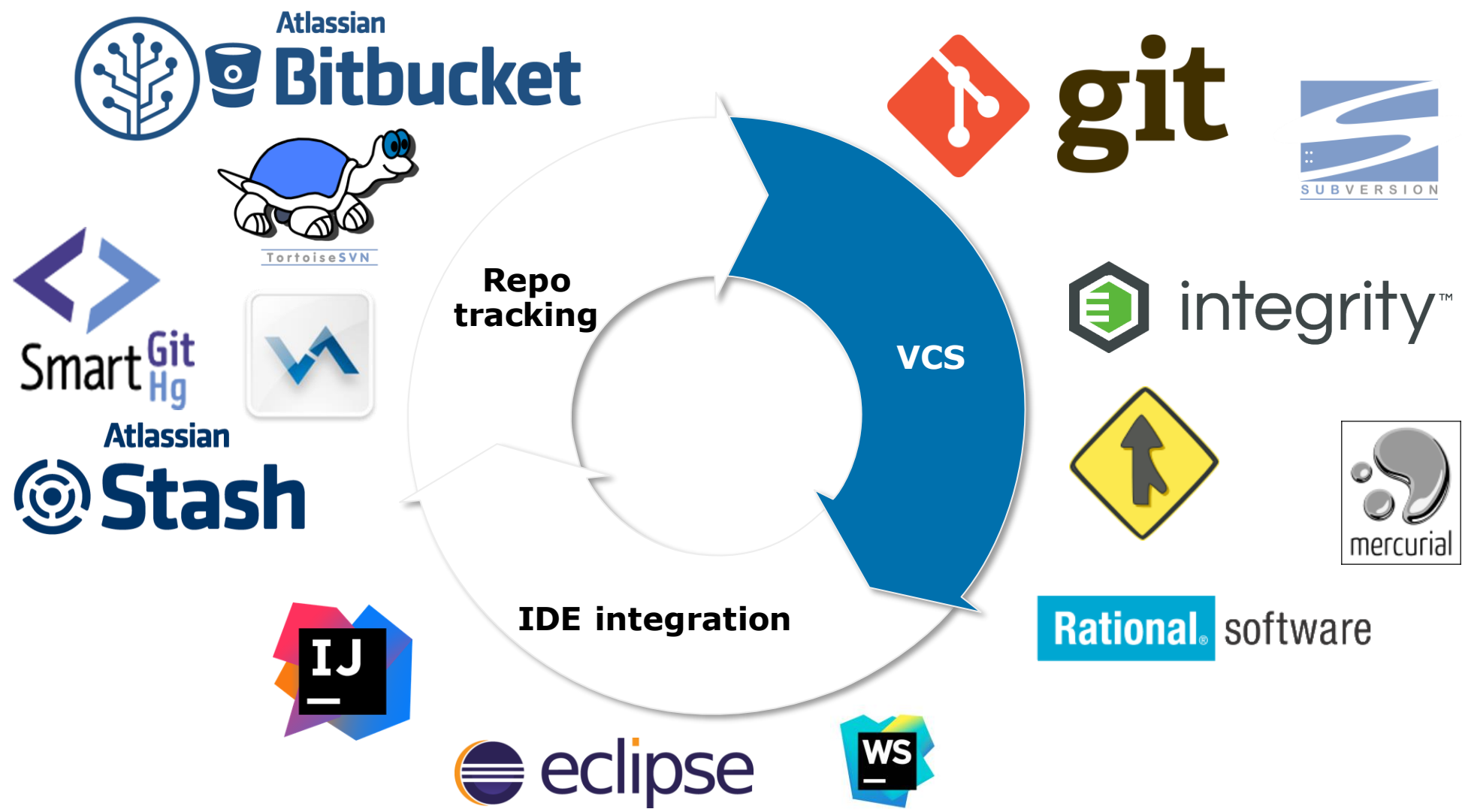
A screenshot of the same VIM editor window after typing ":q!". The status bar now shows ":q!" at the beginning, indicating the quit command is being executed.

Agenda



- Need for VCS
- **Market solutions**
- Why GIT?
- GIT assumptions
- Basic flow
- Working with branches
- Exercises
- Resources

Market solutions



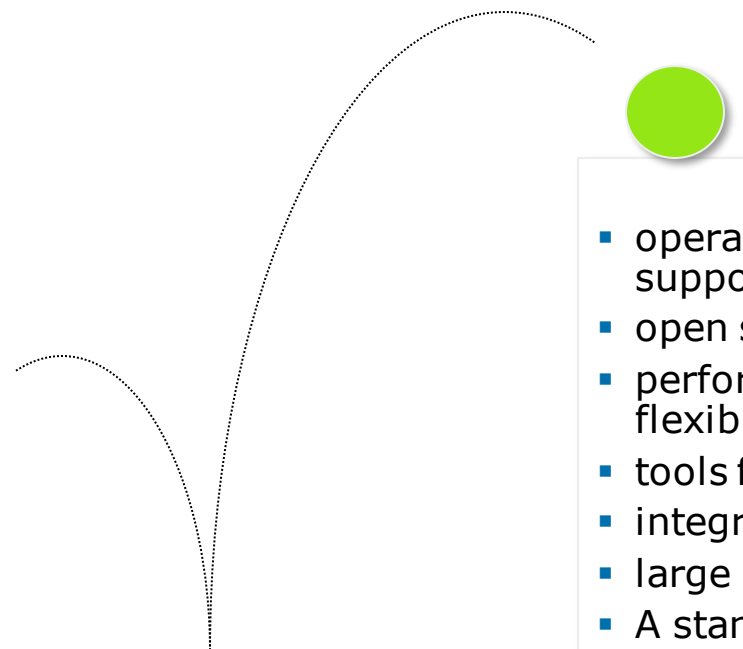
Agenda



- Need for VCS
- Market solutions
- **Why GIT?**
- GIT assumptions
- Basic flow
- Working with branches
- Exercises
- Resources

Why GIT?



- 
- A diagram of a plant with a green circle at the top, connected by a dotted line to a list of reasons for using Git. The plant has a main stem and two smaller branches, all represented by dotted lines. The green circle is positioned above the list.
- operating systems support
 - open source
 - performance, security, flexibility
 - tools for managing repo
 - integration with IDEs
 - large community
 - A standard now

Agenda



- Need for VCS
- Market solutions
- Why GIT?
- **GIT assumptions**
- Basic flow
- Working with branches
- Exercises
- Resources



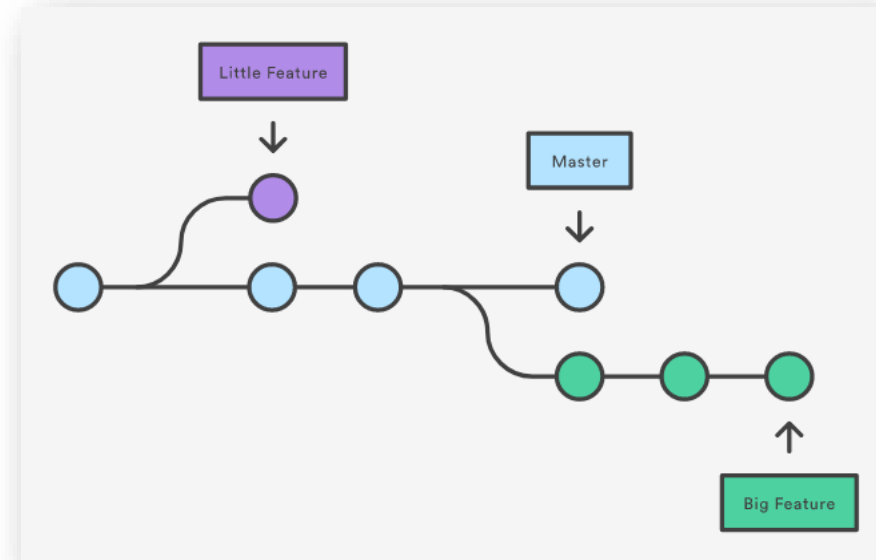
GIT assumptions



Branch – represents an independent line of development, allows to work concurrently on multiple features according to requirements and priorities

Branches meaning

- Master – milestone's , for example releases
- Develop – working branch, team merges to this branch in development phase
- Feature branch – related with only one feature
- Bugfix branch related with only one bug
- others: depends on team



GIT assumptions



Commit – represents a set of changes, with meta information such as subject, author and date

HEAD – pointer to current commit; lowercase head – pointer to something

hash – a hexadecimal (containing 0-9a-f) string of a given length e.g.
b7a9a0416ff59df80950ee5243e33bd26853c67b

GIT assumptions



Working Area – relates to your local directory

Staging area/index – what is going to be in the next commit

Repository – contains all commits and history

GIT assumptions



Git – Application for source code management

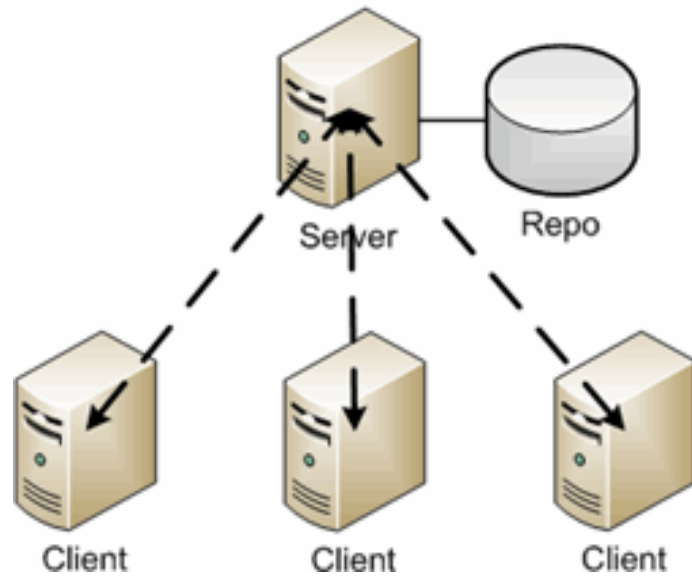
GitHub – a page owned by Microsoft which hosts git repositories so anybody can access them. It host a lot of open source projects

GitLab – a page gitlab.com which hosts git repositories and anybody can access them. But also it is possible to set up gitlab in your own infrastructure. Be careful not to confuse those two!!! Code leakage is a real threat and you will suffer consequences.

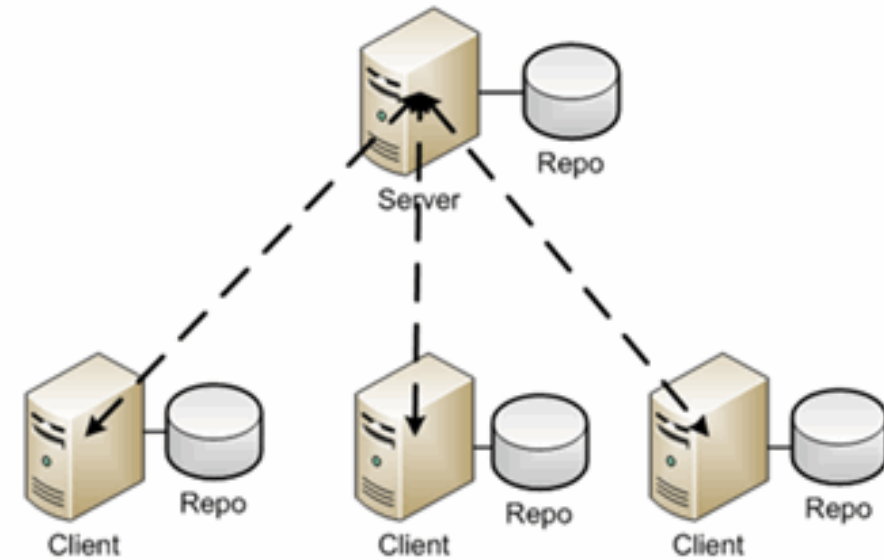
GIT assumptions – distributed VCS



Traditional



Distributed



Agenda



- Need for VCS
- Market solutions
- Why GIT?
- GIT assumptions
- **Basic flow**
- Working with branches
- Exercises
- Resources

Basic flow



Basics

Setting up repository

- git init
- git clone
- git config

Saving changes

- git add
- git commit
- .gitignore

Inspecting repository

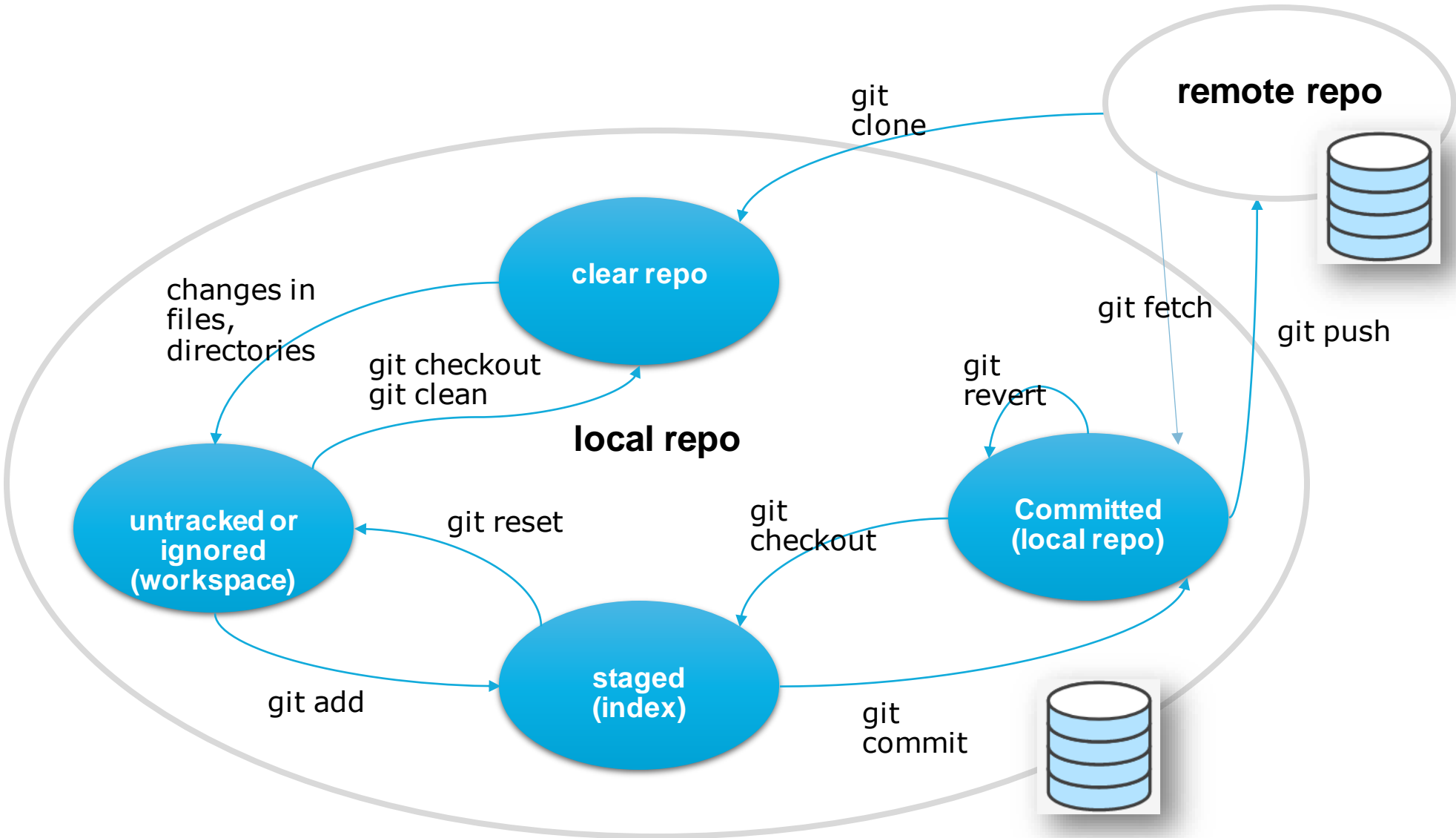
- git log
- git status

Undoing changes

- git checkout
- git revert
- git reset
- git clean

Headline

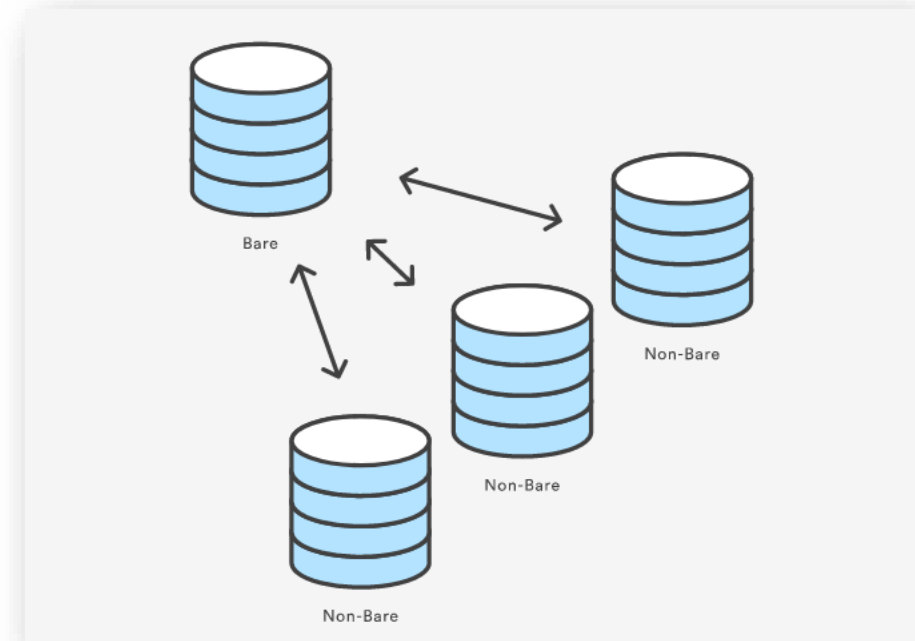
Basic flow



Basic flow – setting up repository

GIT INIT

- creates empty repository
- converts existing unversioned project into git repository
- creates .git subdirectory in the current directory (metadata includes subdirectories for objects, refs and template files)
- `git init --bare #trivia`

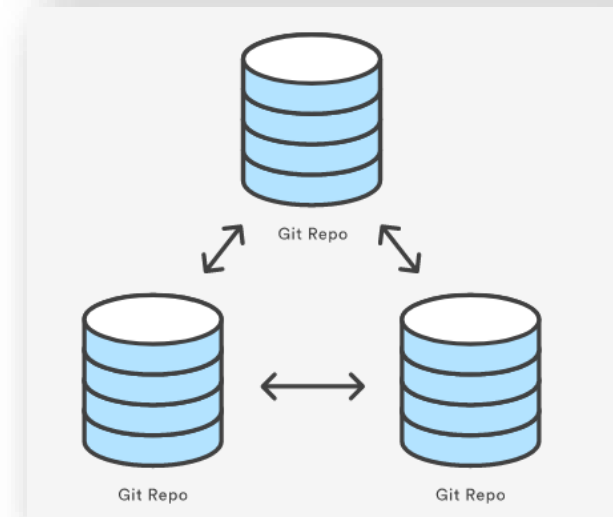
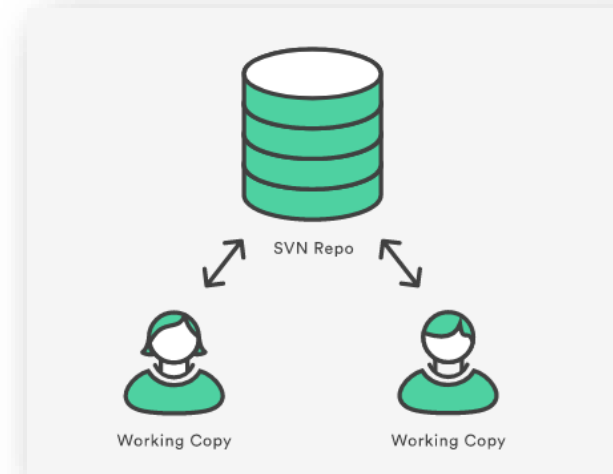


Basic flow – setting up repository



GIT CLONE

- firstly git creates empty repository and then copies the data from the existing repository
- clones local or remote repository
- supports network protocols: ssh, http, https, ftp etc.
- multiple configuration options
 - clone branch
 - clone directory





Basic flow – setting up repository

GIT CONFIG

LOCAL

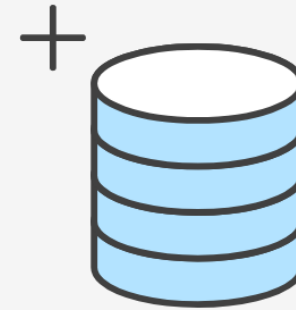
- set by default
- configuration stored in `.git/config`

GLOBAL

- user-specific configuration stored in:
 - UNIX: `~/.gitconfig`
 - WINDOWS: `C:\Users\<username>\.gitconfig`

SYSTEM

- configuration for all users and all repos across machine
 - UNIX: `$(prefix)/etc/gitconfig`
 - WINDOWS: `C:\ProgramData\config`



Basic flow



Basics

Setting up repository

- git init
- git clone
- git config

Saving changes

- git add; git mv; git rm
- git commit
- .gitignore

Inspecting repository

- git log
- git status

Undoing changes

- git checkout
- git revert
- git reset
- git clean

Headline

Basic flow – saving changes

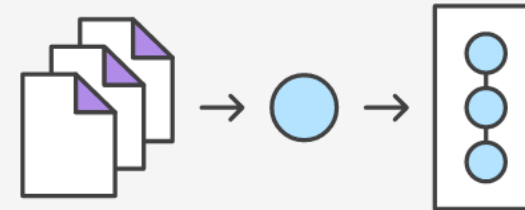


GIT ADD

- stages changes in files, directories

GIT COMMIT

- commits staged changes to local repository





Basic flow – saving changes

.gitignore states

tracked

files and directories which have already been staged or committed

untracked

files and directories which have not been staged or committed yet

ignored

files and directories which GIT has been explicitly told to ignore in .gitignore file

- dependency caches, such as the contents of /node_modules or /packages
- compiled code, such as .o, .pyc, and .class files
- build output directories, such as /bin, /out, or /target
- files generated at runtime, such as .log, .lock, or .tmp
- hidden system files, such as .DS_Store or Thumbs.db
- personal IDE config files, such as .idea/workspace.xml

Basic flow



Basics

Setting up repository

- git init
- git clone
- git config

Saving changes

- git add
- git commit
- .gitignore

Inspecting repository

- git log
- git status

Undoing changes

- git checkout
- git revert
- git reset
- git clean

Headline



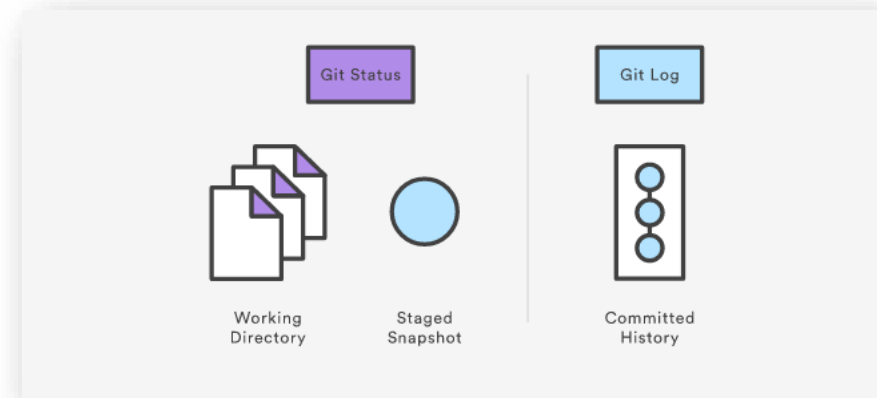
Basic flow – inspecting repository

GIT STATUS

- displays list staged, unstaged and untracked files
- displays the state of the working directory and the staging area

GIT LOG

- displays committed snapshots
- used only for committing history



Basic flow



Basics

Setting up repository

- git init
- git clone
- git config

Saving changes

- git add
- git commit
- .gitignore

Inspecting repository

- git log
- git status

Undoing changes

- git checkout
- git revert
- git reset
- git clean

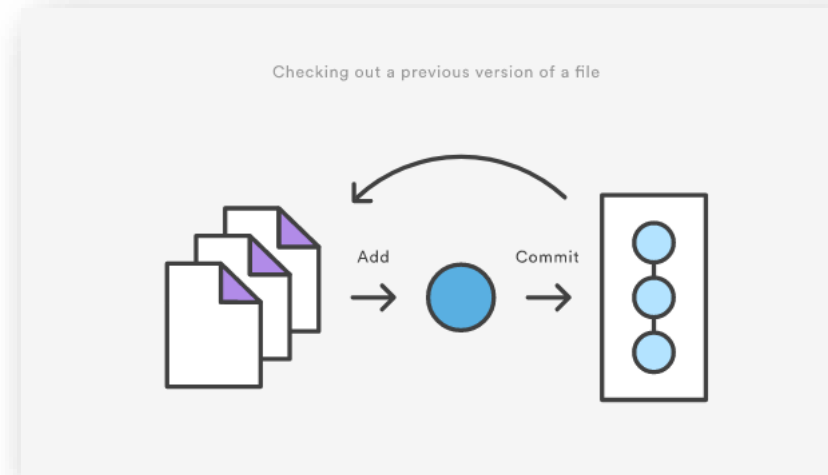
Headline

Basic flow – undoing changes



GIT CHECKOUT

- allows to switch branch
- turns file that resides in the working directory into an exact copy of the one from commit and add to staging area
- updates all files in working directory to match the specified commit





Basic flow – undoing changes

GIT REVERT

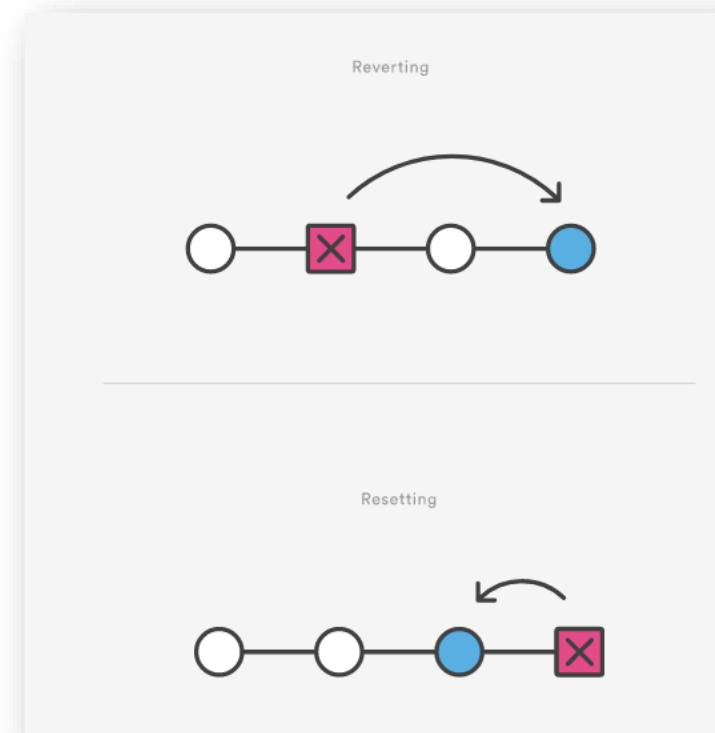
- generates a new commit that undoes all of the changes introduced in <commit> then apply to current branch
- doesn't affect existing commits history

GIT RESET

- removes files from staged area
- --hard flag allows to override current unstaged changes too

GIT CLEAN

- removes untracked files from directories
- -f flag means clean force



Agenda



- Need for VCS
- Market solutions
- Why GIT?
- GIT assumptions
- Basic flow
- **Working with branches**
- Exercises
- Resources

Working with branches



Multibranches work

Syncing

Headline

- git fetch
- git remote
- git pull
- git push

Manage branches

- git branch
- git checkout
- git merge

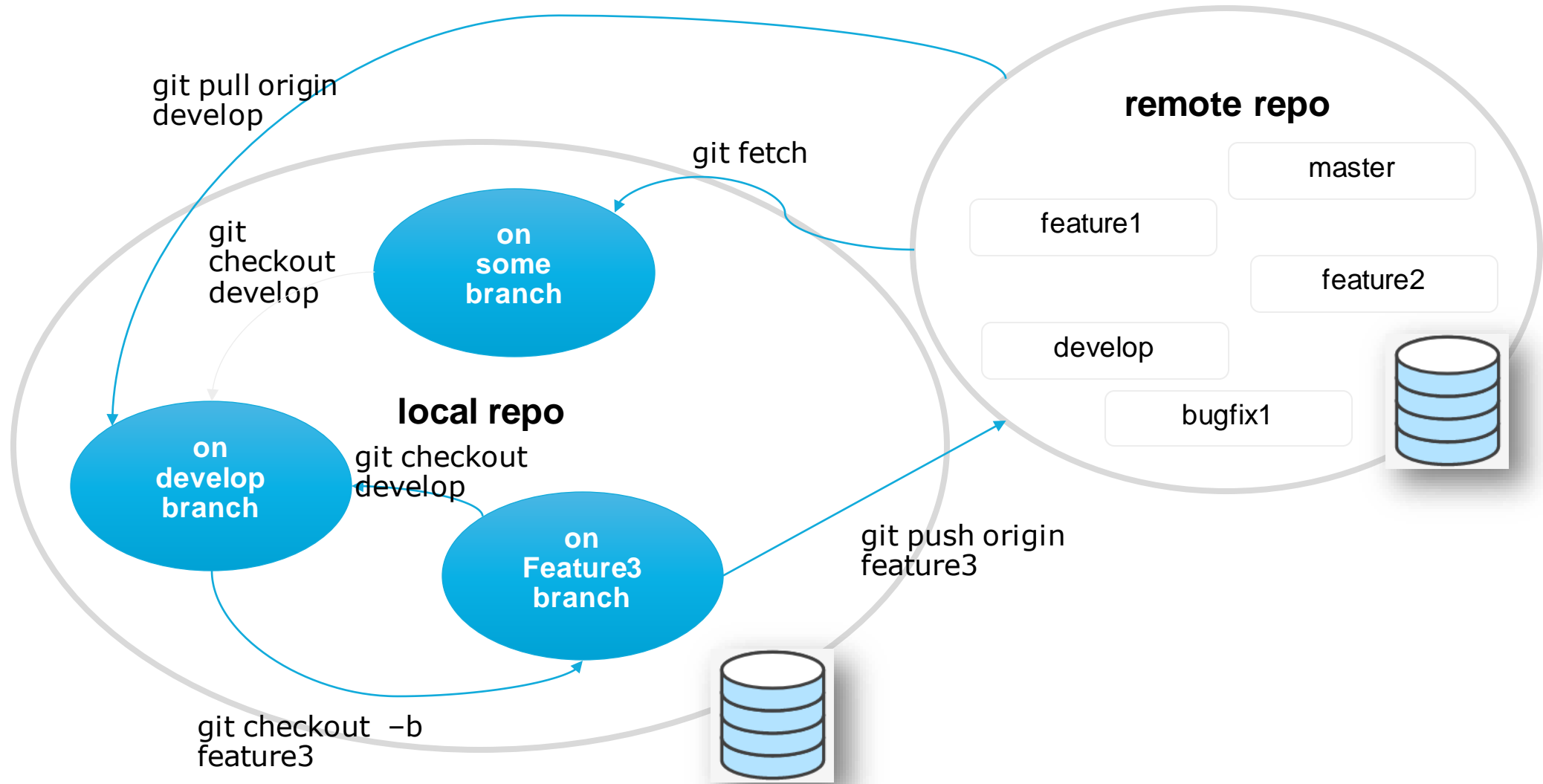
Pull requests

- definition
- flow

Conflicts

- flow

Working with branches - basic flow – add new branch



Working with branches - syncing

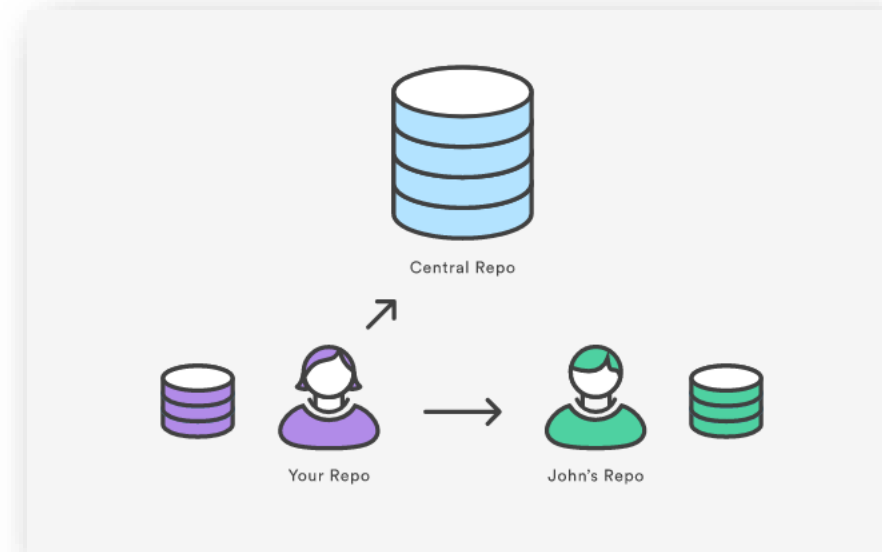


GIT FETCH

- imports all commits from a remote repository to local repository

GIT REMOTE

- creates, view and deletes connections to other repositories

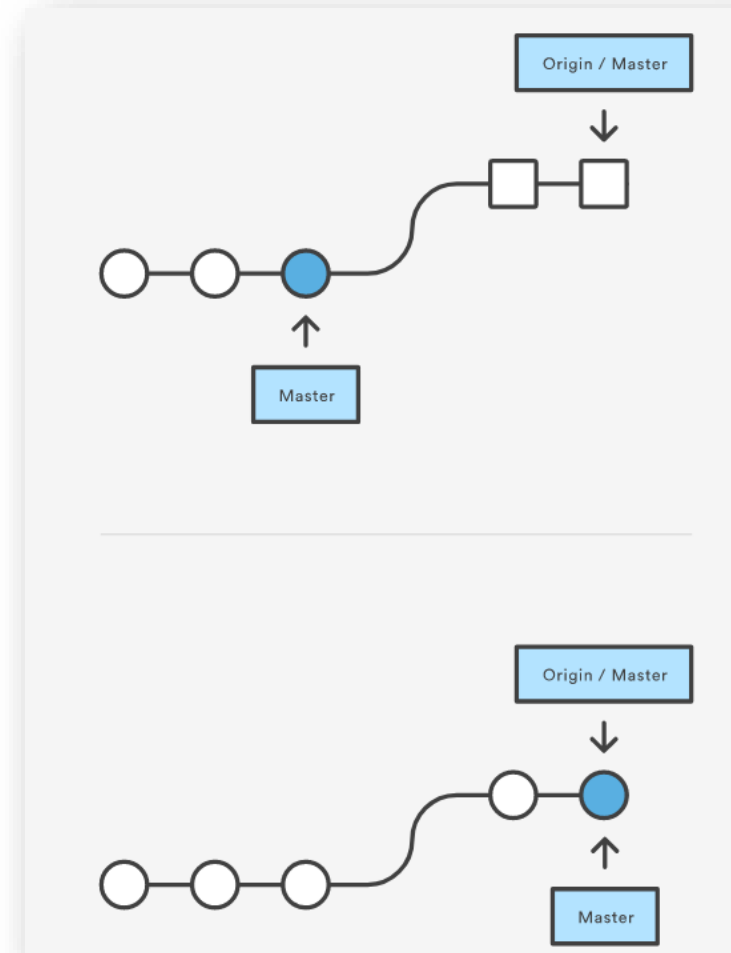


Working with branches - syncing



GIT PULL

- performs git fetch and git merge

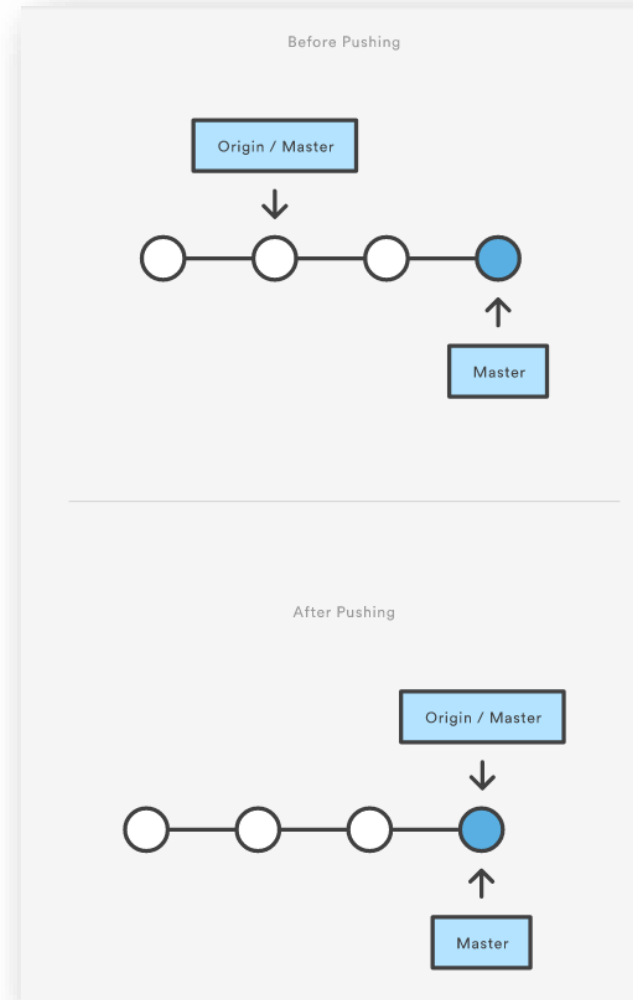


Working with branches - syncing



GIT PUSH

- transfers commits from local repository to remote repository
- counterpart to git fetch
- use it carefully only on single branches**



Working with branches



Multibranches work

Syncing

Manage branches

Pull requests

Conflicts

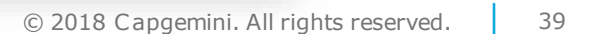
Headline

- git fetch
- git remote
- git pull
- git push
- git branch
- git checkout
- git merge
- definition
- flow
- flow

- lets to create, list, rename and delete branches

- lets to navigate between branches
- creates a new branch

- lets to integrate current branch with single other branch
- generates merge commit



Working with branches



Multibranches work

Syncing

- git fetch
- git remote
- git pull
- git push

Manage branches

- git branch
- git checkout
- git merge

Pull requests

- definition
- flow

Conflicts

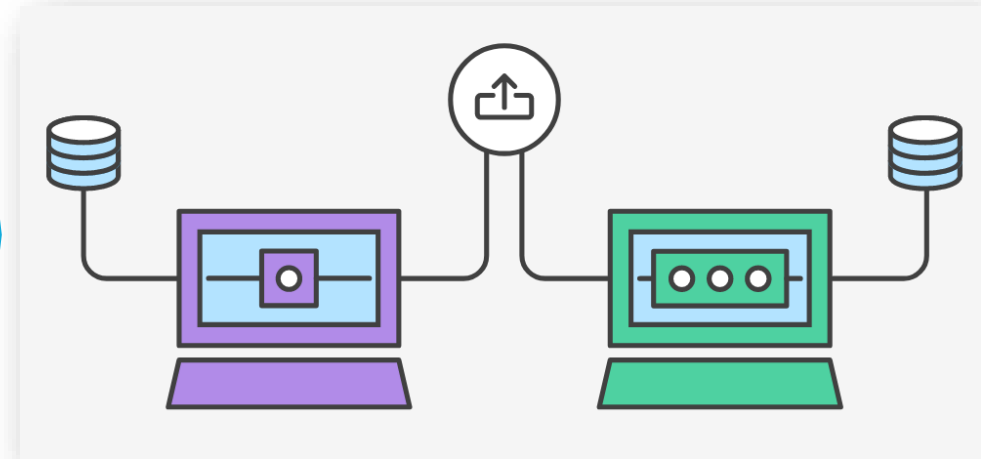
- flow

Headline

Working with branches – pull requests

PULL REQUESTS

- feature that makes it easier for developers to collaborate using git
- provides user-friendly interface for discussing proposed changes before integrating them into the official project
- isn't part of git. It is feature provided by git tracking systems like: stash, bitbucket, github etc.

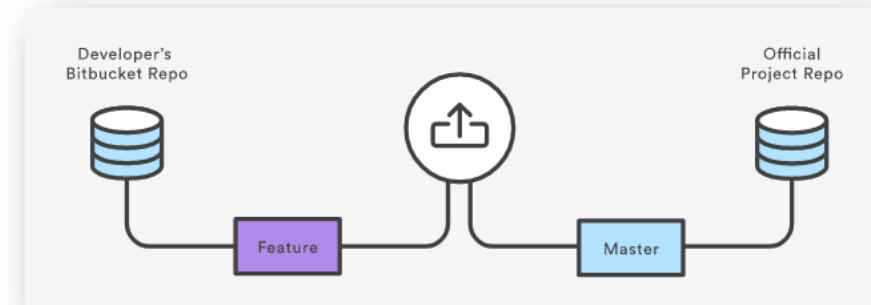




Working with branches – pull requests

FLOW

- developer creates the feature in dedicated branch in local repo
- developer pushes the branch to a remote repository
- developer creates pull request via git tracking system
- team reviews the code, discusses it, developer updates source code
- the project maintainer merges the feature into the official repository
- pull request is closed



Working with branches



Multibranches work

Syncing

- git fetch
- git remote
- git pull
- git push

Manage branches

- git branch
- git checkout
- git merge

Pull requests

- definition
- flow

Conflicts

- flow

Headline



Working with branches – conflicts

If the two branches trying to merge both changed the part of the same file, won't be able to figure out which version to use

RESOLVE CONFLICTS

- processes very familiar to merging process
edit/stage/commit

```
# On branch master
# Unmerged paths:
# (use "git add/rm ..." as appropriate to mark resolution)
#
# both modified: hello.py
#
```

Agenda



- Need for VCS
- Market solutions
- Why GIT?
- GIT assumptions
- Basic flow
- Working with branches
- **Pull Requests**
- Exercises
- Resources



Pull Requests - features of good pull request

Code

- It should be complete, but minimal (there is nothing worse than a pull request spanning along 50 files and hundreds of lines)
- The code should be formatted, according to style guide and before defined autoformatting tools

Review

- Shortly describe a purpose (e.g. fix handling empty json files)
- Consider linking a ticket
- Keep in mind that this description might be a trace in a bugfixing process in the feature
- If you have any doubts about code you have written highlight them upfront
- If for some reason you already created a pull request but you need to amend something then prepend the title with WIP: (work in progress)
- Mention @people directly affected by your changes, and those who might want to take part in a discussion
- Answer to every comment
- Don't hesitate to ask for clarification or explain your standpoint (there is nothing to defend as the reviewer is not your enemy, but an ally)



Pull Requests - features of good feedback

- Don't hesitate to ask
- Try to understand what is a purpose of a change
- Don't argue but discuss and ask (Why did you took this approach? I don't like it because... I would opt for... Instead of ~~Never write like this. Don't do~~)
- Explain your reasoning – code not align to the style guide, personal preference, maybe you spotted some corner case
- Offer ways to improve and simplify your code
- Stay humble, but bold – (I'm not sure Let's try) – even when you are not sure its good to say something than let the bug slip
- Make your review an opportunity to learn for both parties
- Remember that's it's not about shaming and blaming but making the code better
- Sometimes emojis might be beneficial
- Don't hesitate to have a quick call to clarify something



Pull Requests – creating a pull request in Gitlab

- On the left-hand side go to Repository->Branches
- Find a branch you are working with
- Click on Merge request button
- You should see a form to fill in
- Provide appropriate title
- Remember about providing good description (sometimes it's possible to use provided template)
- Depending on your project process select assignee (usually someone who knows context)
- It's good to check remove source branch when merge request is accepted in order to clean up the repo
- You also can consider option to Squash commits when merge request is accepted, in order to use one commit with your merge request title instead of tens like fixing this

Agenda



- Need for VCS
- Market solutions
- Why GIT?
- GIT assumptions
- Basic flow
- Working with branches
- **Exercises**
- Resources

Exercises



1. Create file A.txt and B.txt both containing word test
2. Add them to repository
3. Rename file A.txt to 1.txt with `mv A.txt 1.txt` command and B.txt to 2.txt with `git mv B.txt 2.txt` command
4. What is the difference?

Agenda



- Need for VCS
- Market solutions
- Why GIT?
- GIT assumptions
- Basic flow
- Working with branches
- Exercises
- **Resources**

What to do after?



Documentation

- <https://training.github.com/downloads/pl/github-git-cheat-sheet/>
- <https://git-scm.com/docs>
- <https://ndpsoftware.com/git-cheatsheet.html>
- <https://www.toptal.com/developers/gitignore>
- <https://github.com/LeCoupa/awesome-cheatsheets/blob/master/languages/bash.sh>
- <https://learngitbranching.js.org/>

Interactive tutorials

Food for thought

- <https://opensource.com/article/18/6/anatomy-perfect-pull-request>
- <https://www.gitops.tech/>
- <https://www.atlassian.com/git/tutorials/comparing-workflows>



- **Pro Git, Second Edition**, Scott Chacon; Ben Straub, Apress, 2014
- **Git Pocket Guide**, Richard E. Silverman, O'Reilly Media, Inc., 2013
- **Git Best Practices Guide**, Eric Pidoux, Packt Publishing, 2014
- **Git: Distributed Version Control**, René Preißel; Bjørn Stachmann, Brainy Software, 2014
- website: <https://git-scm.com/docs>
- website: <https://www.atlassian.com/git/tutorials>
- website: <https://try.github.io/levels/1/challenges/1>



- Git hooks



People matter, results count.

This message contains information that may be privileged or confidential and is the property of the Capgemini Group.

Copyright © 2017 Capgemini. All rights reserved.

Rightshore® is a trademark belonging to Capgemini.

About Capgemini

A global leader in consulting, technology services and digital transformation, Capgemini is at the forefront of innovation to address the entire breadth of clients' opportunities in the evolving world of cloud, digital and platforms. Building on its strong 50-year heritage and deep industry-specific expertise, Capgemini enables organizations to realize their business ambitions through an array of services from strategy to operations. Capgemini is driven by the conviction that the business value of technology comes from and through people. It is a multicultural company of 200,000 team members in over 40 countries. The Group reported 2016 global revenues of EUR 12.5 billion.

Visit us at

www.capgemini.com

This message is intended only for the person to whom it is addressed. If you are not the intended recipient, you are not authorized to read, print, retain, copy, disseminate, distribute, or use this message or any part thereof. If you receive this message in error, please notify the sender immediately and delete all copies of this message.