

Современные средства разработки ПО

Инструменты командной разработки ПО

МГТУ им. Н.Э. Баумана, ИУ-6, Фетисов Михаил Вячеславович
fetisov.michael@bmstu.ru

Содержание

- Системы управления версиями
- Git
- Внешние репозитории кода, GitHub/GitLab
- CI/CD и цикл разработки
- Примеры

Системы управления версиями

- Version Control System, VCS
- Программное обеспечение, обеспечивающее версионирование текстовых документов в многопользовательской среде
- Позволяет:
 - хранить несколько версий одного и того же текстового документа;
 - возвращаться к более ранним версиям;
 - определять, кто и когда сделал то или иное изменение;
 - и многое другое.

Централизованные системы управления версиями

- Репозиторий расположен на сервере
- Централизованное управление
- Управление коллизиями: блокировка файла
- Дополнительно:
 - поддержка ветвей;
 - пометка версий тегами.
- Цикл работы:
 - рабочий каталог,
 - checkout,
 - checkin.
- Примеры централизованных систем:
 - Concurrent Versions System (CVS), Subversion, Mercurial;
 - Team Foundation Server (TFS), Visual Studio Team System (VSTS), Azure DevOps Server (с 2019 года).

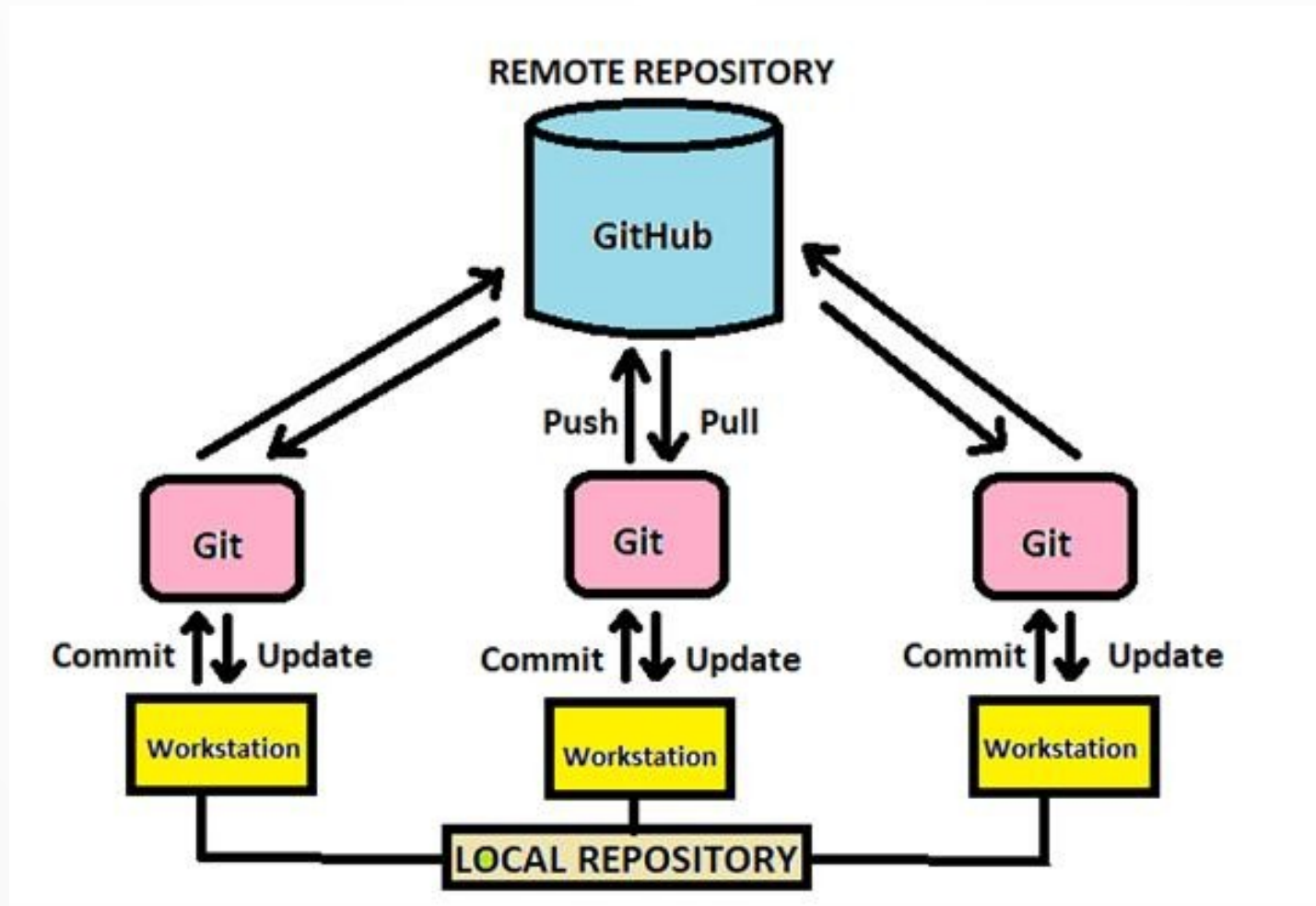
Распределённые системы управления версиями

- Distributed Version Control System, DVCS
- Вся история изменений храниться в локальном репозитории на компьютере разработчика
- Фрагменты истории локального хранилища синхронизируются с аналогичным хранилищем на другом компьютере
- Управление коллизиями:
 - слияние различий (merge),
 - ручное решение конфликтов.

Git

- Распределённая система управления версиями исходного кода ПО
- Возможности:
 - распределённая:
 - предоставляет каждому разработчику локальную историю разработки,
 - не требует захватывать файлы для выполнения изменений;
 - быстрое разделение и слияние версий;
 - предоставляет несколько протоколов для доступа к удалённому репозиторию (GitHub, GitLab и пр.).
- Аналоги:
 - Bazaar (bzd), Launchpad.

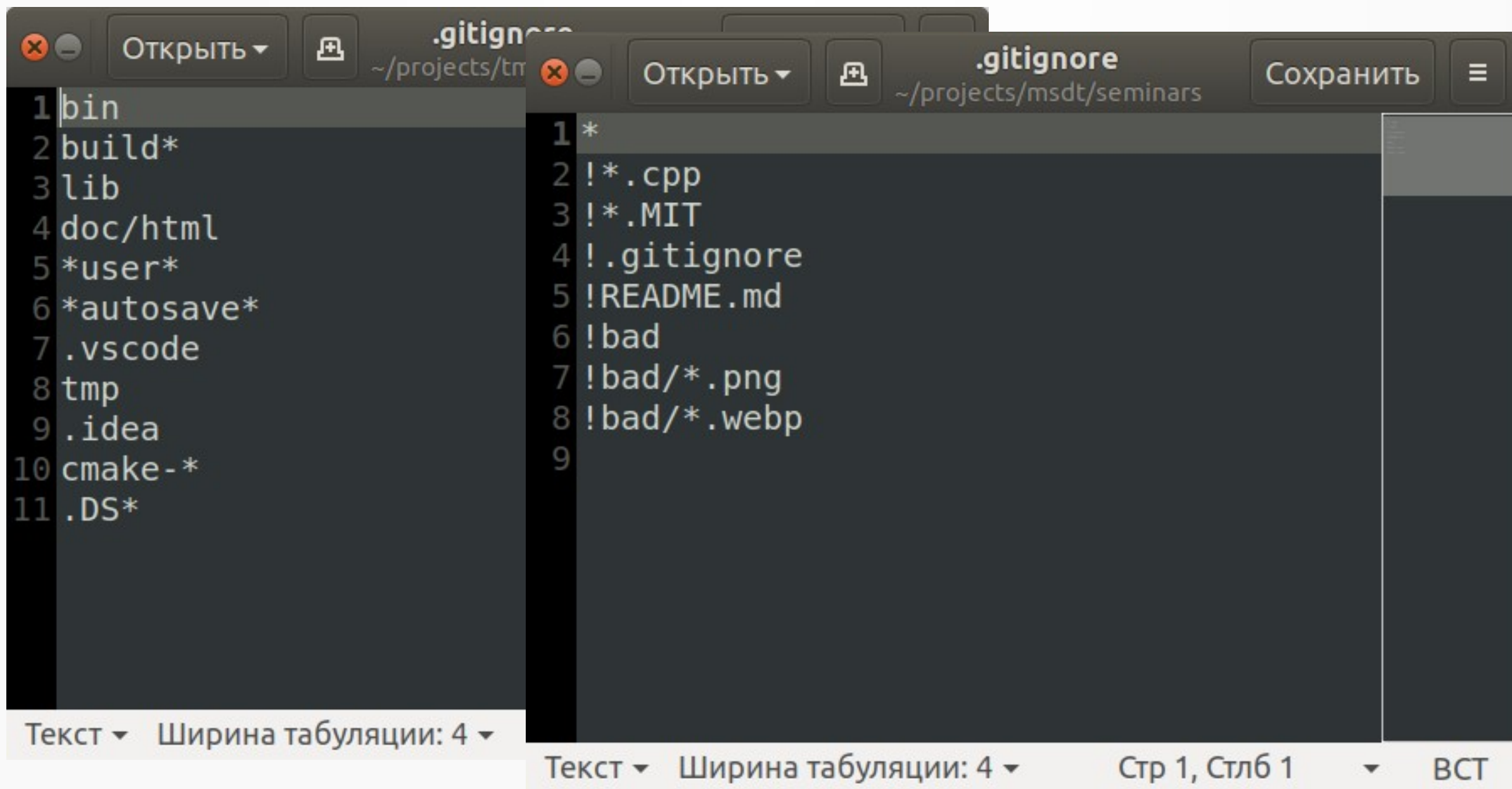
Git. Распределённая управления версиями



Git. Создание проекта, конфигурация

- `git help`
- `git init`
- `git config --local user.name "<Имя Фамилия>"`
- `git config --local user.email "<адрес эл. Почты>"`
- `git config --local -l`
- Области конфигураций:
 - local
 - global
 - system

.gitignore – список исключений



The image shows two overlapping code editor windows, both displaying .gitignore files. The left window is titled ".gitignore" and shows a list of directories and files to ignore. The right window is also titled ".gitignore" and shows a list of file patterns to ignore.

Left Window (.gitignore):

```
1 bin
2 build*
3 lib
4 doc/html
5 *user*
6 *autosave*
7 .vscode
8 tmp
9 .idea
10 cmake-*
11 .DS*
```

Right Window (.gitignore):

```
1 *
2 !*.cpp
3 !*.MIT
4 !.gitignore
5 !README.md
6 !bad
7 !bad/*.png
8 !bad/*.webp
9
```

Both windows have a status bar at the bottom indicating "Текст" (Text) and "Ширина табуляции: 4" (Tab width: 4). The right window also shows "Стр 1, Стлб 1" (Line 1, Column 1) and "ВСТ" (End of file).

Git. Копирование проекта

- Команда:

```
git clone <адрес проекта>
```

- Пример:

```
git clone git@bmstu.codes:lsx/msdt-study-group/homework/detach-2.git
```

```
git clone https://bmstu.codes/lsx/msdt-study-group/homework/detach-2.git
```

Git. Сохранение изменений в проекте

- Индексация файла(ов):
git add <имя файла>
git add .
- Фиксация изменений:
git commit -m "<описание изменения>"
- Сохранение изменений в удалённом репозитории:
git push

Git. Этапы сохранения изменений файла



Git. Удалённый репозиторий (УР)

- Подключение к УР:
git remote add origin <адрес>
- Получение проекта из УР:
git pull
git clone <адрес>
- Сохранение изменений в УР:
git push -u origin master
git push

Git. Просмотр изменений, удаление

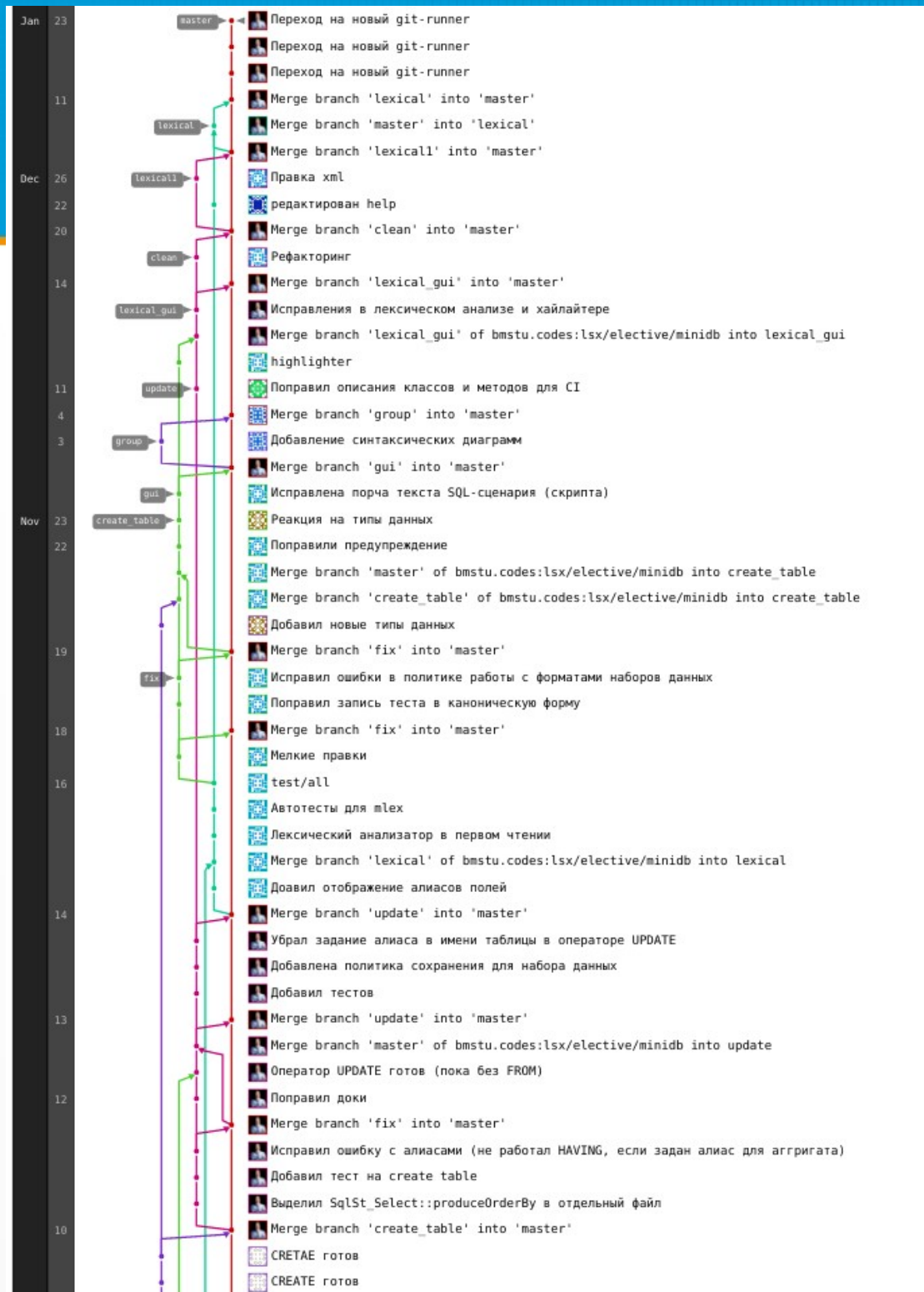
- Изменения на уровне списка файлов:
 - `git status`
- Сравнение файлов:
 - `git diff <имя файла>`
- Удаление:
 - `git rm`
 - `git rm -f`

Git. Ветвление

- Создание ветки:
 - `git branch <имя ветки>`
 - `git checkout <имя ветки>`
 - `git checkout -b <имя ветки>`
- Просмотр веток:
 - `git branch`
- Слияние веток:
 - `git checkout master`
 - `git merge <имя ветки>`

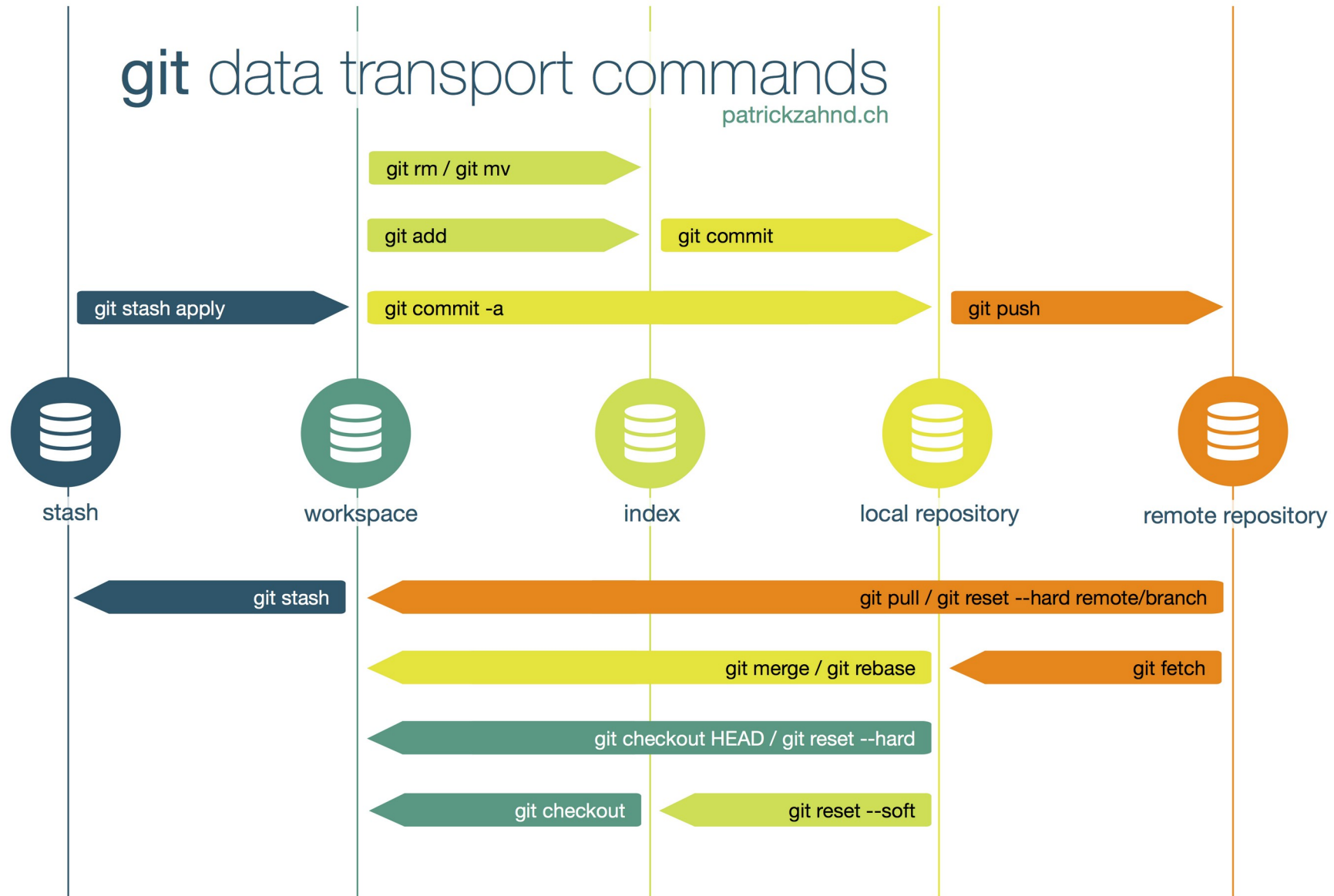
Git. Ветвление

Пример



Git

Cxema 1



Git Cheat Sheet

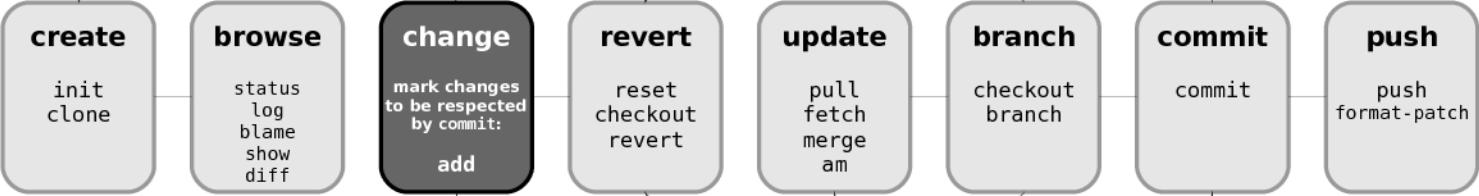
by Jan Krüger <jk@jk.gs>, <http://jan-krueger.net/git/>
Based on work by Zack Rusin

Basics

Use `git help [command]` if you're stuck.

master	default devel branch
origin	default upstream branch
HEAD	current branch
HEAD^	parent of HEAD
HEAD~4	great-great grandparent of HEAD
foo..bar	from branch <i>foo</i> to branch <i>bar</i>

(left to right) Command Flow



Create

From existing files

```
git init
git add .
```

From existing repository

```
git clone ~/old ~/new
git clone git://...
git clone ssh://...
```

Publish

In Git, `commit` only respects changes that have been marked explicitly with `add`.

```
git commit [-a]
  (-a: add changed files
  automatically)
git format-patch origin
  (create set of diffs)
git push remote
  (push to origin or remote)
git tag foo
  (mark current version)
```

Useful Tools

```
git archive
  Create release tarball
git bisect
  Binary search for defects
git cherry-pick
  Take single commit from elsewhere
git fsck
  Check tree
git gc
  Compress metadata (performance)
git rebase
  Forward-port local changes to
  remote branch
git remote add URL
  Register a new remote repository
  for this tree
git stash
  Temporarily set aside changes
git tag
  (there's more to it)
gitk
  Tk GUI for Git
```

Tracking Files

```
git add files
git mv old new
git rm files
git rm --cached files
  (stop tracking but keep files in working dir)
```

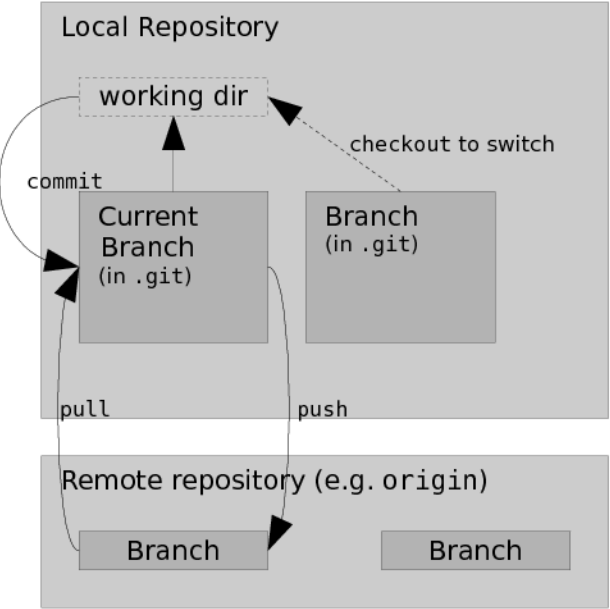
View

```
git status
git diff [oldid newid]
git log [-p] [file|dir]
git blame file
git show id (meta data + diff)
git show id:file
git branch (shows list, * = current)
git tag -l (shows list)
```

Update

```
git fetch (from def. upstream)
git fetch remote
git pull (= fetch & merge)
git am -3 patch.mbox
git apply patch.diff
```

Structure Overview



Revert

In Git, `revert` usually describes a new commit that undoes previous commits.

```
git reset --hard (NO UNDO)
  (reset to last commit)
git revert branch
git commit -a --amend
  (replaces prev. commit)
git checkout id file
```

Branch

```
git checkout branch
  (switch working dir to branch)
git merge branch
  (merge into current)
git branch branch
  (branch current)
git checkout -b new other
  (branch new from other and
  switch to it)
```

Conflicts

Use `add` to mark files as resolved.

```
git diff [--base]
git diff --ours
git diff --theirs
git log --merge
gitk --merge
```

Git Cheat Sheet

git <COMMAND> --help

git config --help

★ Create

```
cd ~/my_project_directory
git init
git add .
```

```
git clone ~/existing_repo ~/new/repo
git clone git://host.org/project.git
git clone ssh://user@host.org/project.git
```

★ Show

```
git status
```

```
git diff
```

```
git diff <ID1> <ID2>
```

```
git log
```

```
git log -p <FILE> <DIRECTORY>
```

```
git blame <FILE>
```

```
git show <ID>
```

```
git show <ID>:<FILE>
```

```
git branch
star (*) marks the current branch
```

★ Revert

```
git reset --hard
This cannot be undone!
```

```
git revert HEAD
Creates a new commit
```

```
git revert <ID>
Creates a new commit
```

```
git commit -a --amend
(after editing the broken files)
```

```
git checkout <ID> <FILE>
```

★ Update

```
git fetch
(this does not merge them)
```

```
git pull
(does a fetch followed by a merge)
```

```
git am -3 patch.mbox
In case of conflict, resolve the conflict and
git am --resolved
```

★ Publish

```
git commit -a
```

```
git format-patch origin
```

```
git push
```

```
git tag v1.0
```

★ Branch

```
git checkout <BRANCH>
```

```
git checkout <BRANCH2>
git merge <BRANCH1>
```

```
git branch <BRANCH>
```

```
git checkout -b <BRANCH> <OTHER>
```

```
git branch -d <BRANCH>
```

★ Resolve merge conflicts

```
git diff
```

```
git diff --base <FILE>
```

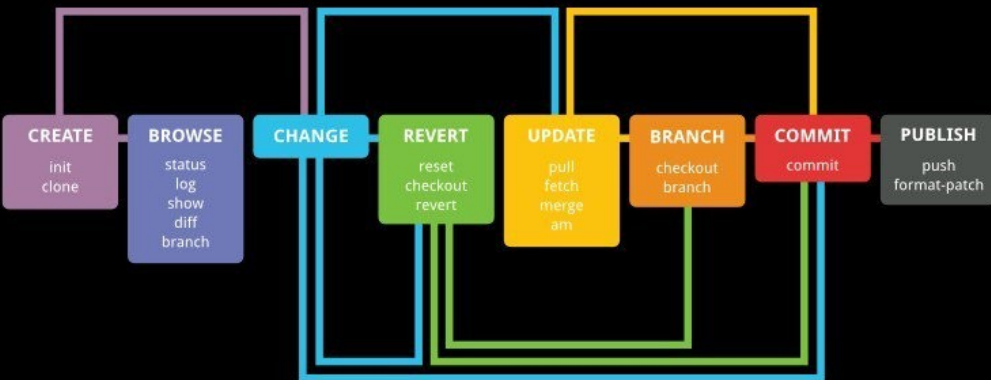
```
git diff --ours <FILE>
```

```
git diff --theirs <FILE>
```

```
git reset --hard
git rebase --skip
```

```
git add <CONFLICTING_FILE>
git rebase --continue
```

★ Workflow





Create a Repository

From scratch -- Create a new local repository

```
$ git init [project name]
```

Download from an existing repository

```
$ git clone my_url
```

Observe your Repository

List new or modified files not yet committed

```
$ git status
```

Show the changes to files not yet staged

```
$ git diff
```

Show the changes to staged files

```
$ git diff --cached
```

Show all staged and unstaged file changes

```
$ git diff HEAD
```

Show the changes between two commit ids

```
$ git diff commit1 commit2
```

List the change dates and authors for a file

```
$ git blame [file]
```

Show the file changes for a commit id and/or file

```
$ git show [commit]:[file]
```

Show full change history

```
$ git log
```

Show change history for file/directory including diffs

```
$ git log -p [file/directory]
```

Working with Branches

List all local branches

```
$ git branch
```

List all branches, local and remote

```
$ git branch -av
```

Switch to a branch, my_branch, and update working directory

```
$ git checkout my_branch
```

Create a new branch called new_branch

```
$ git branch new_branch
```

Delete the branch called my_branch

```
$ git branch -d my_branch
```

Merge branch_a into branch_b

```
$ git checkout branch_b
```

```
$ git merge branch_a
```

Tag the current commit

```
$ git tag my_tag
```

Make a change

Stages the file, ready for commit

```
$ git add [file]
```

Stage all changed files, ready for commit

```
$ git add .
```

Commit all staged files to versioned history

```
$ git commit -m "commit message"
```

Commit all your tracked files to versioned history

```
$ git commit -am "commit message"
```

Unstages file, keeping the file changes

```
$ git reset [file]
```

Revert everything to the last commit

```
$ git reset --hard
```

Synchronize

Get the latest changes from origin (no merge)

```
$ git fetch
```

Fetch the latest changes from origin and merge

```
$ git pull
```

Fetch the latest changes from origin and rebase

```
$ git pull --rebase
```

Push local changes to the origin

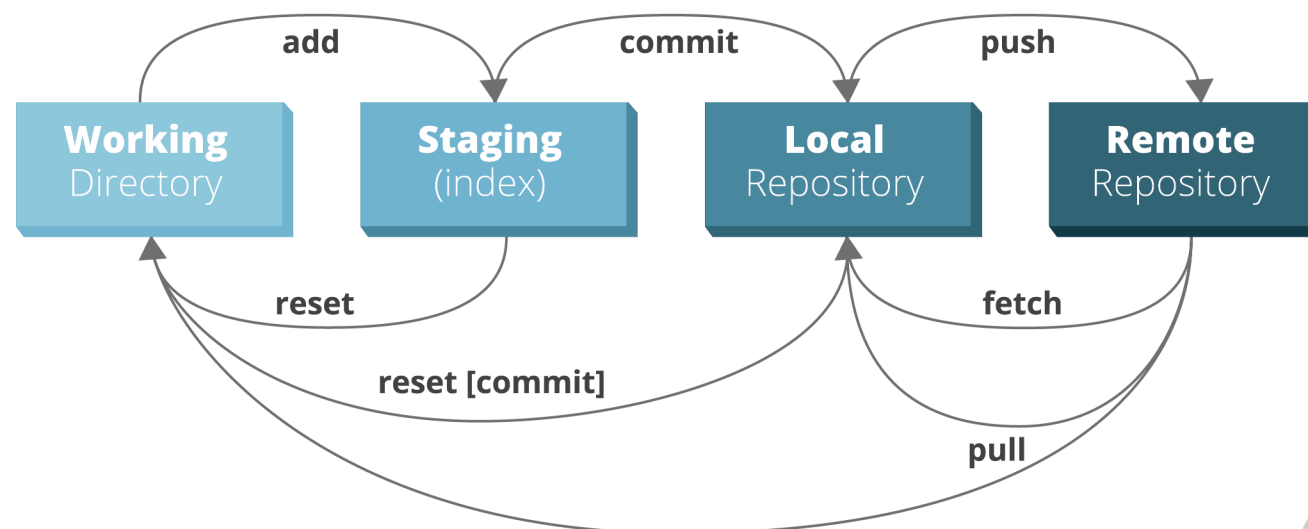
```
$ git push
```

Finally!

When in doubt, use git help

```
$ git command --help
```

Or visit <https://training.github.com/> for official GitHub training.



GitLab

- Web-приложение и система управления репозиториями кода для Git
- Возможности:
 - организация публичных и частных репозиториев;
 - управление правами, группами;
 - импорт проектов, в том числе из GitHub;
 - вики;
 - API;
 - доска идей и задач;
 - лейблы, вехи, шаблоны, поиск;
 - комментирование, объединение;
 - свой CI/CD с поддержкой DevOps;
 - отслеживание изменений и прогресса;
 - отслеживание времени;

Регистрация на <https://bmstu.codes>

- Регистрация
- Выслать никнейм (возможно не понадобится)

Получение открытого ключа SSH

- Необходимо для работы с удалённым репозиторием без ввода пароля
- Проверка существования открытого ключа SSH:
 - каталог «.ssh»,
 - файл «id_rsa.pub»;
 - содержимое этого файла и есть открытый ключ.
- Генерация открытого ключа SSH:
 - ssh-keygen

Непрерывная интеграция (CI)

- **Непрерывная интеграция (CI)** — практика разработки программного обеспечения, которая заключается в постоянном слиянии рабочих копий в общую основную ветвь разработки (до нескольких раз в день) и выполнении частых автоматизированных сборок проекта для скорейшего выявления потенциальных дефектов и решения интеграционных проблем.

Непрерывная интеграция (CI)

- В обычном проекте, где над разными частями системы разработчики трудятся независимо, стадия интеграции является заключительной. Она может непредсказуемо задержать окончание работ.
- Переход к непрерывной интеграции позволяет снизить трудоёмкость интеграции и сделать её более предсказуемой за счёт наиболее раннего обнаружения и устранения ошибок и противоречий, но основным преимуществом является сокращение стоимости исправления дефекта, за счёт раннего его выявления.

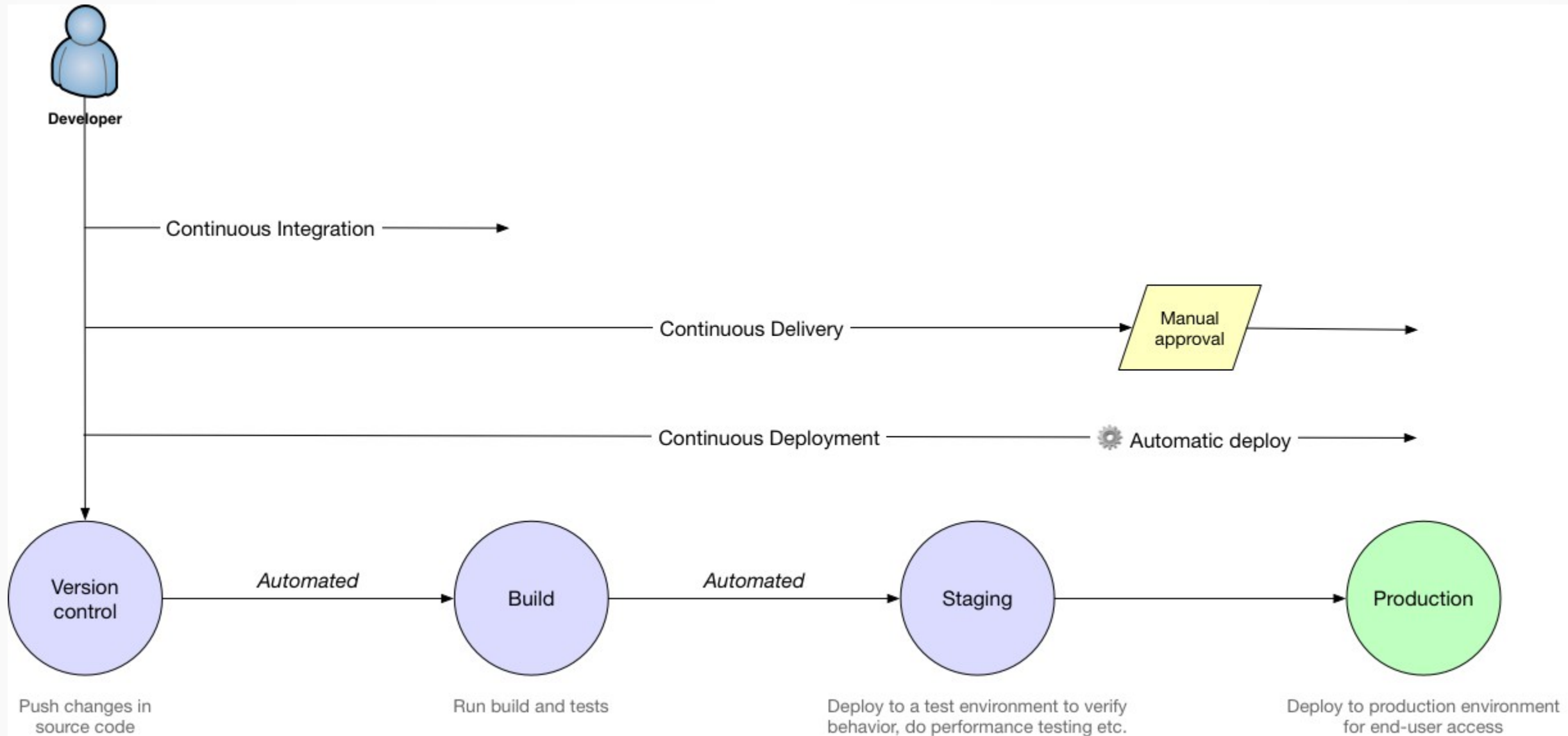
Непрерывная доставка (CD)

- **Непрерывная доставка** (англ. Continuous delivery или CD, или CDE) — это подход к разработке программного обеспечения, при котором программное обеспечение производится короткими итерациями, гарантируя, что ПО является стабильным и может быть передано в эксплуатацию в любое время.
- Передача его происходит вручную.
- Подход позволяет уменьшить стоимость, время и риски внесения изменений путём более частых мелких обновлений в продакшн-приложение.

Непрерывное развёртывание (CD)

- **Непрерывное развёртывание** (англ. Continuous deployment или CD) — это подход к разработке программного обеспечения, при котором все изменения, вносимые в исходный код, автоматически развёртываются в продакшен, без явной отмашки от разработчика. Как правило, задача разработчика сводится к проверке запроса на включение (pull request) от коллеги и к информированию команды о результатах всех важных событий.
- Непрерывное развёртывание требует, чтобы в команде существовала отлаженная культура мониторинга, все умели держать руку на пульсе и быстро восстанавливать систему.

CI/CD/CD



Вопросы?