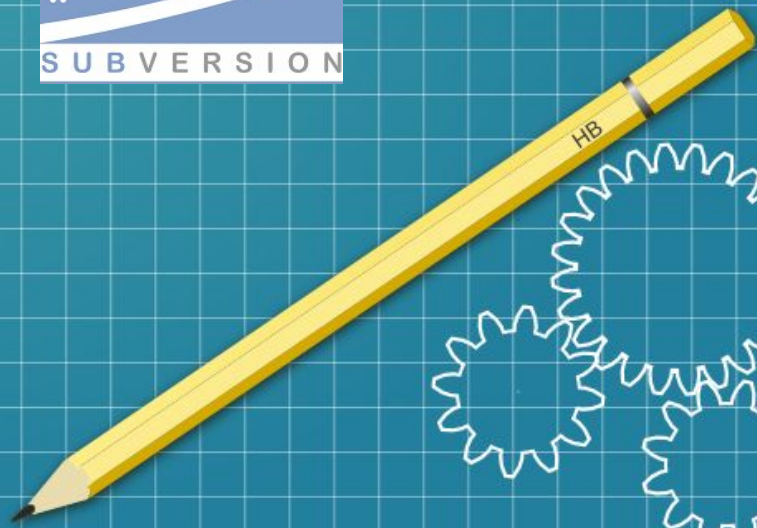
Системы контроля версий

IT-STEP



Система контроля версий VCS

Version Control System

- Хранение истории изменений
- Возможность вернуться к более ранней версии
- Возможность разрабатывать проект в команде
- Отслеживать, кто и какой участок кода написал
- Возможность обмениваться кодом

Виды VCS

1. Централизованные

Более старый подход, пример: SVN (subversion), Perforce

2. Распределённые

Современный подход — Git, Mercurial

Централизованные VCS

Есть единое хранилище всех данных — это **центральный репозиторий**

Все изменения сохраняются на этот центральный репозиторий.

Обмен кодом осуществляется через единое хранилище данных

Распределённые VCS

Нет центрального репозитория.

Есть удалённый репозиторий, но вся история изменения хранится у вас на локальном компьютере и можете в любой момент обносить её до версии актуальной.

Более гибкая разработка за счёт системы ветвей

Удалённые репозитории

Это модификации проекта, которые хранятся в интернете или ещё где-то в сети. Их может быть несколько, каждый из которых, как правило, доступен для вас либо только на чтение, либо на чтение и запись.

Что же такое система контроля версий?

- Это такая программа, которая позволяет хранить всю историю изменений, которые вы вносили в свой проект

Понятие **commit**

- Это пакет изменений, которые вы внесли в ваш проект, например: удаление файла, добавление файла или изменения уже существующих файлов. Представляем commit, как контрольную точку внесённых изменений

Понятие **commit**

- Commit не должен содержать огромное число изменённых файлов, чтобы иметь возможность оптимально **откатить** внесённые изменения в случае ошибки. В commit лучше вносить одну таску, одну фичу

Шаги для установки

Переходим на сайт https://git-scm.com/ скачиваем и устанавливаем программу

Заходим в терминал и проверяем успешно ли установлен git. *git --version*

Создаём пустой проект или в существующем

Выполняем *git init*. Создаётся папка .git (скрыта ls -force)

git status — выяснение текущего статуса

git add название файла или .

git commit -m «good comment» фиксация версии

Каждый commit имеет hash — имя объекта

Игнорирование файлов

Создаём **.gitignore**

.vs

/errors для папок

Нужно сам файл .gitignore нужно добавить в отслеживание

Работа с ветками

git branch текущая ветка

git branch name_branch создание ветки

git branch -D name_branch удаление ветки

git checkout name_branch переключение между ветками

git checkout -b name_branch создать и переключиться

git merge name_branch слияние веток

git rebase name_branch слияние веток

Работа с github

Регистрируемся на https://github.com/

Создаём репозиторий

Для соединения локального и внешнего репозитория делаем так:

git config --global user.name «Name Your»

git config--global user.email «your email»

git remote add origin http://address_repository

git push -u origin main

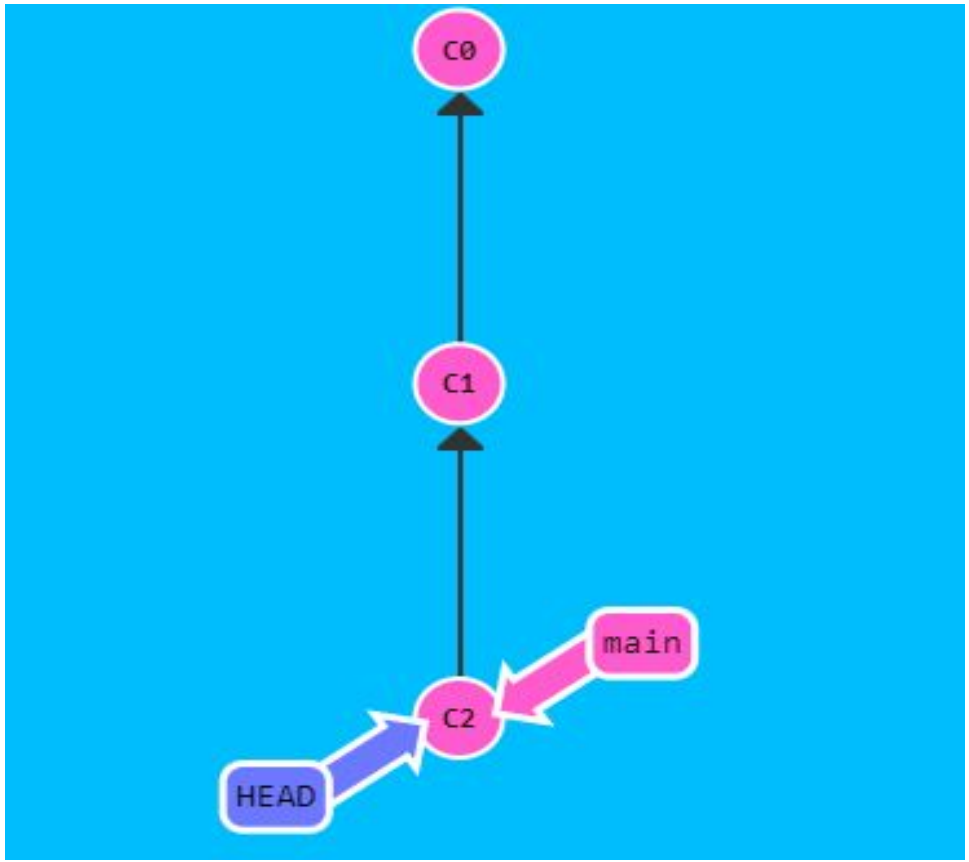
git pull забрать все последние изменения с сервера (обязательно делаем в начале рабочего дня)

git remote set-url origin http

HEAD

- Это псевдоним (второе символическое имя) текущего выбранного коммита.
- Он всегда указывает на последний коммит вашего локального дерева
- Обычно HEAD указывает на имя ветки

HEAD



Detaching HEAD

- Отделение HEAD означает, что можно присвоить его не ветке, а конкретно выбранному коммиту
- *git checkout hash_commit*

Как перемещать HEAD?

- git checkout HEAD~n
- git checkout HEAD^

- ^ перемещение на 1 коммит назад
- ~n переместит указатель HEAD на n коммитов назад, где n — целое число

Перемещение ветки

- Branch forcing

- git branch **-f main HEAD~3**

- Принудительно переместит ветку main на три родительских коммита назад от HEAD

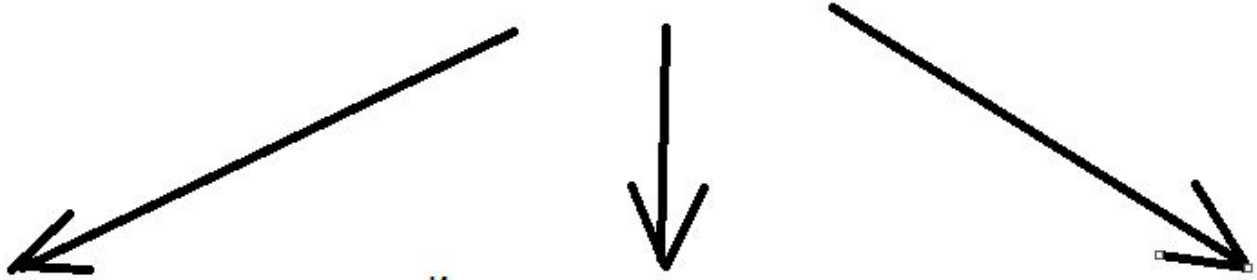
- Главная цель, для которой используются относительные ссылки это перемещение веток по истории коммитов

Отмена изменений

- ***git reset*** - отменяет изменения, перенося ветку на более ранний коммит, как-будто новых коммитов и не было (только для **локальных** репозиториев)

- ***git revert*** отменяет изменения в удалённом репозитории
- *Пример: git revert HEAD^*

Области Git

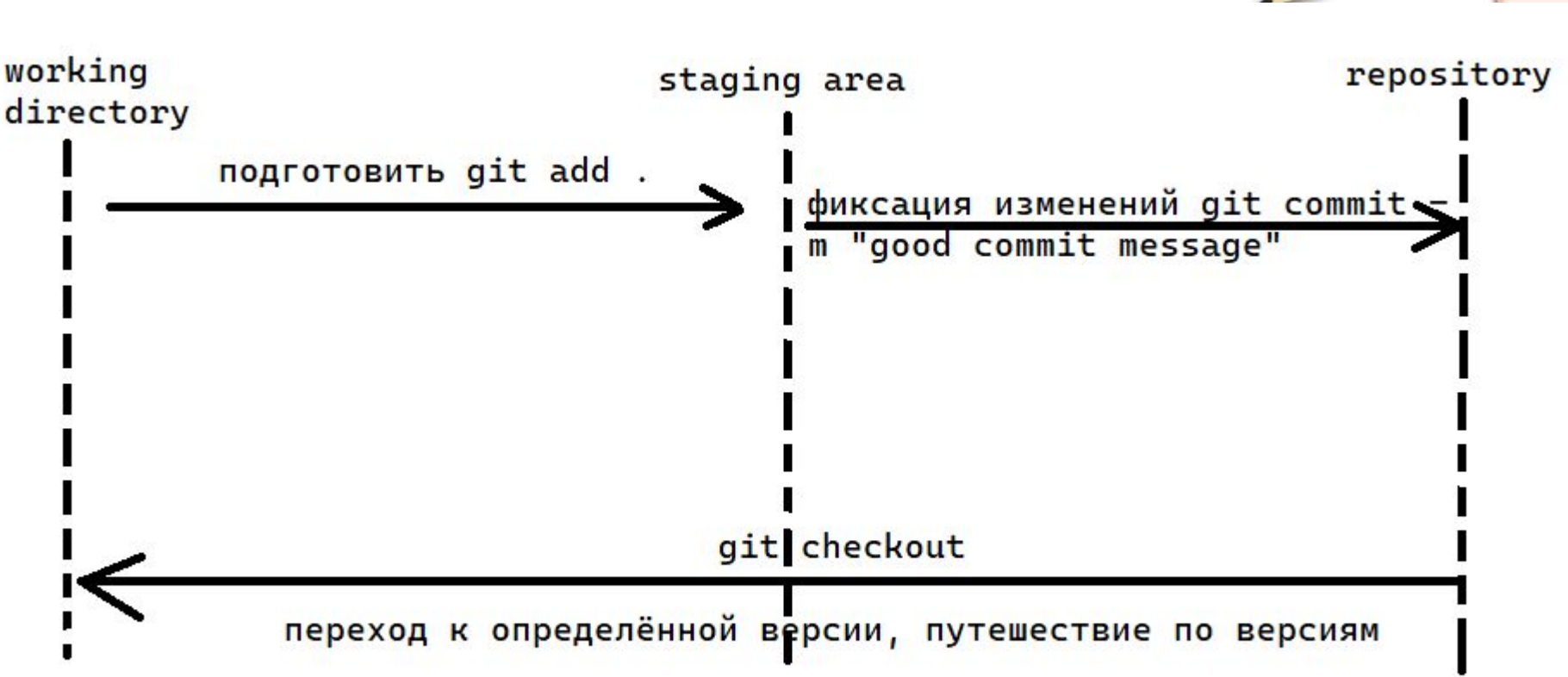


Рабочая директория
working directory
(все файлы, которые
есть в нашей рабочей
директории)

Индекс
Staging area
(Подготовленные для
коммита файлы green)

Репозиторий
Repository
(папка .git является
скрытой папкой,
objects)

Для каждой версии файла GIT создаёт объект и отличаются они по хешу



Файлы, по статусам отслеживания

Untracked
(неотслеживаемый,
новый файл)

Staged
(подготовленный,
после команды git
add, перенесён в
index area)

Unmodified
(немодифицированный,
версии файлов
совпадают)

Modified
(модифицированный,
находится в рабочей
директории)

Типы объектов в GIT (.git/objects)

Blob
(для нас просто файл)

Tree
(для нас просто папка)

Commit

Annotated Tag
(пометка важных моментов в истории)

```
git tag -a v1.0 -m "my version 1.0"
git tag
git show v1.0
```

Из чего состоит коммит?

- name, email автора
- Описание
- Ссылки на родительские коммиты

Сайт с визуализацией работы

https://learngitbranching.js.org/?locale=ru_RU

IT-STEP Система Контроля Версий