Частное учреждение профессионального образования

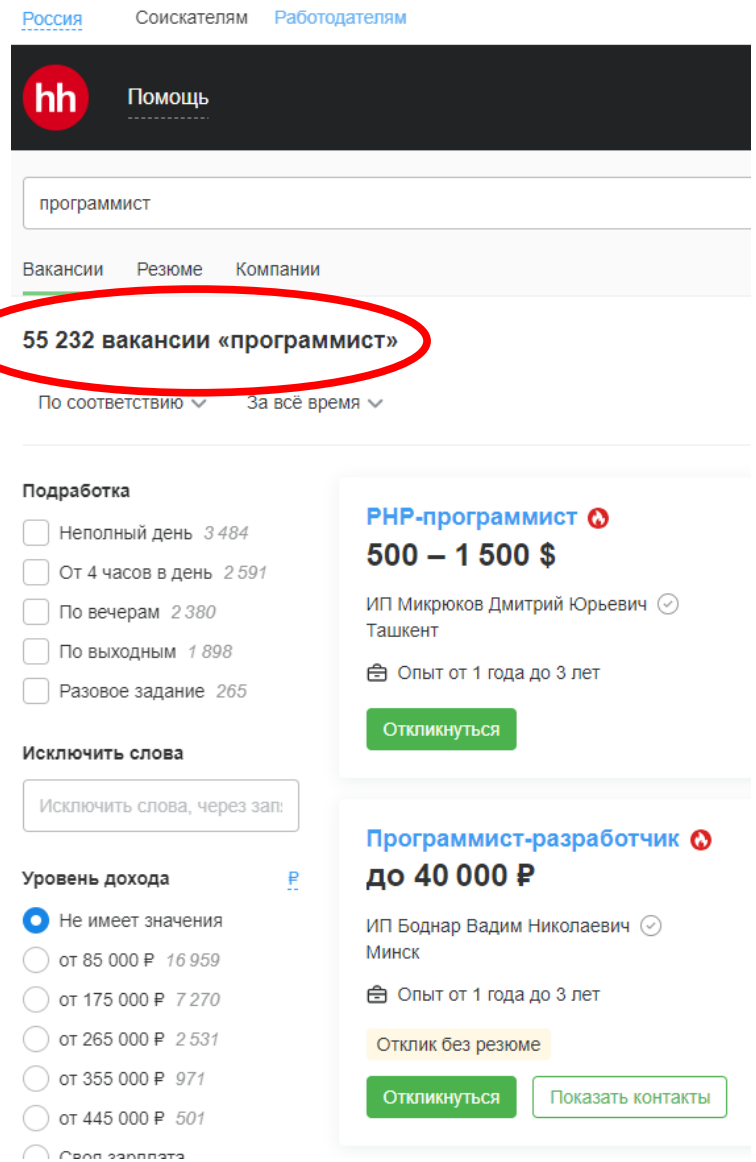
«Высшая школа предпринимательства (колледж)»

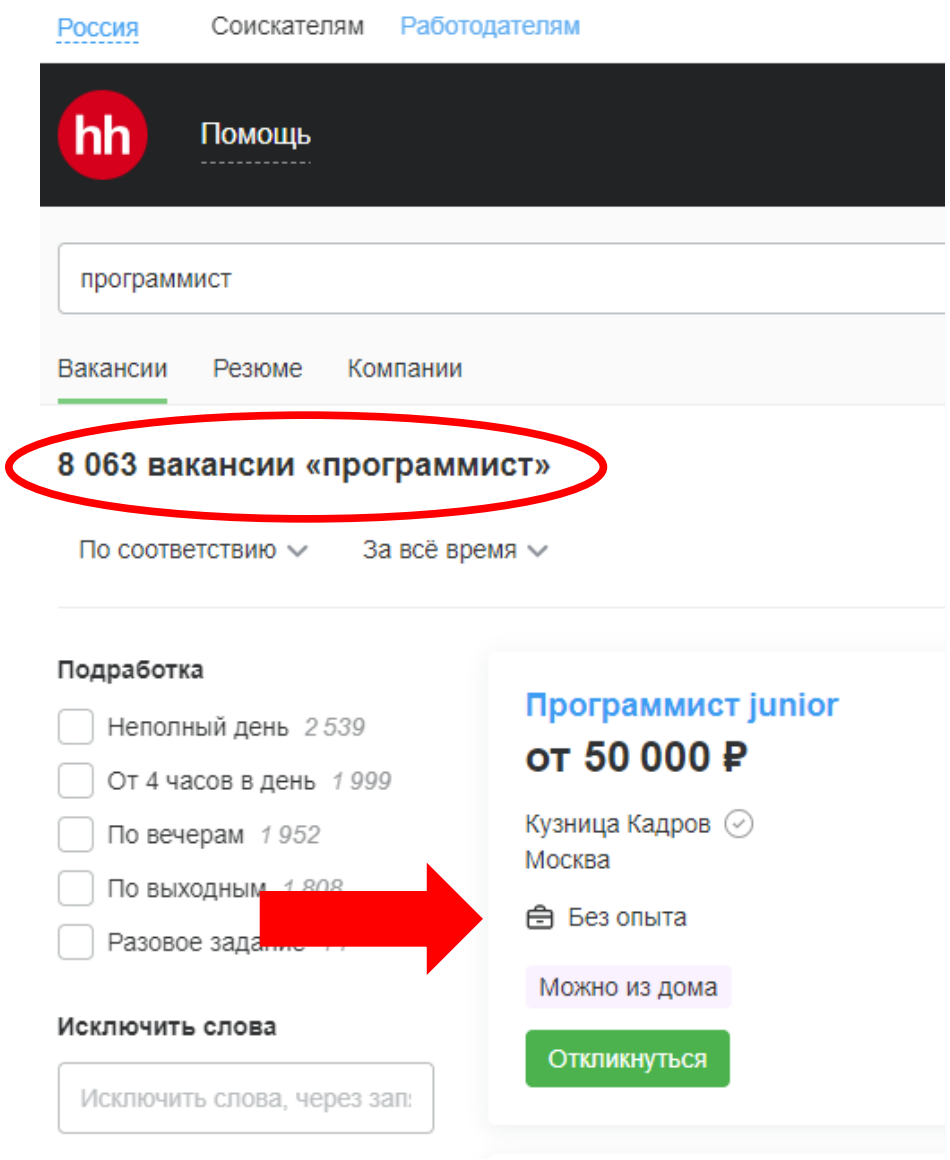
(ЧУПО «ВШП»)

Кафедра информационных технологий

Инструментальные средства разработки программного обеспечения

Для разрядки и настройки





Для разрядки и настройки

Frontend-разработчик/HTML-верстальщик

от 20 000 до 40 000 ₽ на руки

Требуемый опыт работы: не требуется
Полная занятость, полный день

Группа компаний Программный Продукт

Программный Продукт, ИТ-компания

4,2 ★★★★½ 53 отзыва

Работа только в офисе. Вариант удаленной работы не рассматривается.

Обязанности:

- Верстка по макетам
- Привязка к системе администрирования

Требования:

- Знание html, css, js
- Желание работать и развиваться в этом направлении

Условия:

- Работа над крупными проектами
- Офис в центре города
- Работа в офисе с 10 до 19 с перерывом на обед
- Трудоустройство по ТК РФ
- ЗП по результатам собеседования

Предпочтение отдается начинающим специалистам с логическим складом ума и быстрой обучаемостью. Если у Вас мало опыта в верстке, но есть большое желание развиваться в этом направлении, то смело откликайтесь на вакансию.

Что такое «система контроля версий»?

Система контроля версий

Система контроля версий (от англ. Version Control System, VCS) - это система, записывающая изменения в файл или набор файлов в течение времени

Какие виды VCS бывают?

Виды VCS

- Локальные системы контроля версий
- Централизованные системы контроля версий
- Распределённые системы контроля версий

Повторим термины

- Git или Гит
- Репозиторий Git
- Локальный репозиторий
- Удалённый репозиторий
- Форк (Fork)
- Клонирование (Clone)
- Ветка (Branch)
- Мастер (Master) или Мейн (Main)
- Коммит (Commit)
- Мёрдж (Merge)

Разберем основные команды Git

git init

git status

git add

git commit

git log

git init

Эта команда создаёт в текущем каталоге новый подкаталог с именем **.git**, содержащий все необходимые файлы репозитория — структуру Git репозитория.

На этом этапе ваш проект ещё не находится под версионным контролем.

Состояния файлов – что такое?

Состояния файлов

Отслеживаемые файлы — это те файлы, которые были в последнем снимке состояния проекта; они могут быть неизменёнными, изменёнными или подготовленными к коммиту. Если кратко, то отслеживаемые файлы — это те файлы, о которых знает Git.

Неотслеживаемые файлы — это всё остальное, любые файлы в вашем рабочем каталоге, которые не входили в ваш последний снимок состояния и не подготовлены к коммиту.

git add

Если вы хотите добавить под версионный контроль существующие файлы (в отличие от пустого каталога), вам стоит добавить их в индекс и осуществить первый коммит изменений.

Добавить файлы в индекс репозитория можно запустив команду **git add**, указав индексируемые файлы, а затем выполнив **git commit**

git status

```
git status
On branch master
Your branch is up-to-date with 'origin/master'.

nothing to commit, working tree clean
```

Это означает, что у вас чистый рабочий каталог, в нём нет отслеживаемых изменённых файлов.
Git также не обнаружил **неотслеживаемых** файлов, иначе они бы были перечислены здесь.
Команда сообщает вам на какой ветке вы находитесь и сообщает вам, что она не расходится с веткой на сервере

git commit

Коммит сохраняет снимок состояния вашего индекса.
Всё, что вы не проиндексировали, так и висит в рабочем каталоге как изменённое.
Можете сделать ещё один **коммит**, чтобы добавить эти изменения в **репозиторий**.
Каждый раз, когда вы делаете **коммит**, вы сохраняете снимок состояния вашего проекта, который позже вы можете восстановить или с которым можно сравнить текущее состояние

git log

Вы создали несколько **коммитов** или же **клонировали репозиторий** с уже существующей историей коммитов, вам понадобится возможность посмотреть что было сделано — историю **коммитов**.
Одним из основных и наиболее мощных инструментов для этого является команда **git log**

По умолчанию **git log** перечисляет **коммиты**, сделанные в **репозитории** в обратном к хронологическому порядке — последние **коммиты** находятся вверху

Разберем новые команды Git

git branch

git checkout

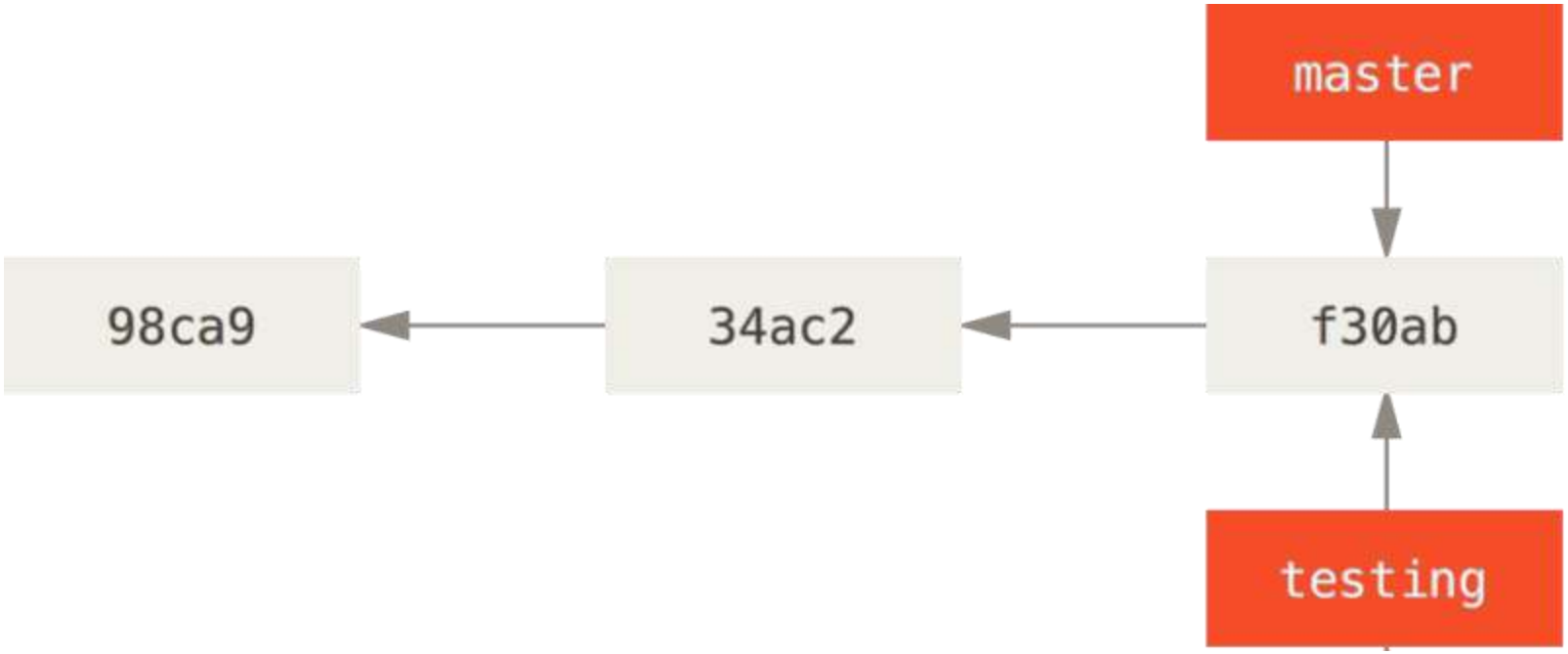
git merge

git branch

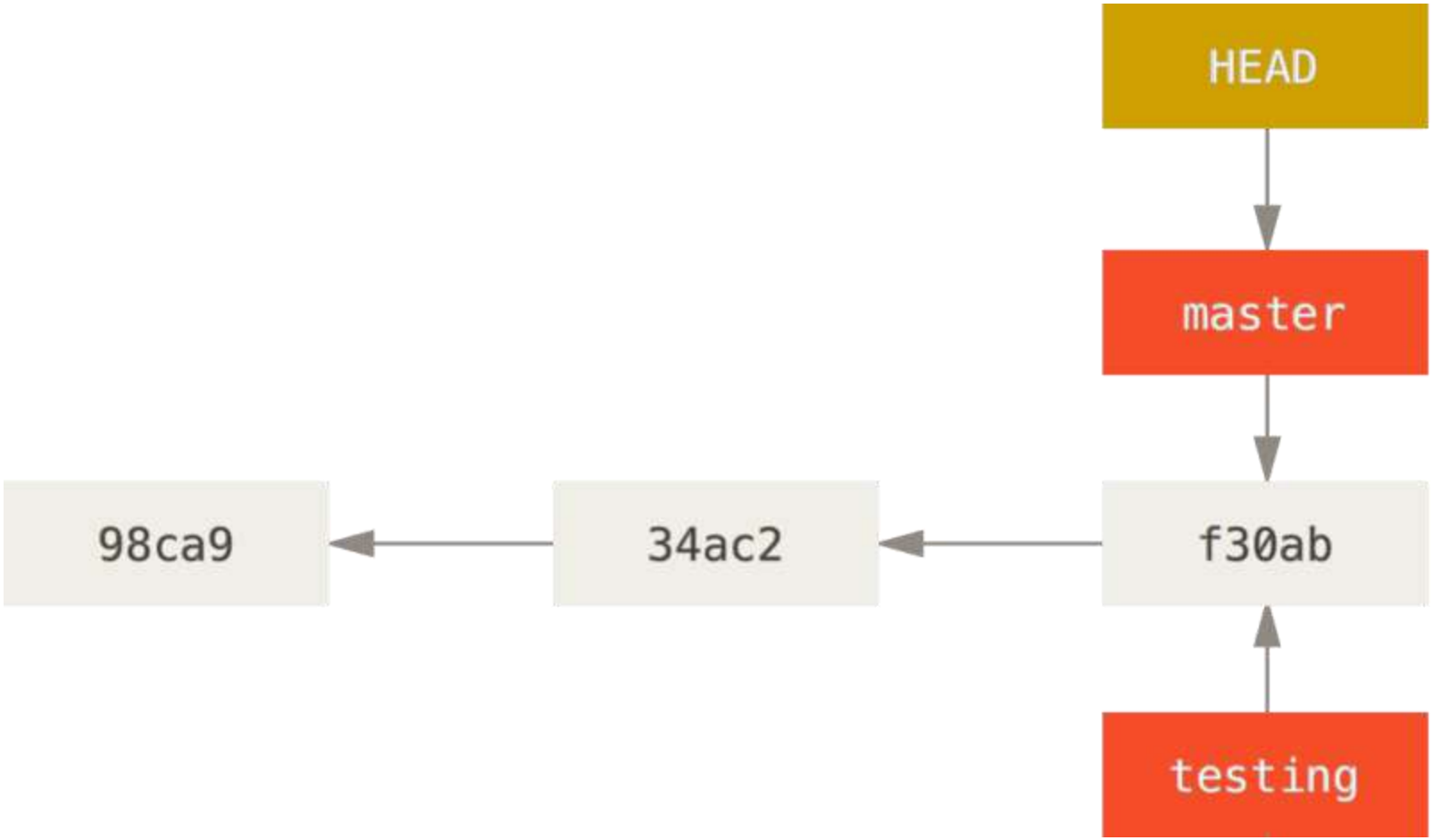
Команда **git branch** позволяет создавать, просматривать, переименовывать и удалять ветки. Она **не дает** возможности переключаться между ветками или выполнять слияние разветвленной истории. Именно поэтому команда **git branch** тесно связана с командами **git checkout** и **git merge**

Ветки — это просто указатели на **коммиты**. Когда вы создаете ветку, Git просто создает новый указатель. Репозиторий при этом никак не изменяется.

git branch



git branch

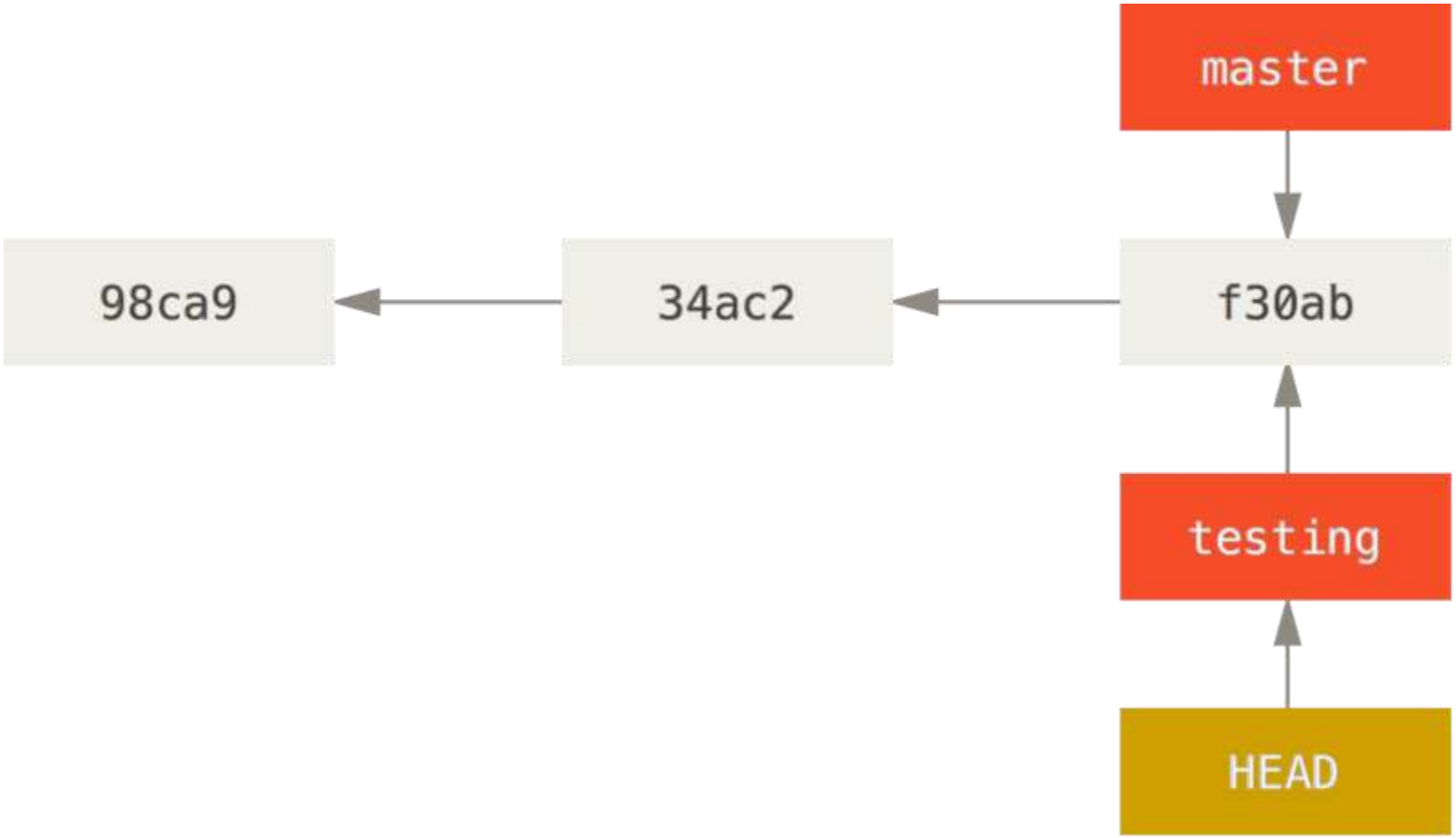


git checkout

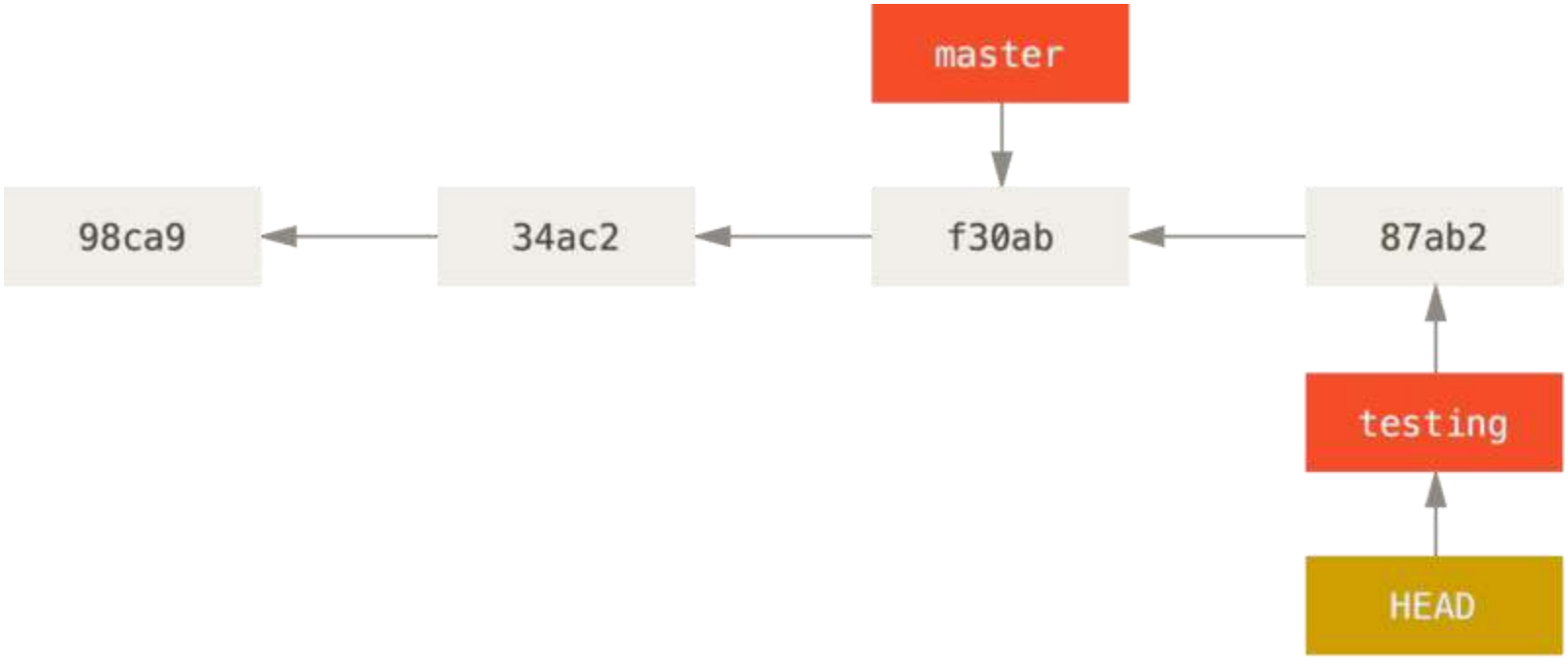
Команда **git checkout** позволяет перемещаться между ветками, созданными командой **git branch**. При переключении ветки происходит обновление файлов в рабочем каталоге в соответствии с версией, хранящейся в этой ветке, а Git начинает записывать все новые коммиты в этой ветке

Команду **git checkout** можно спутать с командой **git clone**. Разница между этими двумя командами заключается в том, что при клонировании (**clone**) выполняется извлечение кода из удаленного репозитория, тогда как при переключении (**checkout**) происходит переключение между версиями кода, который уже находится в локальном репозитории.

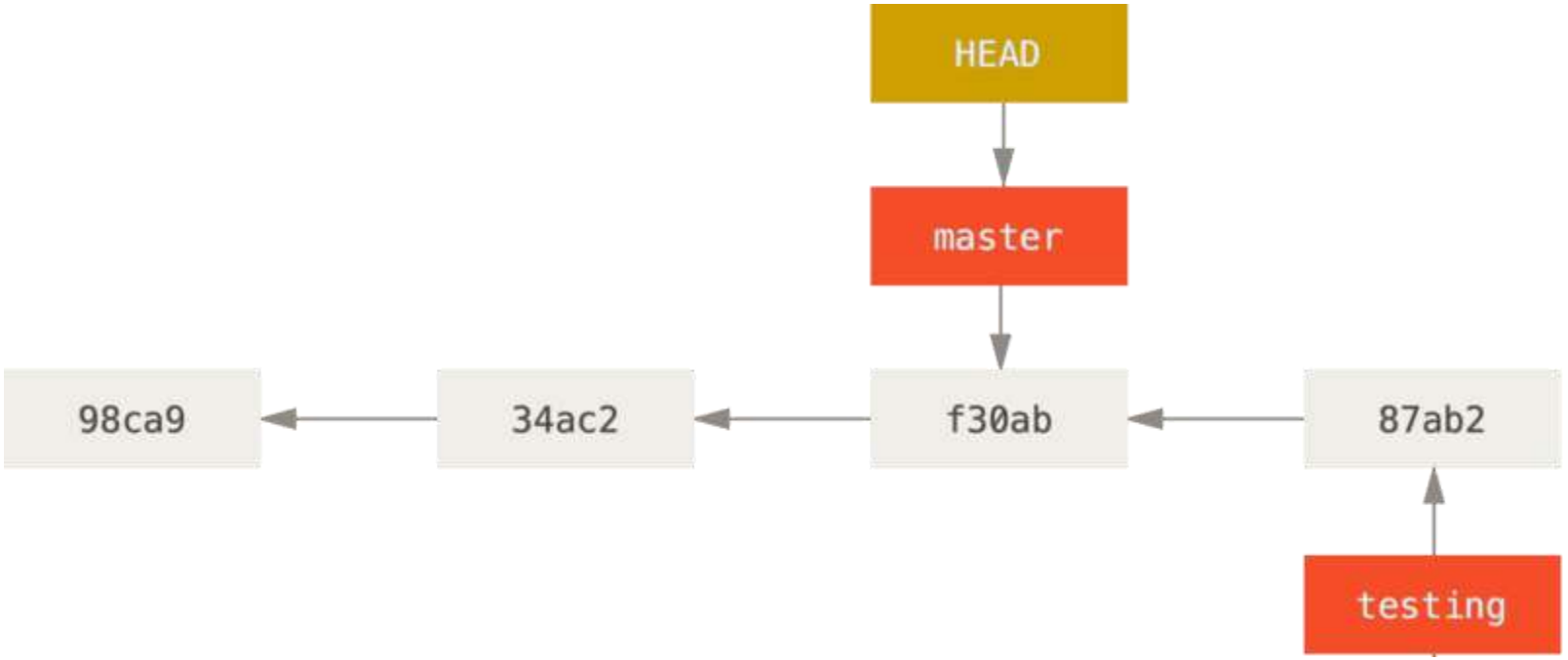
git checkout



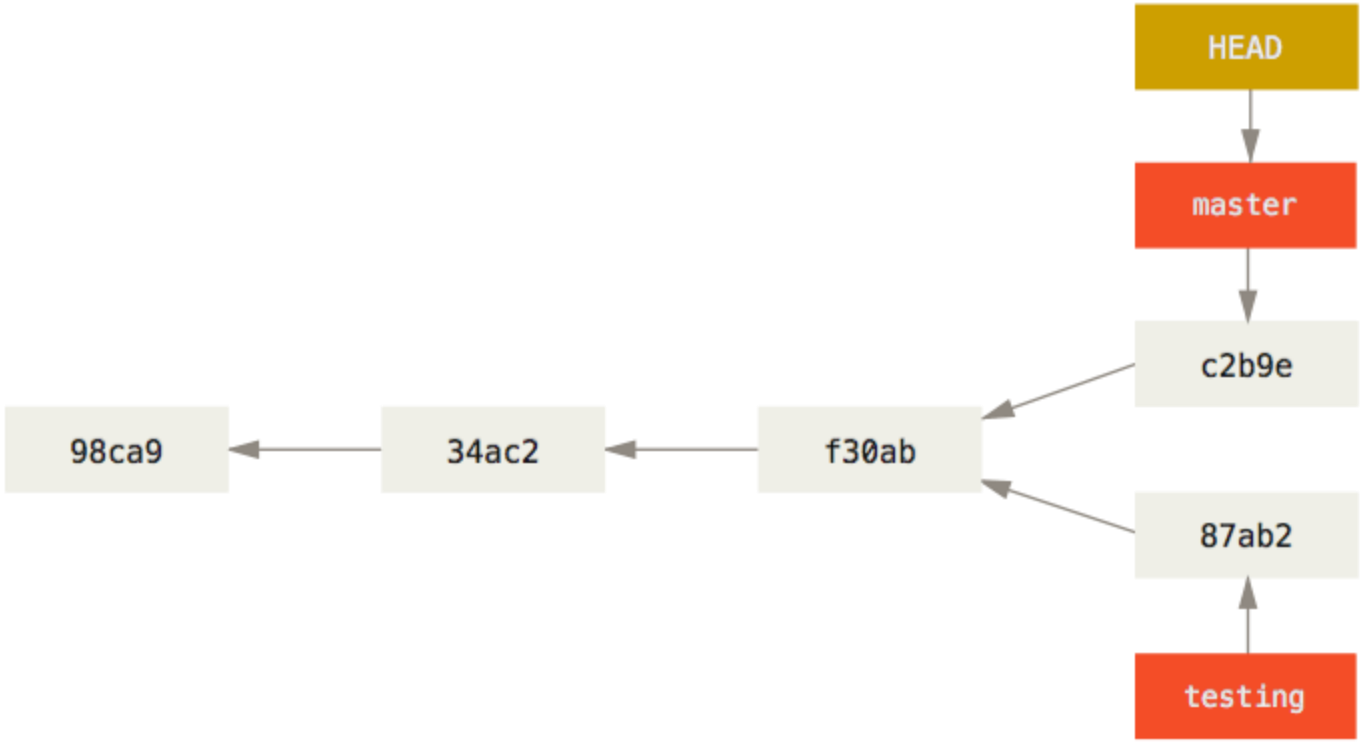
git checkout



git checkout



git checkout



git merge

Команда **git merge** объединяет несколько последовательностей коммитов в общую историю. Чаще всего команду **git merge** используют для объединения двух веток.

В таких случаях команда **git merge** принимает два указателя на коммиты (обычно последние в ветке) и находит общий для них родительский коммит. Затем Git создает коммит слияния, в котором объединяются изменения из обеих последовательностей, выбранных к слиянию

git merge

 # Start a new feature
git checkout -b new-feature main
 # Edit some files
git add <file>
git commit -m "feat: start a feature"
 # Edit some files
git add <file>
git commit -m "feat: finish a feature"
 # Merge in the new-feature branch
git checkout main
git merge new-feature
git branch -d new-feature

Требования к именам коммитов

init: - используется для начала проекта/таска.
init: start task

feat: - это реализованная новая функциональность из технического задания.
feat: add basic page
feat: add button

fix: - исправил ошибку в ранее реализованной функциональности.
fix: change layout to fix bugs

1. Установите git (например, от сюда http://git-scm.com/).
2. Создайте репозиторий.
3. Создайте в репозитории текстовый файл с текстом (2-3 абзаца текста).
4. Разбейте его создание на несколько этапов (добавление 2 абзаца). Коммитьте изменения (2-3 коммита)
5. Создайте две ветки.
6. В первой ветке «поработайте» над текстом (добавьте 2 абзаца).
7. Во второй уберите содержания (удалите 1 абзац)
8. Не забывайте разбивать эту «работу» на коммиты.
9. Слейте все изменения назад в ветку master.
10. Удалите вспомогательные ветки.
11. Изучите команду git log
12. Сделайте скриншот вывод команды git log, который подтвердил бы, что Вы сделали пункты 1-10
13. Сделайте архив из папки репозитория и принесите на занятие

Дополнительный материал

gosuslugi.ru/futurecode

ru.hexlet.io/courses/intro_to_git

practicum.yandex.ru/git-basics