

Version Control with Git

Part 1

ABOUT THIS COURSE

Introduction

The first thing a software engineer need to know is how to collaborate with colleagues. And Version Control systems like Git are tools that everybody in modern world need to know, for reliable and secure development and cooperation.

You'll learn from scratch how to use Git in production, how to keep your history clean and make collaboration pleasant for everybody in your team, how to prepare your project to releases and support versioning and other best practices of using such VCS as Git.

Summary of this course

- What is Version Control System.
- Principles of Git. Definitions, simple operations, using remote repo (Github).
- Problems of collaboration: merge conflicts and keep history clear.
- Branches and Tags: versioning your project.
- Contributing in real world. Clone, Fork, Pull Request.
- Use Git for 100%: CLI tools, GUI interfaces and IDE integration.
- Advanced topics: Rebase, Cherry-Pick, Remotes, Bundles, Tags, Signatures, Rerere, Hooks.
- and **a lot of practice!**

Agenda: Part 1

- Git overview
- Git config and aliases
- Working directory
 - Add
 - Commit
 - Reset and Revert
 - Clean
- Stash
- Tag
- Log
- Blame

Agenda: Part 2

- Pull/Push
- Branch
 - Merge and rerere
 - Rebase and interactive Rebase
 - Cherry-Pick
- Pull request
- Workflows
 - Centralized
 - Feature
 - Gitflow
 - Forking
- Merge vs Rebase workflows
- Remotes
- Synch options/ supported protocols

What you need

- Laptop/PC to run Git
- Git <https://git-scm.com/downloads>
- Possibility set up GitHub account
- Your favorite Text editor/File Manager/IDE
- Curiosity!

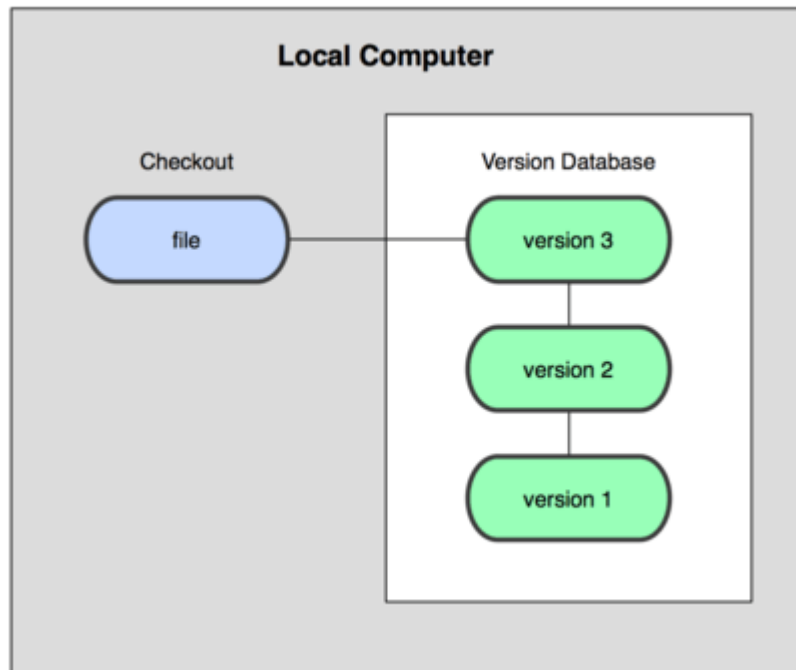
VERSION CONTROL

Version Control System

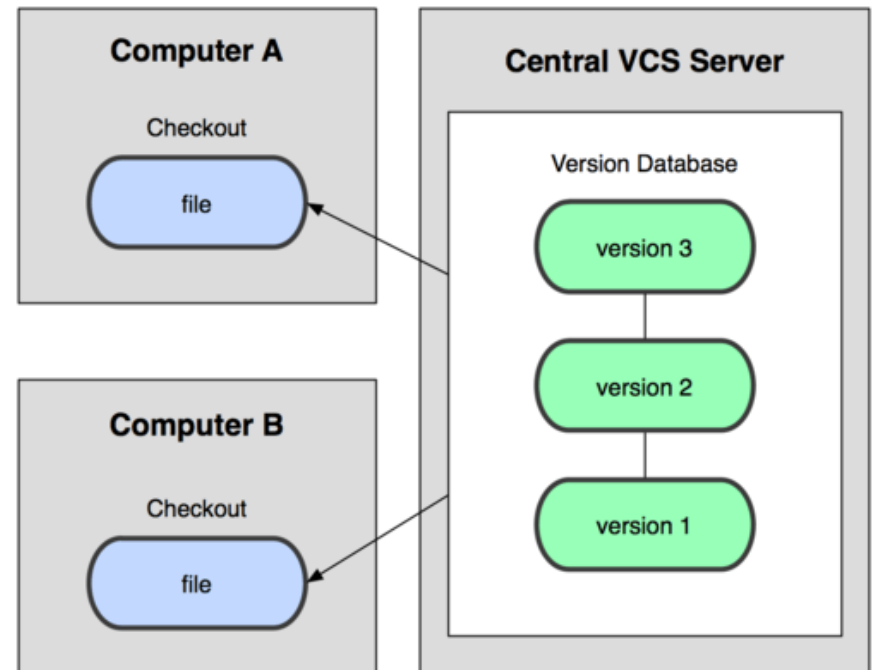
- Collaboration
- Storing versions (properly)
- Restoring previous versions
- Backups
- Understanding what happened
- Explanations for newcomers
 - <https://www.atlassian.com/git/tutorials/what-is-version-control/benefits-of-version-control>
 - <https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>

Version Control System

Local



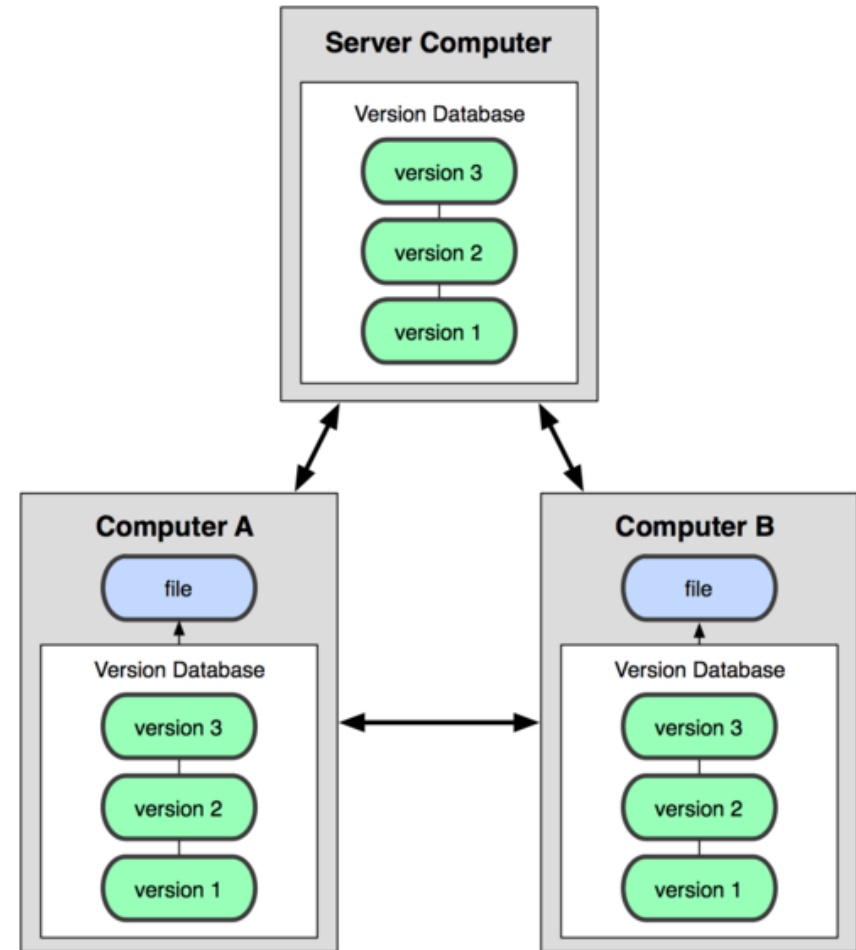
Centralized



Version Control System

Distributed

- Every machine stores a backup
- All copies are up-to-date
- Every copy contains full history log



GIT: BASIC CONCEPTS

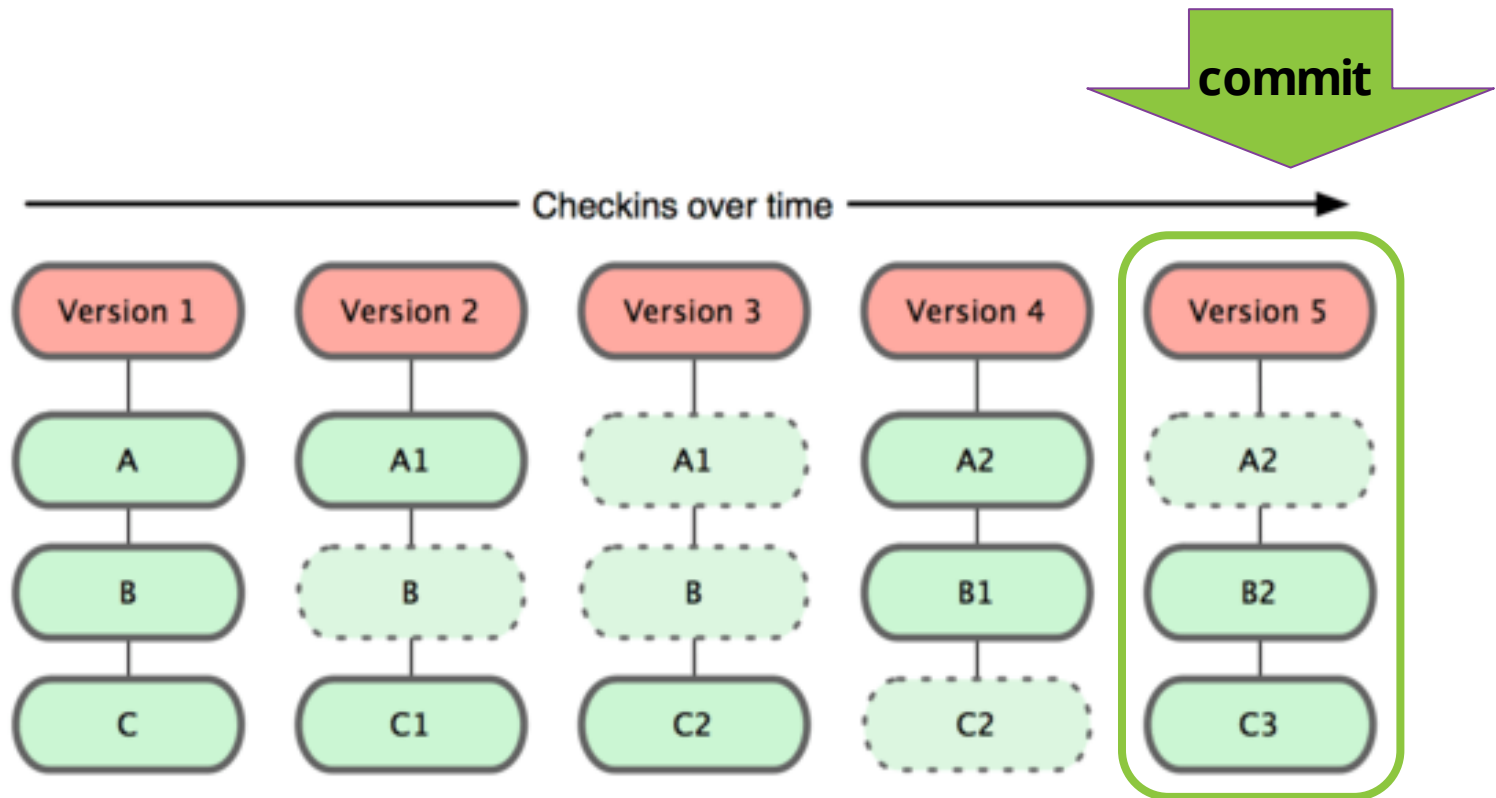
Why Git?

- Easy to use, difficult to screw up
- Distributed VCS: stable and reliable
- Fast and scalable
- Has both Shell and GUI wizards
- Integration with modern IDEs
- Lots of free VCS services
- De-facto standard
- Explanations for newcomers
 - <https://www.atlassian.com/git/tutorials/what-is-version-control/benefits-of-version-control>
 - <https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>

Definitions

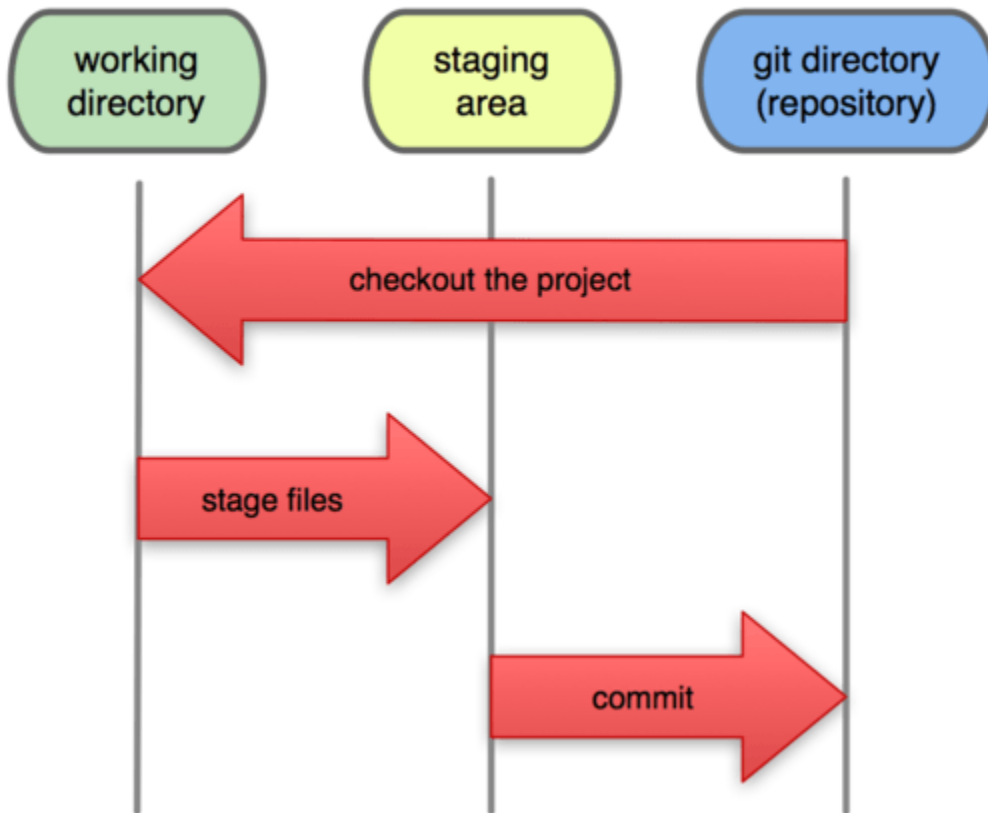
- **Blobs** - a binary representation of a file.
- **Tree objects**- can contain pointers to blobs and other tree objects.
- **Commit objects**- point to a single tree object, and contain some metadata including the commit author and any parent commits.
- **Tag objects**- point to a single commit object, can contain some metadata.
- **References** - pointers to a single object (usually a commit or tag object).

Understanding version and commits



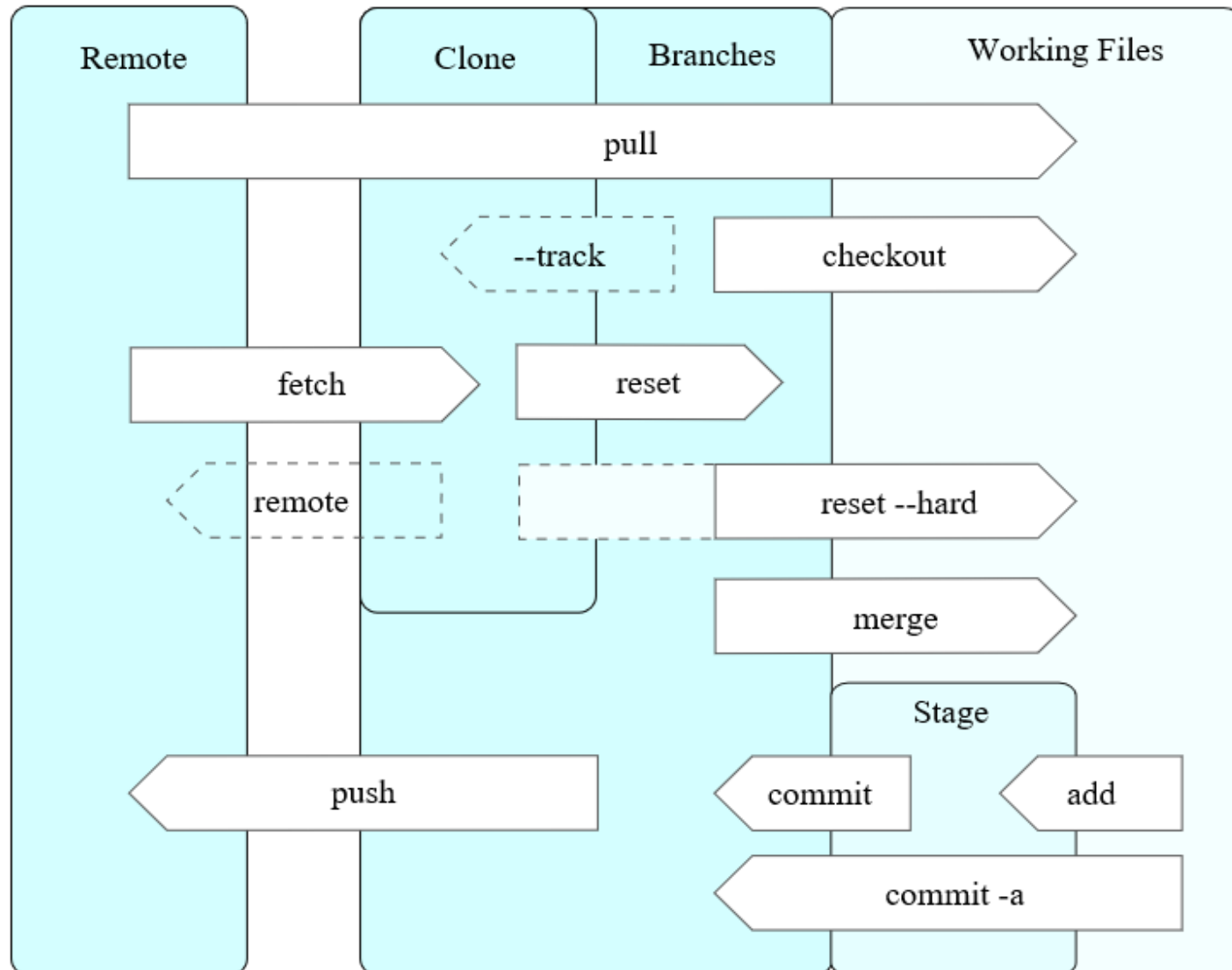
Working directory

Local Operations



- Working directory - copy of current version, extracted from repository for viewing and editing.
- **\$ git add <file>**
- Staging area - it's file with metadata about changes that will be to be fixed in repo during next commit.
- **\$ git commit**
- Git directory (repository) - all the data, metadata, history, logs, etc.

Git Operations



GIT: SETUP AND CONFIGURATION

Configuration

- Right after installation you need to configure your Git. Run
 - **\$ git config --global user.name "<your name>"**
 - **\$ git config --global user.email <your email>**
- To set up your credentials. Every commit contains info about author, and this data will be included.
- If you need help with some commands type
 - **\$ git help <command>**
 - **\$ git <command> --help**

Configuration options

- Possible configuration locations:
 - **--global** ~/.gitconfig
 - **--system** \$(prefix)/etc/gitconfig
 - **--local** .git/config
 - **--file** specific file
- Commands to try
 - **git config --list**
 - **git config --list --show-origin**
- Find out more in <https://git-scm.com/docs/git-config>

Set-up

- **git init** creates empty repo
- **git init --bare** creates empty *bare* repository
- **git clone *path*** downloads a copy of remote repo
- Explanations for newcomers
 - <https://www.atlassian.com/git/tutorials/setting-up-a-repository/git-init>

Add/Remove content

- **git add *file***
- **git rm**
- **git status**
- **git commit**
- **git log**

Ignore specific files

- You can enforce your Git to ignore some files (Git won't track changes) by creating .gitignore file with patterns of files that should be ignored.
- You'd like to ignore auto-generated files, compiled binaries, project configuration files.

```
# IntelliJ IDEA files  
  
*.iml  
.idea/  
/out/
```

Aliases

- Simplify your life in console
- Examples:

git config --global alias.*st status*

git config --global alias.*l3 "log -n 3"*

**git config --global alias.*lg "log --pretty=format:%h %s\"
--graph -n 7"***

- How to find what is available
git config --list --show-origin

Commit

- **git commit**
- **git commit -m "message"**
- **git commit --amend**

```
[master (root-commit) 9e878c2] Initial commit
2 files changed, 5 insertions(+)
create mode 100644 .gitignore
create mode 100644 readme.txt
```

Commit

- **git log**

```
commit 3be4cbbb16a4f29e0f9a3f181b18997e4f9d0aa8
Author: TDiva <tanushadomanova@gmail.com>
Date:   Mon Feb 1 01:24:27 2016 +0300
```

Initial commit

Commit log

- Let's add another commit: create file **new.txt**, use git **add new.txt** and **git commit**. Now our history log looks like:

```
commit f98414af0528c51d231a0384e15408f0ee5efaec
Author: TDivya <tanushadomanova@gmail.com>
Date:   Mon Feb 1 01:32:25 2016 +0300
```

Add new.txt

```
commit 3be4cbbb16a4f29e0f9a3f181b18997e4f9d0aa8
Author: TDivya <tanushadomanova@gmail.com>
Date:   Mon Feb 1 01:24:27 2016 +0300
```

Initial commit

- But what actually happened?

Commit log

- Let's add another commit: create file **new.txt**, use git **add new.txt** and **git commit**. Now our history log looks like:

```
commit f98414af0528c51d231a0384e15408f0ee5efaec
Author: TDivya <tanushadomanova@gmail.com>
Date:   Mon Feb 1 01:32:25 2016 +0300
```

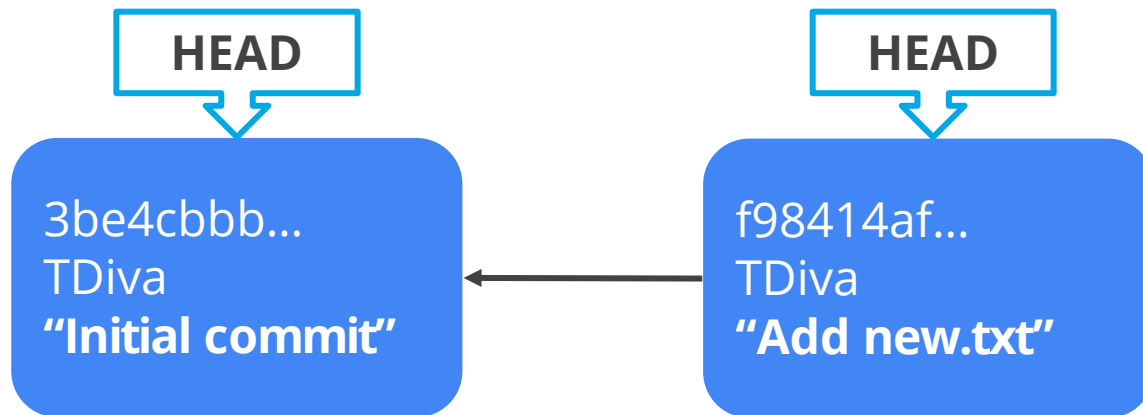
```
Add new.txt
```

```
commit 3be4cbbb16a4f29e0f9a3f181b18997e4f9d0aa8
Author: TDivya <tanushadomanova@gmail.com>
Date:   Mon Feb 1 01:24:27 2016 +0300
```

```
Initial commit
```

- But what actually happened?

Commit history



Every commit has a reference to previous one (parent).

HEAD is a link to the version you currently work on.

Now you have 2 versions of your project - initial and after adding **new.txt**, and at any moment you can easily switch you project files to one of these snapshots.

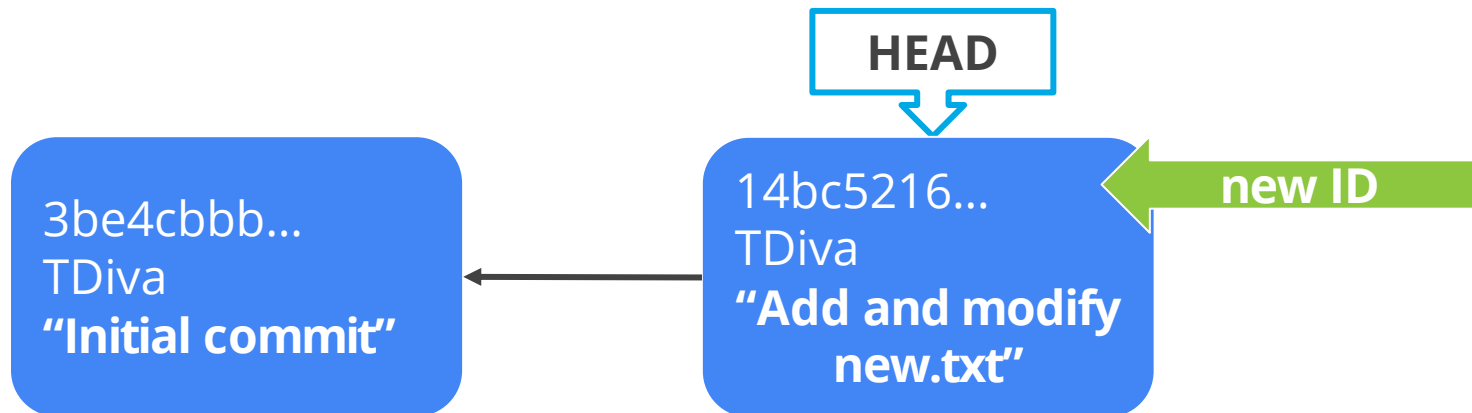
Commit modification

- If you think you need to change the latest commit, you can easily do it running

git commit --amend

- changes the latest commit in history

- After that git will take a snapshot of your file system and replace the latest commit with new one:



Commit modification: revert

- Let's assume that we want not to change latest commit but delete all the changes we made in **3be4cbbb**. What does it mean for Git?
 - *we can apply changes that annihilate our commit.*

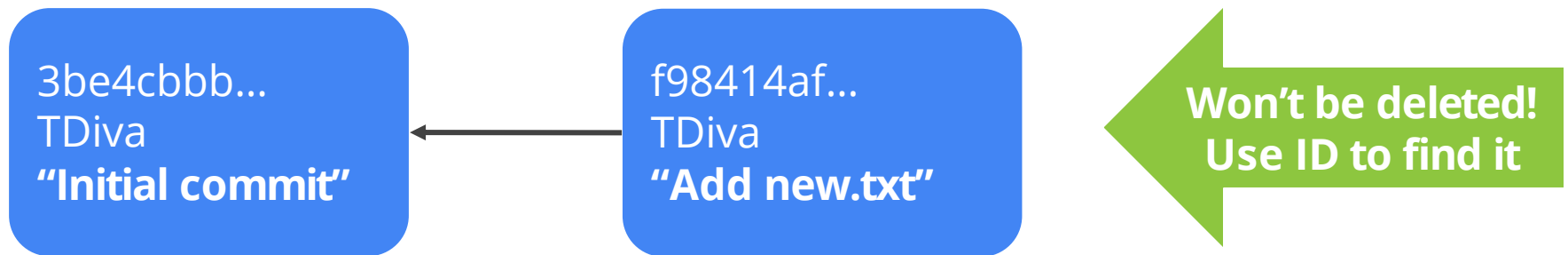
git revert <commit>

- generate a new commit that undoes all changes made in <commit>



Commit modification: restore

- Restore changes
 - **git reset <commit>**
 - moves branch pointer and HEAD, won't affect working directory
 - **git reset --hard <commit>**
 - moves branch pointer and HEAD, and updates working directory



Undo: more options

- Remove unversioned
 - **git clean -f**
 - **git clean -fdx**
 - **git clean -n**
- Undo range of changes
 - **git revert HEAD~3**
 - **git revert <commit>**
 - **git revert -n master~3..master~1**

Get revision and history

- Get specific revision
 - **git checkout <commit>**
- Find authors
 - **git blame *file***

Stash

- **git stash**
- **git stash show**
- **git stash pop**
- Explanations for newcomers
 - <https://www.atlassian.com/git/tutorials/git-stash/>

Log

- `git log`
- `git log -n 3`
- `git log --abbrev-commit`
- `git log --since="2 weeks ago" -- 1.txt`
- `git show master@{yesterday}`
- `git log --pretty=format:%h`
- `git log --pretty=format:"%h %s" --graph`
- `git rev-list --parents -n 1 2c43ae16b`
- `git rev-parse --short=7`
`28af1134efec499195b09abe79c68949f9e36ea2`
- `git show 9ce95f9ba8c933f127bd2bd7b3cfe0a75f85bd7a`
- Find out more information
 - <https://git-scm.com/book/en/v2/Git-Tools-Revision-Selection>
 - <https://git-scm.com/docs/git-rev-list>

Revision selection

- HEAD~1 - the first parent of the commit
- HEAD^n - n-th parent of the commit
- HEAD~ == HEAD^
- **..** *double dot* range of commits that are reachable from one commit but aren't reachable from another
 - Example 1:
git log master..experiment
all commits in master that aren't in experiment
 - Example 2:
git log refA..refB
git log ^refA refB
git log refB --not refA
- **...** *tripple dot* specifies all the commits that are reachable by either of two references but not by both of them
- Find out more information
 - <https://git-scm.com/book/en/v2/Git-Tools-Revision-Selection>
 - <https://git-scm.com/docs/git-rev-list>

Tags

- `git tag v0.1 -m "My version 0.1 (local)"`
- `git tag -a v0.1 -m "My annotated v 0.1"`
- `git tag -a -s vSig -m "My Signed Tag"`
- `git describe`
- `git push origin --tags`
- Find out more information
 - <https://git-scm.com/book/en/v2/Git-Basics-Tagging>
 - <https://git-scm.com/book/en/v2/Git-Tools-Signing-Your-Work>

Synchronization with the server

- **git fetch -all**
 - Get remote changes, but don't apply
- **git pull origin master**
 - Get and apply remote changes
- **git push**
 - Submit you changes

PRACTICE.

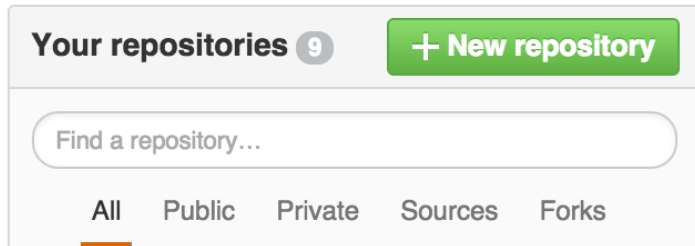
SETUP AND BASIC OPERATIONS

Get ready

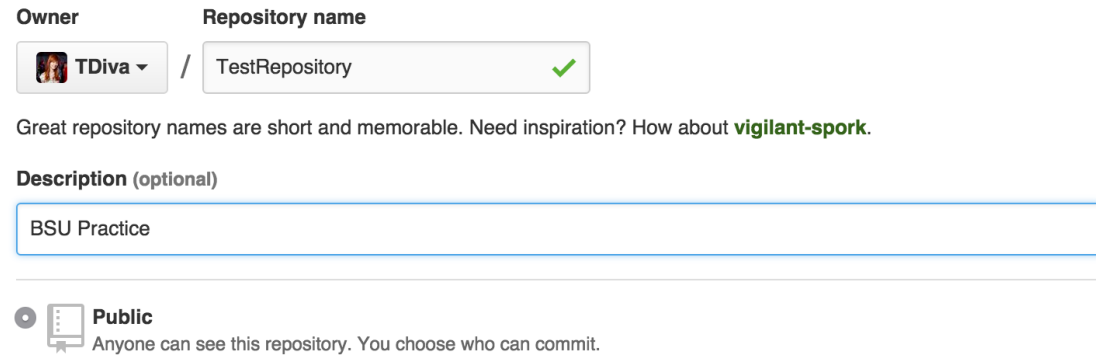
- Setup GitHub account
- Install Git client – GUI and **Console**
- For newcomers
 - <https://git-scm.com/downloads>
 - <https://www.atlassian.com/git/tutorials/install-git>

GitHub(Create)

- **GitHub is a Web-based Git repository hosting service.**
 - Create account [here](#)
 - Create a repo: + **New repository** on the left-hand pane



This screenshot shows the 'Your repositories' sidebar on GitHub. At the top, it says 'Your repositories 9' next to a green button labeled '+ New repository'. Below this is a search bar with the placeholder text 'Find a repository...'. At the bottom, there are five tabs: 'All', 'Public', 'Private', 'Sources', and 'Forks'. The 'All' tab is currently selected and highlighted with an orange underline.



This screenshot shows the 'Create new repository' form on GitHub. It has two main sections: 'Owner' and 'Repository name'. The 'Owner' section shows a dropdown menu with 'TDiva' selected. The 'Repository name' section shows a text input field with 'TestRepository' and a green checkmark icon. Below these sections, there is a line of text: 'Great repository names are short and memorable. Need inspiration? How about **vigilant-spork**.' The next section is 'Description (optional)', which has a text input field containing 'BSU Practice'. At the bottom, there is a radio button next to the word 'Public', which is selected. Below the radio button, there is a small icon of a person and the text: 'Anyone can see this repository. You choose who can commit.'

GitHub(Clone)

- Get the existing repo (instead of **git init**) from GitHub (or other service) run

\$ git clone <repo url>

The screenshot shows the GitHub interface for a repository named 'TDiva / Examples'. At the top, there are buttons for 'Unwatch' (1), 'Star' (0), and 'Fork' (0). Below this is a navigation bar with tabs for 'Code', 'Issues' (0), 'Pull requests' (0), 'Wiki', 'Pulse', 'Graphs', and 'Settings'. The main content area shows the repository description 'Stuff I've wrote for reuse — Edit'. Below this, there are statistics: '6 commits', '3 branches', '0 releases', and '1 contributor'. A section for the 'master' branch includes a 'New pull request' button and a 'Download ZIP' button. The file list shows two files: '.gitignore' (Initial commit, 9 months ago) and 'README.md' (Update README.md, 8 months ago). A tooltip 'Copy to clipboard' is visible over the repository URL.

TDiva / Examples

Unwatch 1 Star 0 Fork 0

<> Code Issues 0 Pull requests 0 Wiki Pulse Graphs Settings

Stuff I've wrote for reuse — Edit

6 commits 3 branches 0 releases 1 contributor

Branch: master New pull request

New file Find file HTTPS https://github.com/TDiva/ Examples Copy to clipboard Download ZIP

TDiva Update README.md Latest commit 07956ca on May 17, 2015

.gitignore	Initial commit	9 months ago
README.md	Update README.md	8 months ago

Tasks

- Use your GitHub account and do your tasks in Console

Q&A

- Questions?

Links to read!

- Docs
 - <https://git-scm.com/book/en>
 - <https://www.atlassian.com/git/tutorials/>
- “Survival” commands
 - <https://confluence.atlassian.com/bitbucketserver041/basic-git-commands-792301384.html>
- Git cookbook
 - <https://git.seveas.net/>
- Self-Practice
 - <https://githowto.com/>
- Tips/FAQ
 - <https://git.wiki.kernel.org/index.php/GitFaq>
 - <https://github.com/git-tips/tips>
- Best Practices
 - <https://sethrobertson.github.io/GitBestPractices/>
 - <http://moveelo.com/blog/git-best-practices-for-teams>
- Quick Introduction
 - <https://wildlyinaccurate.com/a-hackers-guide-to-git/>
 - <http://www.tutorialspoint.com/git/>