zehnplus.

GIT ZERO TO HERO

Bill Seremetis (bserem)
Vasilis Papoutsakis (Zekvyrin)
zehnplus.ch

WHAT IS GIT

A version control system (VCS) widely used in open source projects.

UUH... WHAT'S A VCS?

A tool that monitors changes in non-binary files*

Think of it as wikipedia's history tab, or revisions in Drupal content.

CENTRALIZED VS DECENTRALIZED

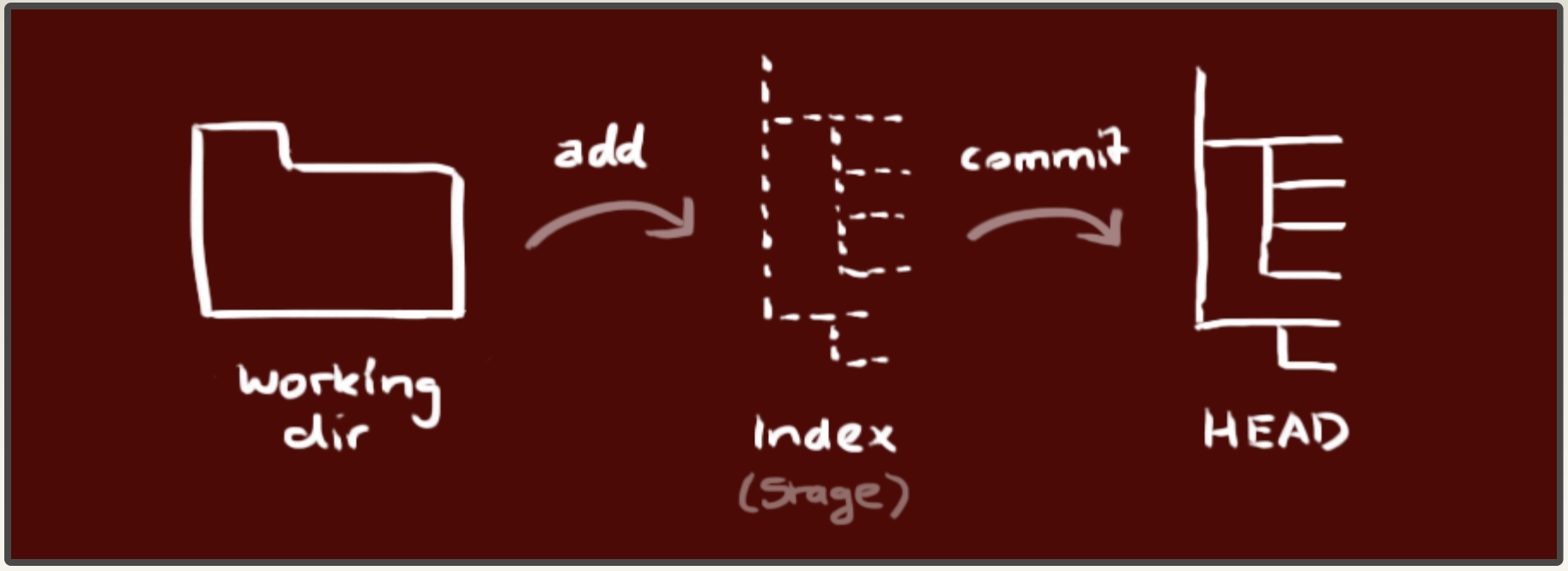
Git is a decentralized (or distributed) VCS:

- many developers can work on the same project without access to the same network
- you can sync changes from/to (pull/push) the repository whenever you want

MOUNTAIN CAMP

zehnplus.

THE 3 STATES OF INSANITY!



When working with Git, we instruct it to check specific files for changes. These files are called `tracked` files. Anything else is `untracked`.

UNTRACKED

Git doesn't care about `Untracked` files,
but doesn't ignore them either.

We can tell Git to ignore files.

TRACKED FILES

They can exist in the following states:

- `unstagged` the file has changes that haven't been stagged yet
- `stagged` but not commited: there are stagged changes that haven't been commited yet. *A file can have both staged and unstagged changes!*
- `commited` our changes are saved and git has saved the current state of our file in its history

YOU LOST ME IN THE FIRST SLIDE...

You must think in multiple dimensions in order to fully grasp what Git is. Maybe even multiple universes too!

- When a tracked file gets altered, Git sees that change immediately.
- We can save that change **temporarily** by staging it, or...
- We can save it permanently by commiting it

Note: *saving* above doesn't reflect the file being saved by your editor. It means saving the changes in Git history.

SO, GIT CAN EDIT FILES?

tl;dr: **NO!**

Git is not an editor, but it knows the current and past states of your file and can point you to these states. The files you see in your file system are affected by git.

zehnplus.

WHEN WILL WE GO TO THE `HERO` PART OF THIS PRESENTATION?

CREATING A NEW REPOSITORY

~~To create a new repository~~

WAIT, WHAT IS A REPOSITORY?

Simply put, a `repository` or `repo`, is a place where we store stuff.

CREATING A NEW REPOSITORY

```
git init
```

To tell git to monitor a directory for files you must initialize it first.

CLONING AN EXISTING REPOSITORY

```
git clone /path/to/repository
git clone username@host:/path/to/repository
```

- We can clone **existing** local repositories, or remote ones.
- Once cloning is done, we will have a local copy of the repo.

WORKFLOW

Our local repo can have files in all three states we already described:

- Unstagged (aka Working Directory)
- Stagged (aka Index)
- Commited (latest commit is also known as HEAD)

MOUNTAIN CAMP

zehnplus.

GIT STATUS

```
[git_zero_hero:master]$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   README.md

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   README.md
```

The most typed command in git: `git status`

It informs you about the current status (duh...) of your repo.

ADDING FILES

```
git add <filename>
git add *
```

`Add` puts the current file and/or its changes into `stage`. They **are not permanently saved yet**.

This is the **1st** step in a basic git workflow.

COMMITING FILES OR CHANGES

```
git commit -m "Your Commit Message"
```

Your changes are now commited to the **HEAD**, but only locally at the time being.

You must push them to the remote repo if you want to share them with the world.

PUSH & PULL

```
git push origin master
```

To send these changes to your remote repository you `push` them. To get changes from the remote repo locally you `pull` them.

- The remote repo you cloned from is called `origin` by default
- The first branch git creates for you after init is called `master`

So, the above command is: `push WHERE WHAT`

MULTIPLE REMOTE REPOSITORIES

```
git remote add <remote_name> <path_or_server>
```

Git allows you to have multiple branchs, and multiple remotes (repos).

To add a new remote: `git remote add <remote_name> <path_or_server>`.

COMMIT LOGS

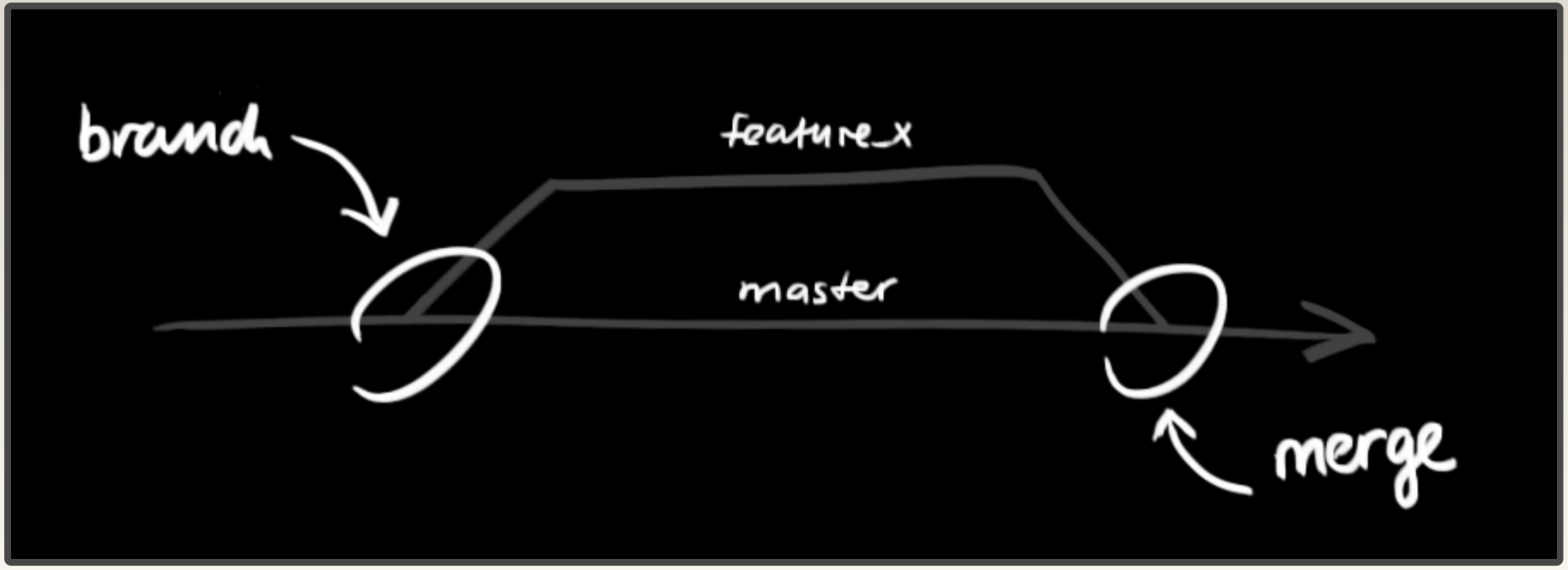
```
git log
git log -p
git log --oneline
```

```
commit 3d73fd21a40502c0dc410c7e3398d98e7d7106c9
Author: Bill Seremetis <bill@seremetis.net>
Date:   Fri Feb 17 13:17:12 2017 +0200

    Prepare slides version of the repository.

commit 9bbb3762f5f7b8e65f713424d891be3bf682d961
Merge: 33f447b cc286e1
Author: Bill Seremetis <bill@seremetis.net>
Date:   Fri Feb 17 13:10:01 2017 +0200

    Merge branch 'master' into slides
```

MOUNTAIN CAMP

BRANCHES

zehnplus.



We use branches to develop features in parallel from each other, or isolated.

Once we decide they are complete we can merge them back in our main branch (most often this will be `master`).

CREATING A BRANCH

```
git checkout -b feature_x
```

Do your work and commit it.

GET BACK TO ANOTHER BRANCH

```
git checkout master
```

Magic: Work committed under *feature_x* is not available here!

PUSHING A BRANCH

```
git push origin feature_x
```

Push the branch to a remore repository, so it is available to others.

MERGING

```
git merge feature_x
```

Gets the changes from feature_x branch to the branch you are currently in.

GETTING CHANGES FROM A REPOSITORY

```
git pull
```

```
git pull origin feature_x
```

and

```
git fetch
```

zehnplus.

THANKS!

Bill Seremetis (bserem) - zehnplus.ch

Vasilis Papoutsakis (Zekvyrin) - zehnplus.ch

With images and ideas from: http://rogerdudler.github.io/git-guide/