

# Oh my GIT!

Git ate my work and other stories

Vojtěch Vladyka

December 10, 2019

# Table of contents

## 1. What is git

...and why you should care

## 2. Setup git

Clients, differences, common issues

## 3. Git essentials

Git controls crash course

## 4. Scenarios

We are going to break things there. A lot.

## 5. Q & A

Umm, why was I here again?

# What is git?

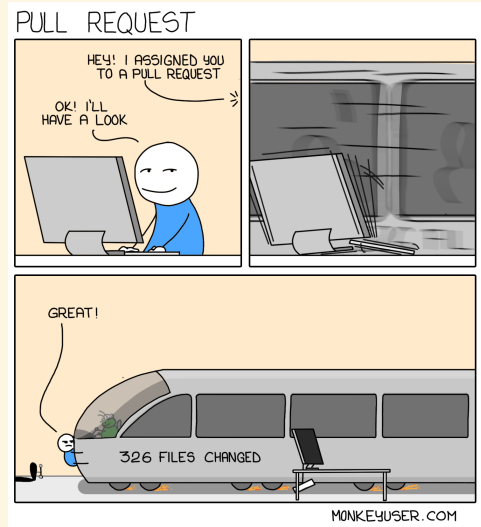
- ▶ Git is distributed by design
- ▶ Works with whole sourcetree (unlike SVN which works with individual files and their revisions) and always capture state of working tree
- ▶ Strong support for non-linear development → rapid branching & merging
- ▶ Cryptographic authentication of history - every commit has SHA-1 hash of everything leading to that point
- ▶ Version control system made by Linus Torvalds (mostly) at 2005 for Linux Kernel development
- ▶ Supports rapid branching & merging
- ▶ it is pronounced GIT with G like GIF<sup>1</sup>

---

<sup>1</sup>Google. *Tech Talk: Linus Torvalds on git*. URL:  
<https://www.youtube.com/watch?v=4XpnKHJAok8>.

# Git philosophy

- ▶ Create branch for each feature
- ▶ Commit often
- ▶ Merge back to master as soon as possible
- ▶ Create small isolated as-much-as-possible pull requests



# Setup git

## Git clients

1. Git (all platforms)
2. Git Extensions (Windows, Mac)
3. Tortoise Git (Windows)
4. Sourcetree (Windows)
5. Fork (Windows, Mac)
6. Gitkraken (all platforms)
7. ...many others

# Setup git

## Git Extensions (Git client)

The screenshot displays the Git Extensions application window. The title bar reads 'legallyrelevantcore (GEMPFW-56\_NVM\_error) - Git Extensions'. The menu bar includes 'Start', 'Repository', 'Navigate', 'View', 'Commands', 'GitHub', 'Plugins', 'Tools', and 'Help'. The toolbar shows icons for repository operations, navigation, and commit management, with a status bar indicating 'Commit (10)'.

The left sidebar shows the repository structure:

- Search: [ ]
- Branches
  - GEMPFW-56\_NVM\_error
  - GEMPFW-59\_Wdog\_error
  - master (161)
  - GEMPFW-609\_Third\_batch
  - GEMPFW-609\_Second\_batch
  - GEMPFW-609\_Fixing\_log\_code\_issues
  - GEMFW-609\_Added\_log\_code\_to\_analysis
  - SystemView
- Remotes
  - origin
- Submodules
  - legallyrelevantcore [GEMPFW-56\_NVM\_error]
    - src
      - libs
        - cpptest\_code [no branch]
        - cryptolib\_code [no branch]
        - eosal\_code [no branch]
        - freertos\_code [no branch]
        - hal\_code [no branch]
        - runservice\_code [no branch]
        - storage\_code [no branch]
        - shareddefinitions [master]

The main pane shows the commit history and a diff view. The commit list includes:

- GEMPFW-56\_NVM\_error (selected)
- origin/GEMPFW-56\_NVM\_error
- GEMPFW-56 fixed unit test
- origin/GEMPFW-666\_NumberOfTheBeast
- GEMPFW-666
- GEMPFW-666 - Added RollOver feature to all Total Register Objects - Affected objects: OBIS\_A...
- GEMPFW-56 returned shareddefinitions to master branch
- GEMPFW-56 Added missing include
- origin/master
- Merge pull request #209 in PVRMIDKS/legallyrelevantcore from GEMPFW-59\_W...
- GEMPFW-56 Added NVM alarm to setup store/restore
- GEMPFW-56 Added NVM alarm to runservice
- origin/GEMPFW-676\_LCDstatusCheck
- GEMPFW-676 - finished implementation. Waiting for t...
- Merge branch 'GEMPFW-59\_Wdog\_error' into GEMPFW-56\_NVM\_error

The diff view shows changes in the file 'src/UnitTest/inc/COSEMDData.h'.

```
diff --git a/src/UnitTest/inc/COSEMDData.h b/src/UnitTest/inc/COSEMDData.h
index 5ef17077..b3502938 100644
--- a/src/UnitTest/inc/COSEMDData.h
+++ b/src/UnitTest/inc/COSEMDData.h
@@ -5,7 +5,21 @@
5 5
6 6 namespace COSEMDATA
7 7 {
8 -
8 +class AlarmRegister {
9 + public:
10 +
11 + static void SetAlarmWatchdog() {
12 + }
13 +
14 + static void ResetAlarmWatchdog() {
15 + }
```

Download it here → <https://gitextensions.github.io>

# Git essentials

## Short and incomplete command overview

```
git init  
clone
```

```
fetch  
merge  
push  
pull
```

```
branch  
checkout  
merge  
branch -d
```

```
git log  
log --follow [file]  
reflog
```

```
add  
commit
```

```
reset  
reset --hard  
stash
```

```
rebase  
cherry-pick
```

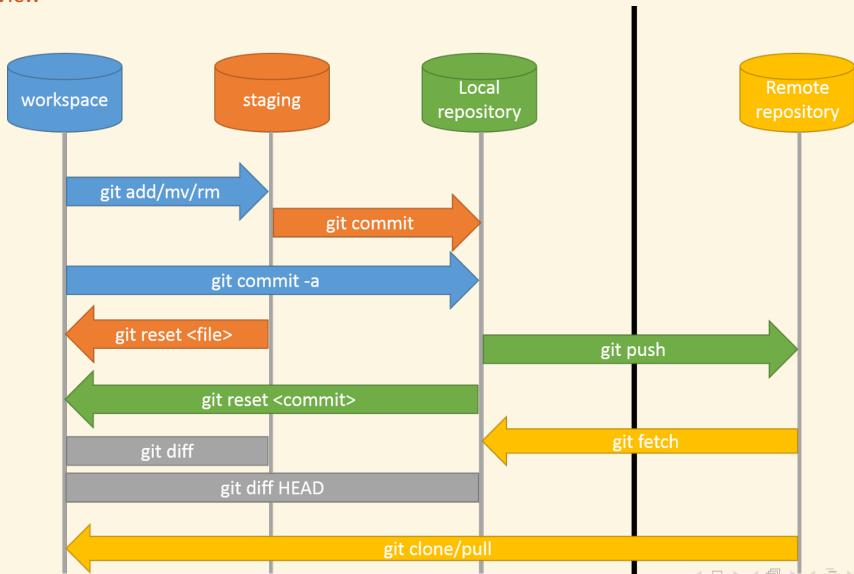
# Git essentials





# Git essentials

## Workflow overview



# Git essentials

## Init repo

```
git init
```

→ initialize new repository in current directory with all its contents

```
git clone https://somerepo.com/repo.git
```

→ downloads repository with all current branches & commits

# Git essentials

## Fetching changes and returning them back

```
git pull
```

→ Download commits from server and apply them to your working tree

```
git push
```

→ Upload your commits to server

```
git fetch --all
```

→ Download commits from server (from all branches eventually)

```
git commit [files] -m "Commit message"
```

→ Create a commit with commit message and specified files

→ You can use -a to commit all changes instead list of files

```
git add [file]
```

→ Move file to staging area - add it to commit list

# Git essentials



	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

# Git essentials

## Undoing changes

```
git reset
```

→ Soft reset - remove files from staging area and move them back to working area. Nothing is actually reverted, this resets it only for git.

```
git reset --hard
```

→ Hard reset - this do all what soft reset but also REMOVES your uncommitted changes.

→ You can do this only locally. Once you push your changes, you cannot get it back. (not exactly true but you will break a lot of thing for all colleagues). Only way how to properly revert things it using git revert.

IT HELL.

OK, LET'S  
SEE WHAT  
YOU'RE IN  
FOR...

TO QA AREA

ENJOY  
YOUR STAY

PERL



NO COMMENTS

IGNORED  
STYLE  
GUIDE

BROKE  
BUILD

 $O(n!)$ 

USED =  
(INSTEAD OF  
=)

SUDO  
RM -RF /\*

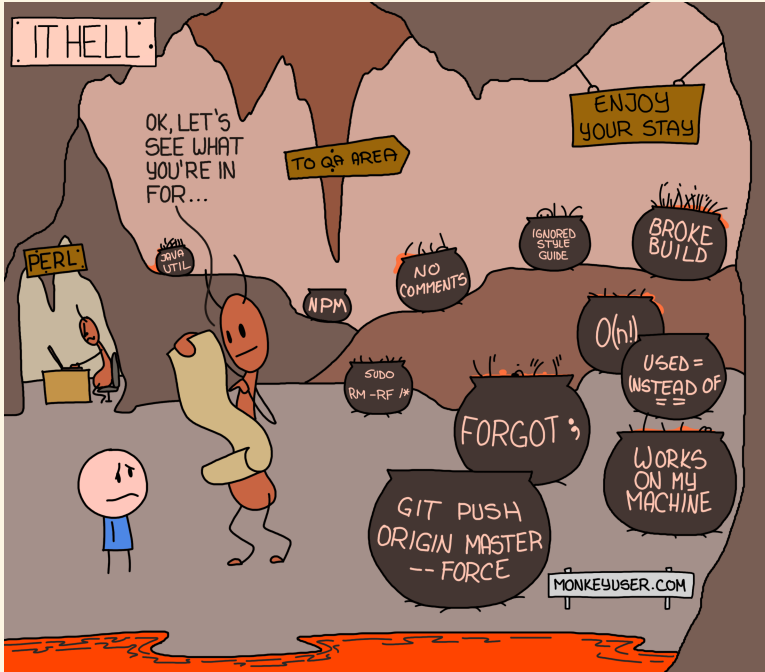
FORGOT :



GIT PUSH  
ORIGIN MASTER  
--FORCE

WORKS  
ON MY  
MACHINE

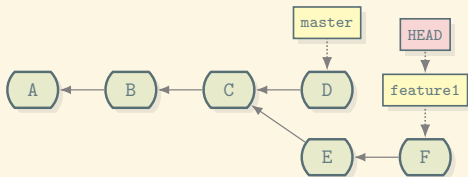
MONKEYUSER.COM



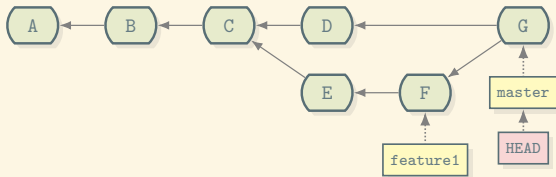
# Git essentials

## Branching

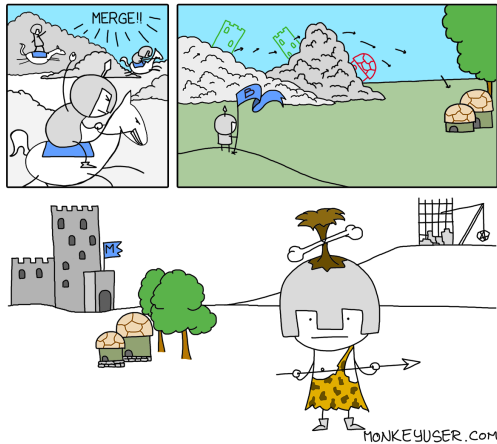
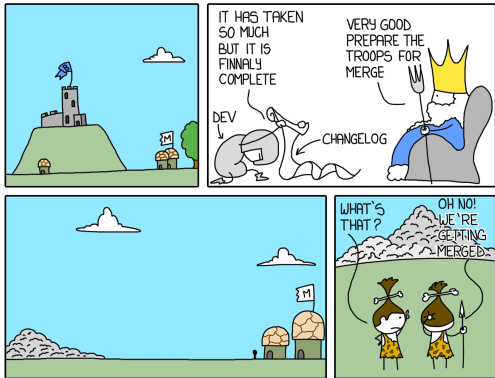
```
git checkout -b feature1  
(git branch feature1 ; git checkout feature1)
```



```
git checkout master  
git merge feature1
```



## MERGING BRANCHES





# Git essentials

## Logging and searching lost stuff

```
git log --graph
```

→ Shows log for whole repository tree

```
git log --follow [file]
```

→ Shows log for selected file even across renames

```
git reflog
```

→ Shows log of ALL changes in repository. Stashes, detached branches, all...

# Git essentials

Stashing - let me just put this here for later (and never use it again)

```
git stash
```

```
git stash push
```

→ Stashes current changes to new stash stack

```
git stash pop
```

→ Pops newest stash and apply it to current working copy

```
git stash list
```

→ Show all stashes in list

```
git stash apply stash@{2}
```

→ Get stash #2 and apply it to current working copy

```
git checkout stash@{2} -- somefile
```

→ Get stash #2 and apply it to specified files in current working copy

# Git essentials

## Submodules

```
git submodule add <url to submodule repository>
```

→ This adds new submodule to current repository and also clones it.

```
git submodule init
```

→ This clones submodules when you just cloned new repository with submodules.

```
git submodule update --init --recursive
```

→ And this downloads submodules to version specified in repository (and eventually clones new submodules)

```
git submodule foreach 'do something'
```

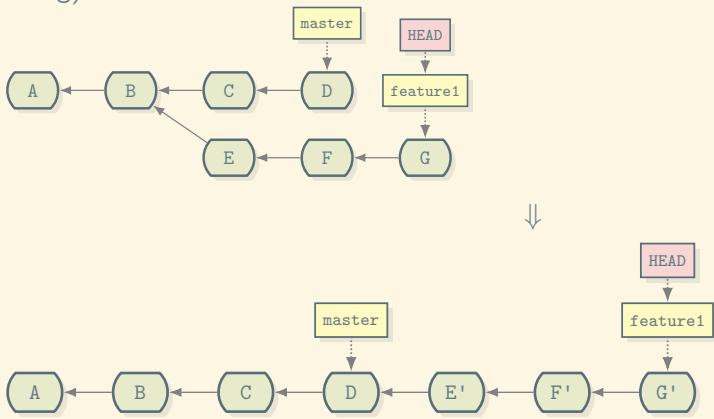
→ This runs 'do something' on every submodule.

# Git essentials

## Rebase

```
git rebase target_branch source_branch
```

→ Take source branch and reattach it on end of target branch. This creates series of new commits and remove old ones from tree (although they will still be accessible by reflog)

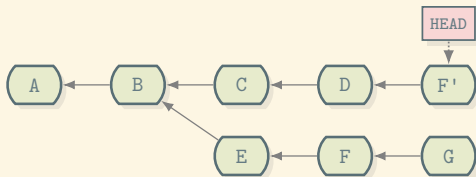
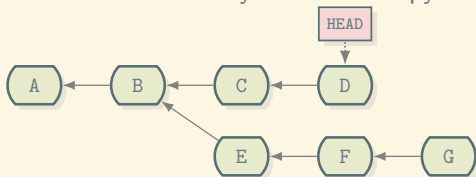


# Git essentials

## Cherry picking

```
git cherry-pick <commit hash>
```

→ Select commit by hash and copy it on end of current branch as new commit.



# Scenarios

It's time to break some stuff.

# Scenarios

## Fresh start - cloning

```
git clone https://bitbucket.honeywell.com/users/e878714/  
repos/playground/commits
```

→ This downloads whole repository into current working directory. (that URL should not be splitted but slide is too short...)

# Scenarios

## Branching

→ We do not want to fight all together so we create own branches

```
git branch your-branch-name  
git checkout your-branch-name
```

→ Now we should all be at 'your-branch-name' branch.



# Scenarios

## Committing

→ Do some changes (anything) in code you just cloned

```
git commit -am "Your commit message"
```

→ This commits all your changes with your commit message.

# Scenarios

## Pushing

→ Let's put our changes on server

```
git push
```

→ This should actually fail because we are pushing to non-existent branch on server

```
git push --set-upstream origin your-branch-name
```

→ And this will work. It creates new branch with our name on remote server named 'origin'

# Scenarios

## Adding

→ Let's create new file and add it to repository

```
git add our-new-file.txt
```

→ This adds our-new-file.txt to staging area. We need to commit it next.

```
git commit -am "Your commit message"
```

→ If we are happy with it, we can push it.

```
git push
```

# Scenarios

## Merging and pulling

→ Now we want to merge our stuff into master. Normally we will create pull request but today we will merge directly to master. We start with checking out master branch.

```
git checkout master
```

→ Now we need to get latest changes from it.

```
git pull
```

→ And now we can merge it.

```
git merge vojtuvs-test
```

→ And push result.

```
git push
```

# Scenarios

## Log, Checkout

- We want to work on our branch again (or somebody else's branch).
- We get latest changes from all branches and delete references to non-existent ones.

```
git fetch --all --prune
```

- Now we can take a look on current repository log

```
git log --graph
```

- And eventually checkout something

```
git checkout our-branch-name
```

# Scenarios

## Stash

→ Let's do some changes...

→ Ok, we don't want them actually. We need to get rid of them. But it's really nice piece of work so we will keep them for later.

```
git stash
```

→ Ok, it's gone. You can verify it with status.

```
git status
```

→ Now fast forward a little bit and assume we want them back. We can be on different branch, we can be in future, it doesn't matter.

```
git stash list
```

```
git stash apply stash@{0}
```

→ Check who's back!

# Scenarios

## Reset

→ Oh my god, that change was rubbish... We need to delete it.

```
git reset --hard
```

→ Uff, it's gone.

# Scenarios

Git online playground

If you want to play with branching, rebasing, merging, squashing etc., this is great tool for playing with it.

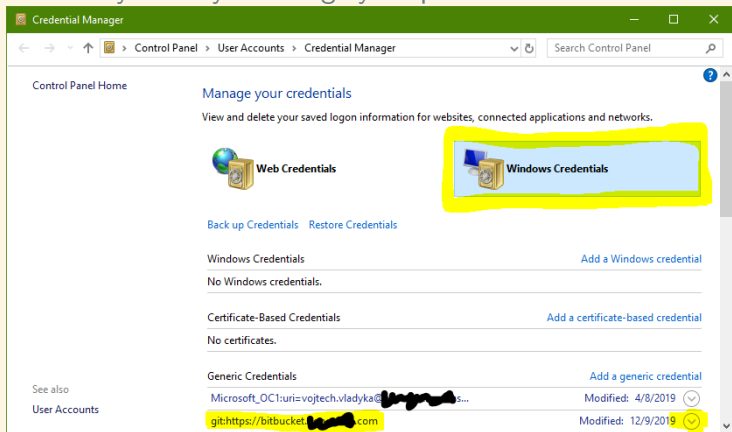
<https://learngitbranching.js.org>







# Troubleshooting

## Windows Credential Manager

Git.exe uses Credential manager for storing login information. It's nice feature with one undesired issue - password change doesn't affect stored credentials so you need to remove record manually when you change your password.



# Resources

-  Google. *Tech Talk: Linus Torvalds on git*. URL: <https://www.youtube.com/watch?v=4XpnKHJAok8>.
-  Mark Erikson. *Git Under the Hood*. URL: <https://blog.isquaredsoftware.com/presentations/2019-03-git-internals-rewrite/#/0>.
-  GitHub. *Git Cheat Sheet*. URL: <https://github.github.com/training-kit/downloads/github-git-cheat-sheet.pdf>.
-  Git community. *Git –distributed-even-if-your-workflow-isnt*. URL: <https://git-scm.com>.

# Q & A

Thank you for your attention.