GIT WORKSHOP

DragonSec SI

Ljubljana FRI, December 2024

KDO SMO

DragonSec SI

@DragonSec_SI

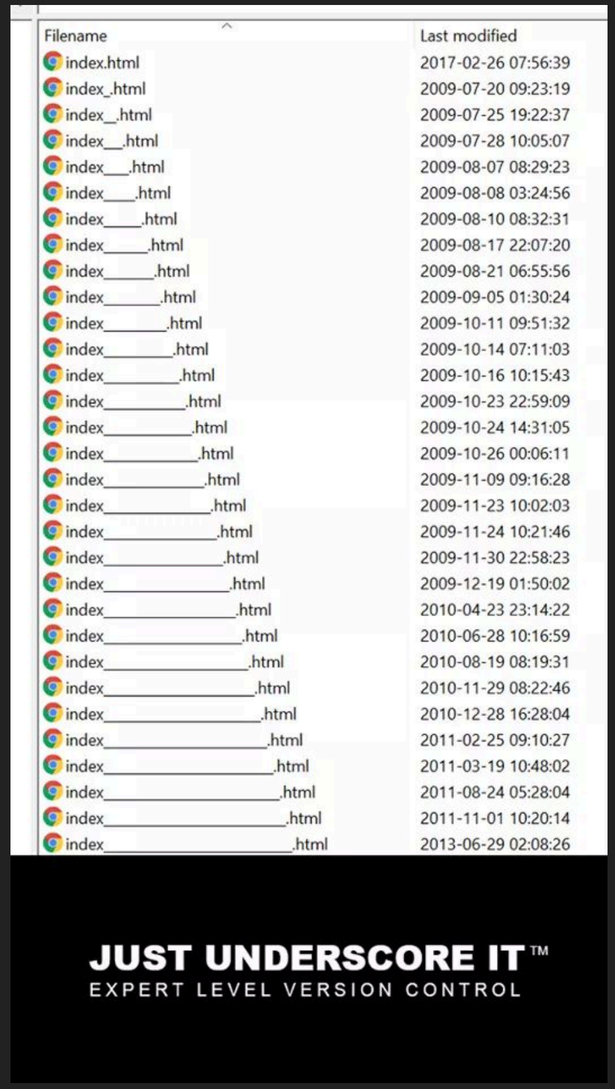
info@dragonsec.si

GIT

Kaj sploh je git?

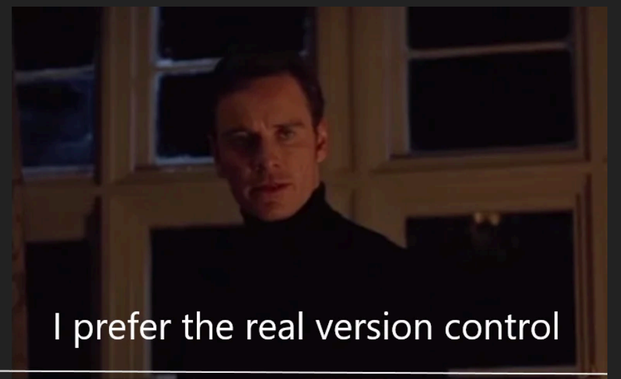
GIT VS. GITHUB

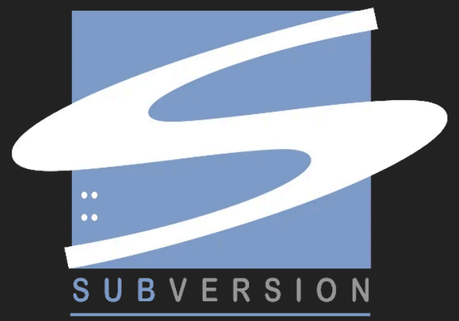
ZAKAJ BI UPORABLJALI GIT?



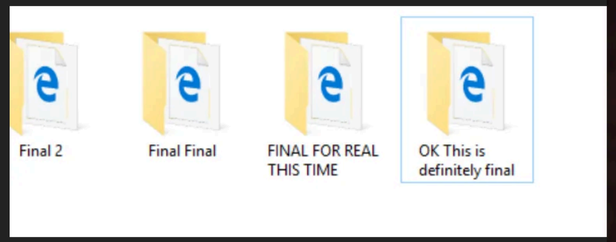

git
Ctrl + Z
Ctrl + Z
Ctrl + Z
Ctrl + Z
Ctrl + Z
Ctrl + Z
Ctrl + Z


git


I prefer the real version control


SUBVERSION


I said the *real* version control


Final 2
Final Final
FINAL FOR REAL THIS TIME
OK This is definitely final


Perfection

KAJ JE DIFF?

```
λ cat -n a.txt
 1  AAAAA
 2  BBBBB
 3  CCCCC
 4  DDDDD
 5  EEEEE
 6  FFFFF
 7  GGGGG
 8  HHHHH
```

```
λ cat -n b.txt
 1  BBBBB
 2  AAAAA
 3  CCCCC
 4  DDDDD
 5  IIIII
 6  EEEEE
 7  FFFFF
 8  HHHHH
 9  LLLLL
```

```
λ diff a.txt b.tx
1d0
< AAAAA
2a2
> AAAAA
4a5
> IIIII
7d7
< GGGGG
8a9
> LLLLL
```

KAKO ZAČETI Z GITOM?

```
λ git init
```

```
λ ls -lah
drwxr-xr-x spagnologasper users 4.0 KB ... 2024 .
drwxr-xr-x spagnologasper users 4.0 KB ... 2024 ..
drwxr-xr-x spagnologasper users 4.0 KB ... 2024 .git
```

KAJ SE USTVARI OB GIT INIT?

```
λ ls -lah .git
drwxr-xr-x spagnologasper users 4.0 KB ...  .
drwxr-xr-x spagnologasper users 4.0 KB ...  ..
drwxr-xr-x spagnologasper users 4.0 KB ...  branches
.rw-r--r-- spagnologasper users  92 B  ...  config
.rw-r--r-- spagnologasper users  73 B  ...  description
.rw-r--r-- spagnologasper users  23 B  ...  HEAD
drwxr-xr-x spagnologasper users 4.0 KB ...  hooks
drwxr-xr-x spagnologasper users 4.0 KB ...  info
drwxr-xr-x spagnologasper users 4.0 KB ...  objects
drwxr-xr-x spagnologasper users 4.0 KB ...  refs
```

DODAJANJE NOVE DATOTEKE

```
init on master
λ echo "#TO JE PRVA DATOTEKA" >> prva_datoteka.txt

init on master [?]
λ ls -lah
drwxr-xr-x spagnologasper users 4.0 KB ... .
drwxr-xr-x spagnologasper users 4.0 KB ... ..
drwxr-xr-x spagnologasper users 4.0 KB ... .git
.rw-r--r-- spagnologasper users  21 B  ... prva_datoteka.txt
```

DODAJANJE NOVE DATOTEKE

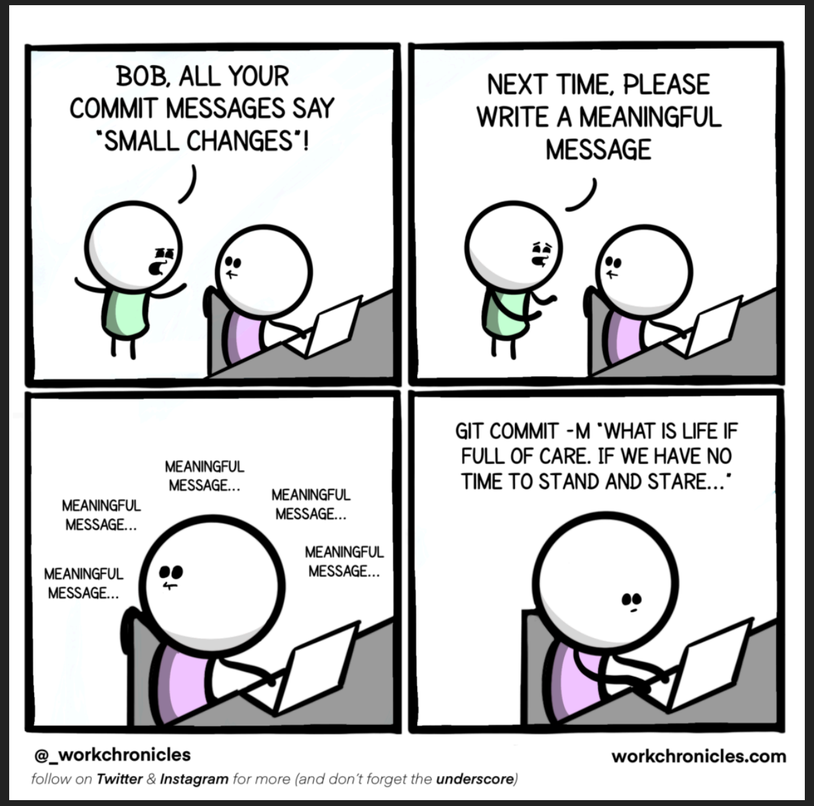
```
λ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        prva_datoteka.txt

nothing added to commit but untracked files present (use "git ad
```

DODAJANJE NOVE DATOTEKE

```
λ git add prva_datoteka.txt
```

```
λ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   prva_datoteka.txt
```

UVELJAVLJANJE SPREMEMB




```
λ git commit -m "Zelo deskriptivno sporocilo!"
```

KAKO DODAMO ODDALJENI REPOZITORIJ?

Želja po povezavi z oddaljenim repozitorijem:

```
λ git remote add origin \
    https://gitea.spanskiduh.dev/spanskiduh/test-repo.git
```

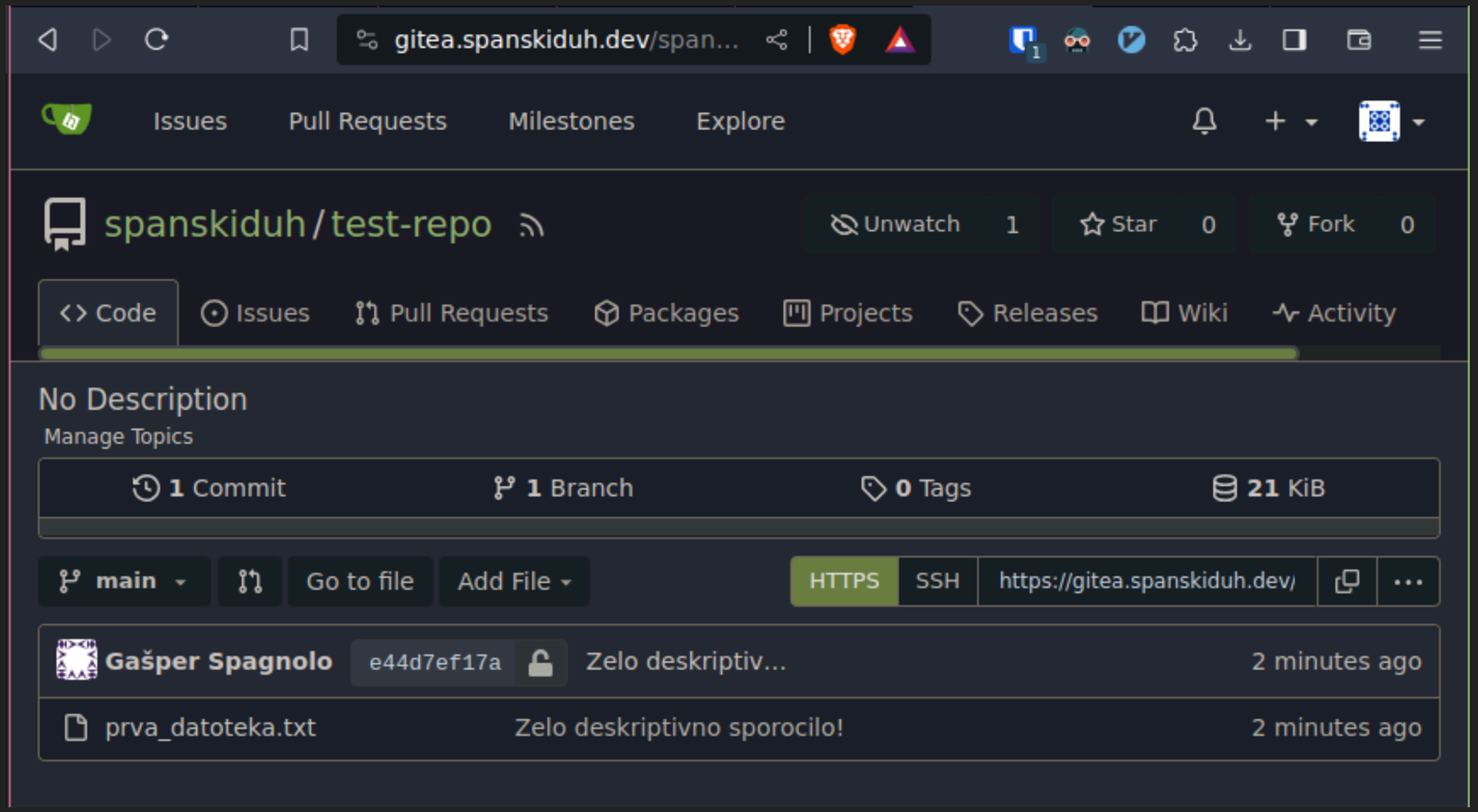
V datoteko `.git/config` se doda vrstica:

```
[remote "origin"]
        url = https://gitea.spanskiduh.dev/spanskiduh/test-repo.
        fetch = +refs/heads/*:refs/remotes/origin/*
```

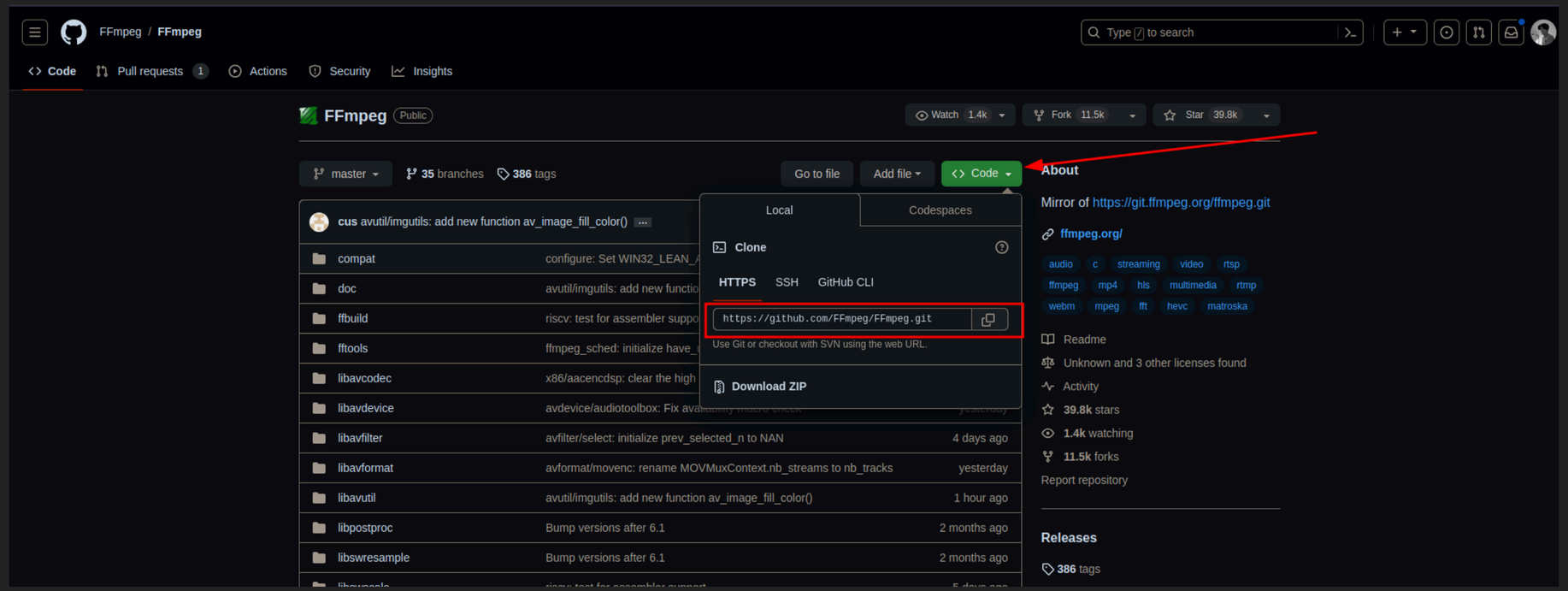
SINHRONIZACIJA LOKALNEGA REPOZITORIJA Z ODDALJENIM


```
λ git checkout -b main; git push -u origin main
```

SINHRONIZACIJA LOKALNEGA REPOZITORIJA Z ODDALJENIM



KLONIRANJE ODDALJENEGA REPOZITORIJA




```
λ git clone https://github.com/FFmpeg/FFmpeg.git
```

GIT PULL VS. GIT FETCH

Recimo, da smo klonirali ta repozitorij kakšen mesec nazaj in želimo pridobiti najnovejše spremembe.

```
λ git fetch
```

GIT PULL VS. GIT FETCH

```
λ git pull
```

Basically `git pull = git fetch + git merge`

PREGLED ZGODOVINE UVELJAVITEV

PREGLED ZGODOVINE UVELJAVITEV

```
λ git log
```

```
commit e44d7ef17a34fa9fa2d92ace8dd1296c06353d90 (HEAD -> main, c
Author: Gašper Spagnolo <gasper.spagnolo@outlook.com>
Date:   Tue Dec 12 22:02:03 2024 +0100

    Zelo deskriptivno sporocilo!
```

```
λ git log --oneline
```

```
e44d7ef (HEAD -> main, origin/main, master) Zelo deskriptivno sp
```

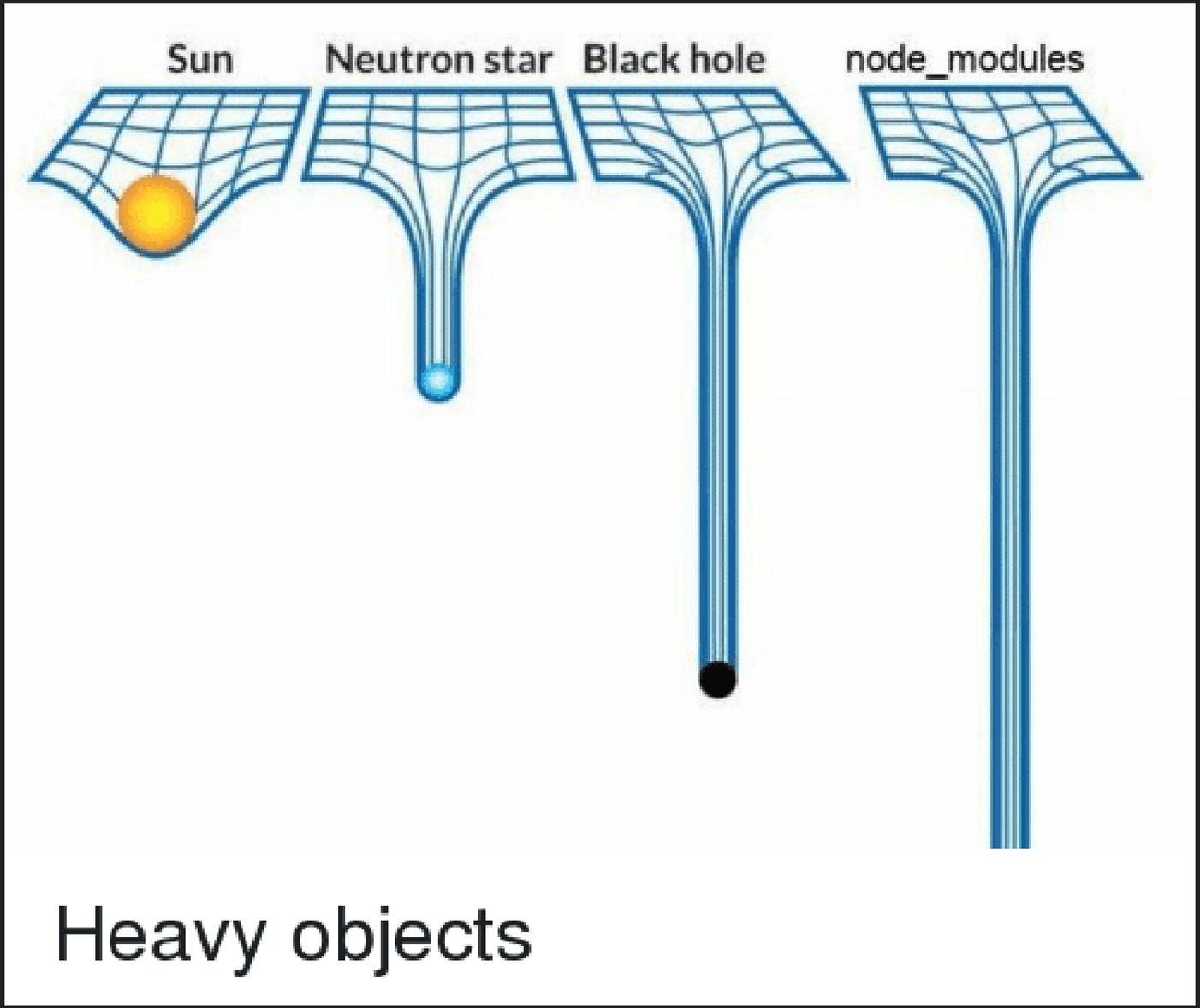
.GITIGNORE

Pomaga nam ignorirati datoteke, ki jih ne želimo dodati v repozitorij.

```
λ echo "mojabaza.sqlite" >> .gitignore
λ ls
mojabaza.sqlite prva_datoteka.txt
```

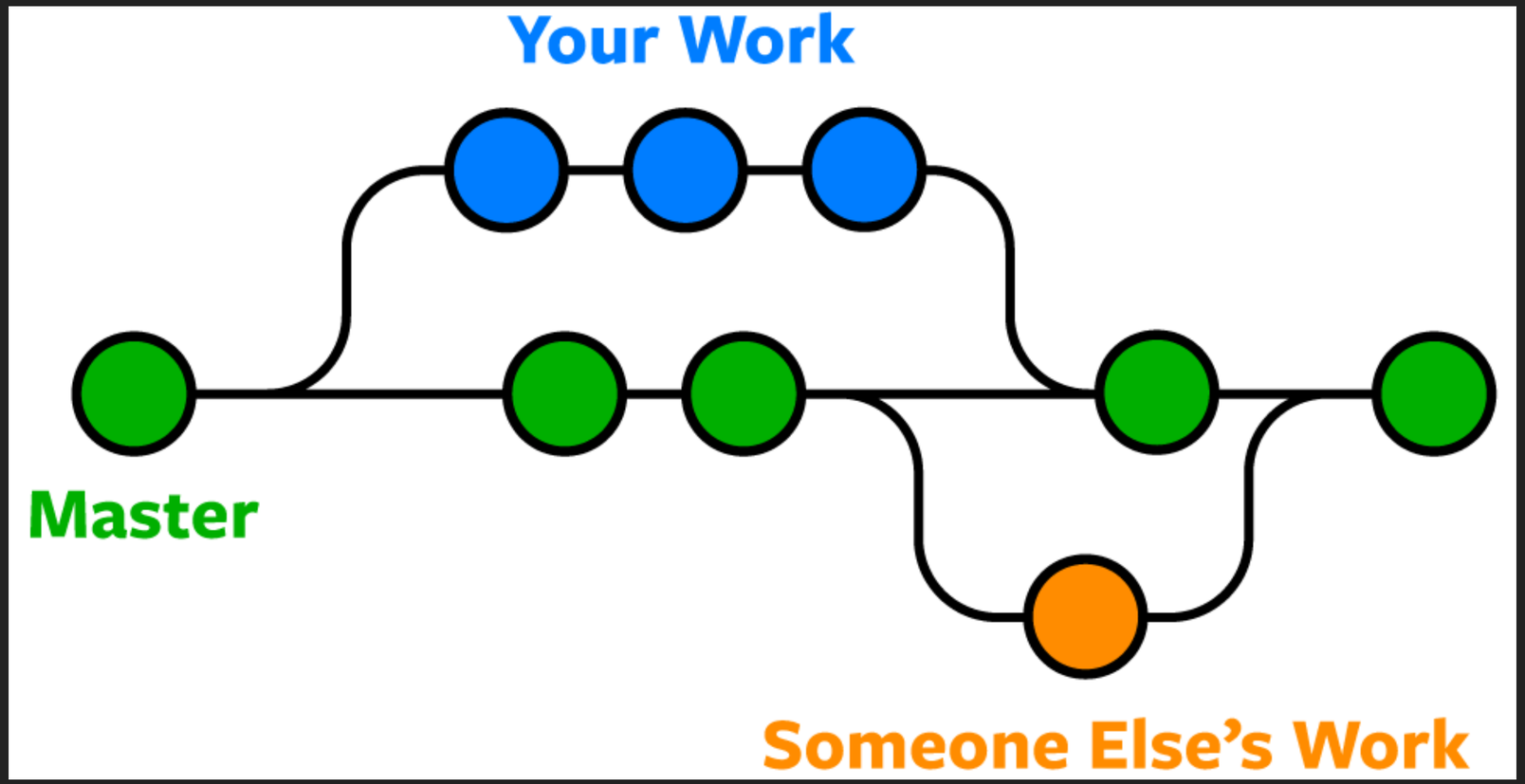
```
λ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore

nothing added to commit but untracked files present (use "git ad
```

.GITIGNORE

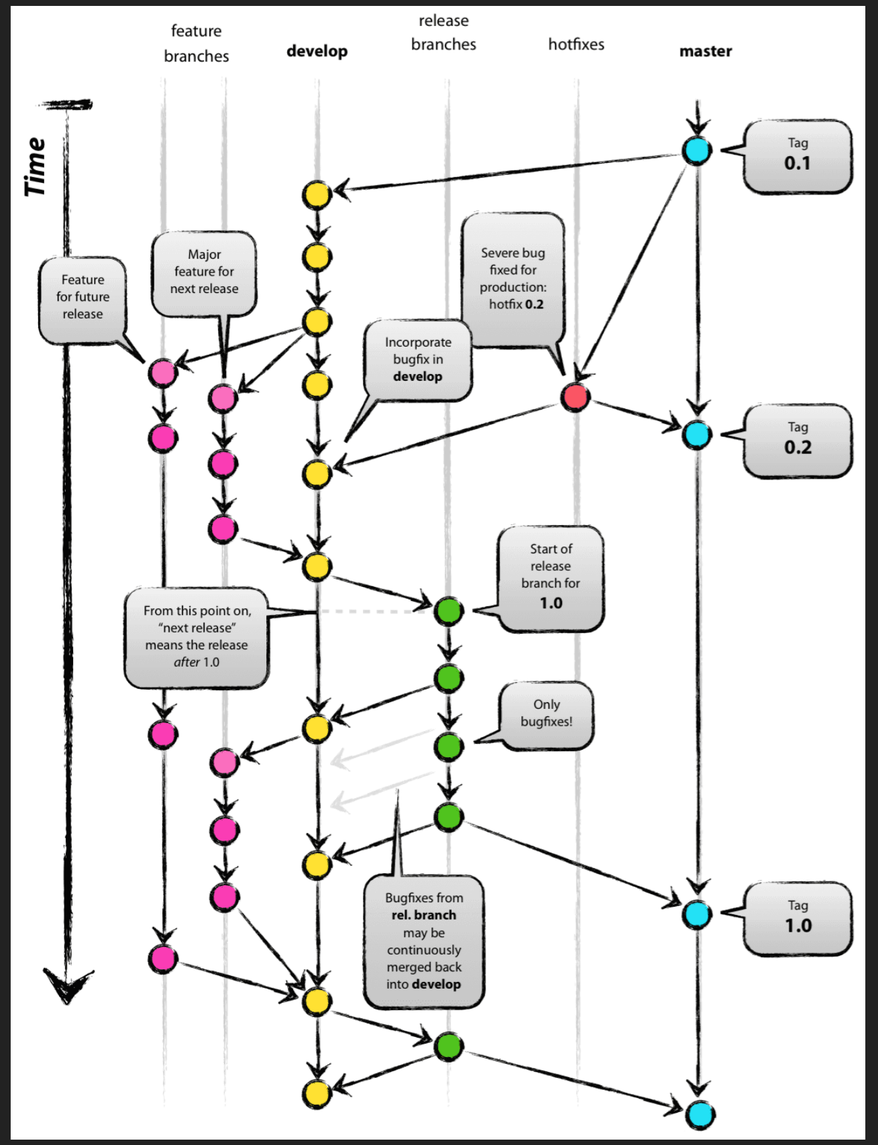


BRANCHING

Zakaj?



REALITY


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Major feature for next release
Feature for future release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

USTVARJANJE NOVE VEJE

```
λ git checkout -b feats1337
Switched to a new branch 'feats1337'
# -b pomeni, da ustvarimo novo vejo
```

Spremenino datoteko `prva_datoteka.txt`:

```
λ printf "\n1337 feature" >> prva_datoteka.txt
```

Dodamo spremembe v repozitorij:

```
λ git add prva_datoteka.txt
```

USTVARJANJE NOVE VEJE

Uveljavimo spremembe:

```
λ git commit -m "Dodaj vrstico v prvo datoteko"
```

```
λ git push -u origin feats1337

Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a new pull request for 'feats1337':
remote:   https://gitea.spanskiduh.dev/spanskiduh/test-repo/comp
remote:
remote: . Processing 1 references
remote: Processed 1 references in total
To https://gitea.spanskiduh.dev/spanskiduh/test-repo.git
 * [new branch]      feats1337 -> feats1337
branch 'feats1337' set up to track 'origin/feats1337'.
```

PREGLED VEJ

Lokalno:

```
λ git branch

* feats1337
  main
  master
```

Remote:

```
λ git branch  -r

  origin/feats1337
  origin/main
```

PREGLED VEJ

```
λ git log --graph --oneline --all

* commit 634d79a74c2f7347c0afb8fb8305b523ee16d05a (HEAD -> feats
| Author: Gašper Spagnolo <gasper.spagnolo@outlook.com>
| Date:   Tue Dec 12 23:06:49 2024 +0100
|
|     Dodaj vrstico v prvo datoteko
|
* commit e44d7ef17a34fa9fa2d92ace8dd1296c06353d90 (origin/main,
  Author: Gašper Spagnolo <gasper.spagnolo@outlook.com>
  Date:   Tue Dec 12 22:02:03 2024 +0100

      Zelo deskriptivno sporocilo!
```

GLOBJE

```
* commit b62623b5451187faf463bf5f0e0a93ea0c1fbf45 (HEAD -> feats
| Author: Gašper Spagnolo <gasper.spagnolo@outlook.com>
| Date:   Tue Dec 12 23:11:14 2024 +0100
|
|     Se ena vrstica
|
* commit e60b7db578566b1f4110e04d95580118a6a3160d
| Author: Gašper Spagnolo <gasper.spagnolo@outlook.com>
| Date:   Tue Dec 12 23:10:56 2024 +0100
|
|     Dodaj novo vrstico v datoteko
|
* commit 8a79fbcaefdd8f445f6ff1d4a5e693963f4772dc
| Author: Gašper Spagnolo <gasper.spagnolo@outlook.com>
| Date:   Tue Dec 12 23:10:23 2024 +0100
|
```

TREUNUTNO STANJE

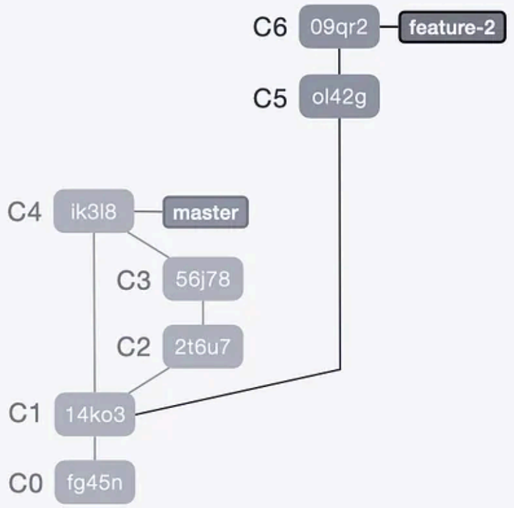
```
main: e44d7 --
            |
feats1337:  634d79 --  8a79f -- e60b7 -- b62623
```

ZDRUŽEVANJE VEJ

Start case

There are two main ways to integrate changes between branches, **merge** or **rebase**.

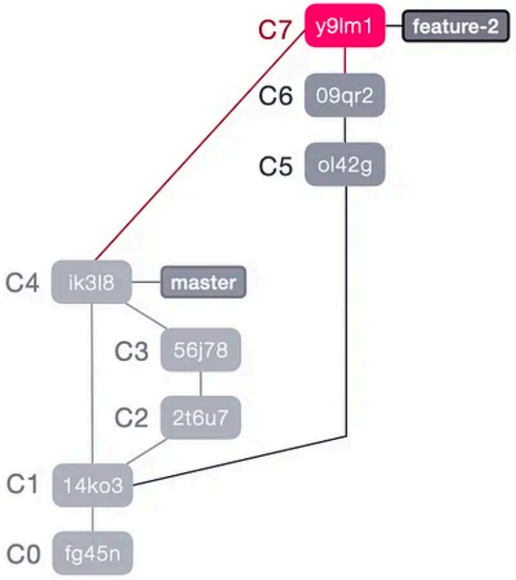
Here, *feature-2* is to be updated with changes from *master*.



Post merge

Merge preserves history as it happened, creating only one new merge commit.

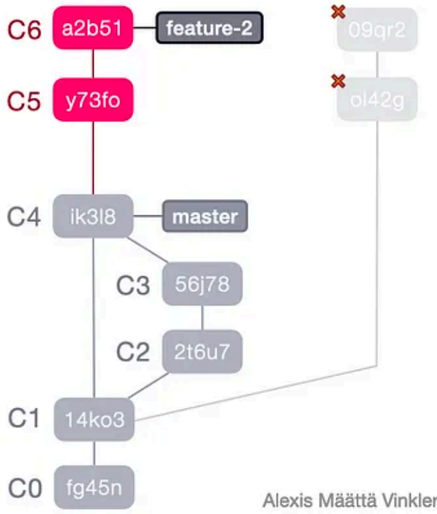
Here, commit **C7** intertwines the two branches – creating a non-linear diamond shaped history.



Post rebase

Rebase rewrites history, reapplying commits on top of another base branch.

Here, commits **C5** and **C6** have been reapplied on top of **C4** – creating a linear history.



Alexis Määttä Vinkler

ZDRUŽEVANJE VEJ

Merge:

```
main: e44d7 -- M
        |
feats1337:        634d79 --  8a79f -- e60b7 -- b62623
```

Rebase:

```
e44d7 -- [634d79 --  8a79f' -- e60b7' -- b62623']
```

REBASE

```
λ git checkout feats1337
λ git rebase main
λ git checkout main
λ git merge feats1337
```

REBASE

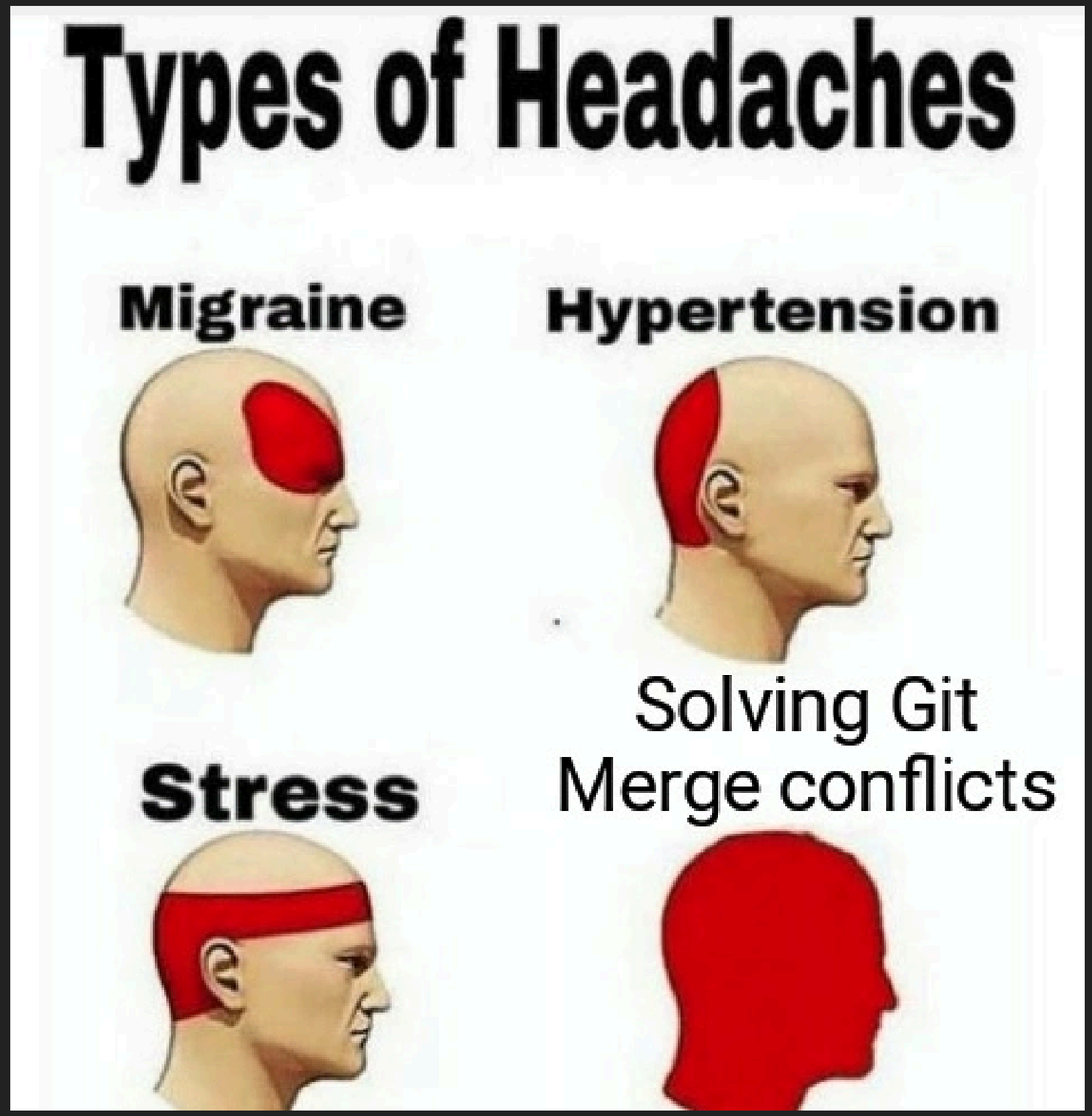
```
* commit b62623b5451187faf463bf5f0e0a93ea0c1fbf45 (HEAD -> main,
| Author: Gašper Spagnolo <gasper.spagnolo@outlook.com>
| Date:   Tue Dec 12 23:11:14 2024 +0100
|
|     Se ena vrstica
|
* commit e60b7db578566b1f4110e04d95580118a6a3160d
| Author: Gašper Spagnolo <gasper.spagnolo@outlook.com>
| Date:   Tue Dec 12 23:10:56 2024 +0100
|
|     Dodaj novo vrstico v datoteko
|
* commit 8a79fbcaefdd8f445f6ff1d4a5e693963f4772dc
| Author: Gašper Spagnolo <gasper.spagnolo@outlook.com>
| Date:   Tue Dec 12 23:10:23 2024 +0100
|
```

MERGE

```
λ git checkout -b feats222
# Create some commits on feats222
λ git checkout main
λ git merge feats222
```

MERGE CONFLICTS

MERGE CONFLICTS



PRIMER MERGE CONFLICTA

Najprej ustvarimo `example.txt` datoteko z vsebino:

```
λ git checkout main
λ echo "Initial content" > example.txt
λ git add example.txt
λ git commit -m "Add example.txt with initial content"
```

PRIMER MERGE CONFLICTA

Ustvarimo novo vejo in dodamo novo vrstico v datoteko:

```
λ git checkout -b feature-branch
λ echo "Content added in feature-branch" > example.txt
λ git add example.txt
λ git commit -m "Modify example.txt in feature-branch"
```

PRIMER MERGE CONFLICTA

Se vrnemo nazaj na `main` in dodamo novo vrstico v datoteko:

```
λ git checkout main
λ echo "Conflicting content in main branch" > example.txt
λ git add example.txt
λ git commit -m "Quickfix, push to production"
```

PRIMER MERGE CONFLICTA

Združimo obe veji:

```
λ git merge feature-branch
```

O ne!

```
λ git merge feature-branch
Auto-merging example.txt
CONFLICT (content): Merge conflict in example.txt
Automatic merge failed; fix conflicts and then commit the result
```

PRIMER MERGE CONFLICTA

V tem primeru datoteko uredimo tako da obdržimo uveljavitev, ki jo zelimo obdržati.

```
λ cat example.txt
<<<<<<< HEAD
Conflicting content in main branch
=======
Content added in feature-branch
>>>>>>> feature-branch
```

PRIMER MERGE CONFLICTA

Po popravku datoteke jo dodamo v repozitorij:

```
λ git add example.txt
```

Ter uveljavimo spremembe:

```
λ git commit -m "Fix merge conflict"
```

PULL REQUEST

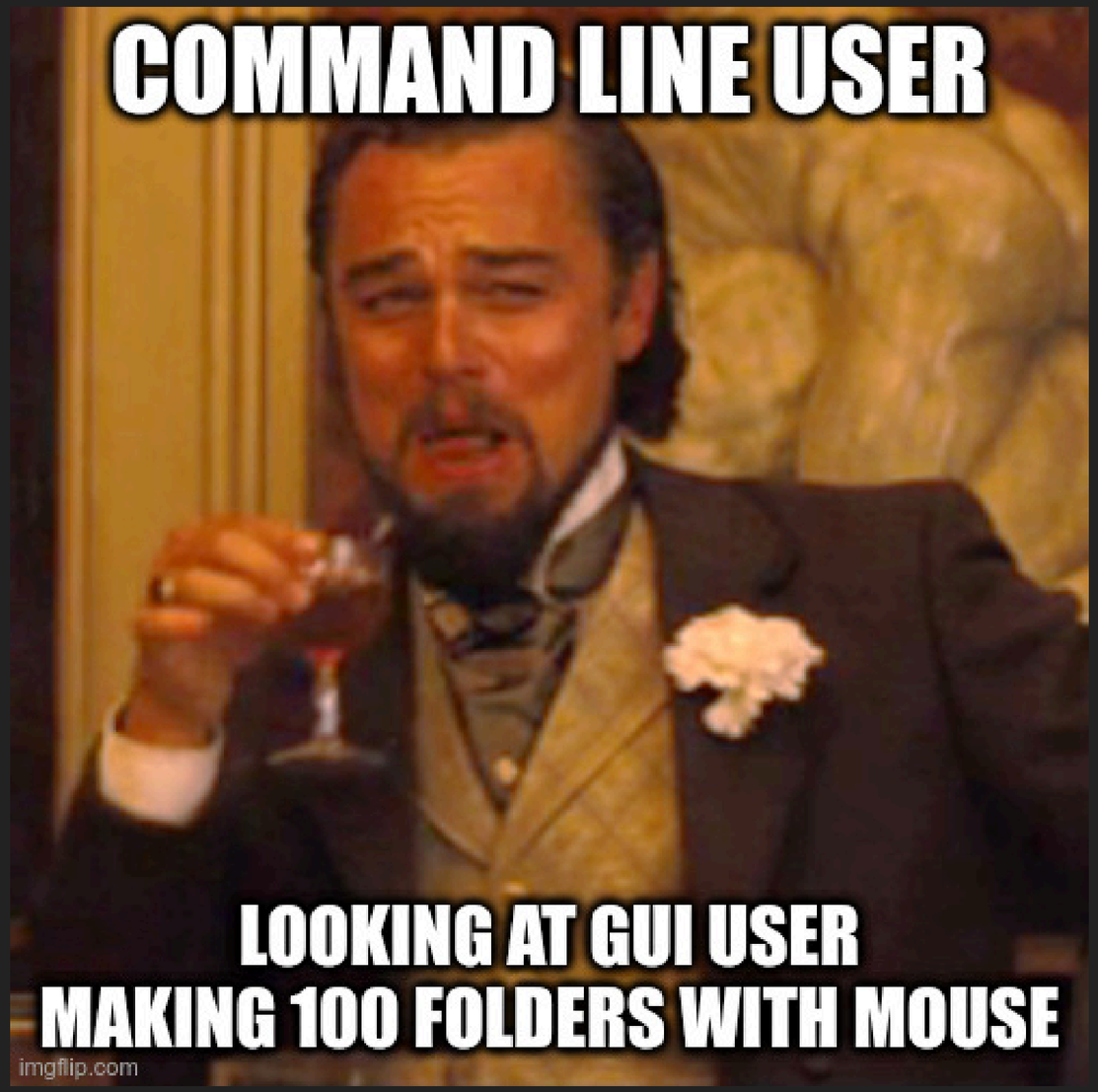
PULL REQUEST

`main` veja zaščitena => potrebujemo odobritev.

```
λ git checkout -b moja-veja
λ git commit -am "Dodal sem novo funkcionalnost"
λ git push origin moja-veja
```

```
λ gh pr create --base main --head moja-veja \
    --title "HUDA FUNKCIONALNOST" \
    --body "Ta funkcionalnost nas bo obogatila za 1000€"
```

PULL REQUEST



GIT RESET

GIT RESET

```
commit a1af9a2ce1e46f336e2a904149ded1078eedd43e
Author: Gašper Spagnolo <gasper.spagnolo@outlook.com>
Date:   Wed Dec 13 00:09:41 2024 +0100

    Modify example.txt in main with conflicting content

commit 2f12b6c100b20a7e1fc0ce64a902a3109e6edaee (feature-branch)
Author: Gašper Spagnolo <gasper.spagnolo@outlook.com>
Date:   Wed Dec 13 00:09:24 2024 +0100

    Modify example.txt in feature-branch

commit 9fdd5980c45b35dcbe7d0b84d9a82fa3e3babeb6
Author: Gašper Spagnolo <gasper.spagnolo@outlook.com>
Date:   Wed Dec 13 00:08:53 2024 +0100
```

GIT RESET

Po izboru se lahko vrnemo nazaj na stanje pred združevanjem vej.

```
λ git reset --hard 9fdd5980c45b35dcbe7d0b84d9a82fa3e3babeb6
```

```
commit 9fdd5980c45b35dcbe7d0b84d9a82fa3e3babeb6 (HEAD -> main)
Author: Gašper Spagnolo <gasper.spagnolo@outlook.com>
Date:   Wed Dec 13 00:08:53 2024 +0100

    Add example.txt with initial content
```

In sedaj HEAD kaže na commit, ki smo ga izbrali.

GIT DIFF

Pokaže razliko med trenutnim stanjem in stanjem v izbranem commitu, v primeru da commmit ni specificiran, avtomatsko vzame HEAD.

Dodajmo nekaj v datoteko `example.txt`:

```
λ git diff

diff --git a/example.txt b/example.txt
index 8430408..894d3bf 100644
--- a/example.txt
+++ b/example.txt
@@ -1 +1,4 @@
 Initial content
+
+
+Nekaj sem dodal v example.txt
```

GIT DIFF

Primer uporabe `git diff` med dvema različnima uveljavitvima.

```
λ git diff e60b7db57 8a79fbcae

diff --git a/prva_datoteka.txt b/prva_datoteka.txt
index 7305551..9349f3f 100644
--- a/prva_datoteka.txt
+++ b/prva_datoteka.txt
@@ -1,3 +1,3 @@
 #TO JE PRVA DATOTEKA

- 1337 feature\n 1338 feature\n 1339 feature
\ No newline at end of file
+ 1337 feature\n 1338 feature
\ No newline at end of file
```

GIT BLAME


POINTED OUT A BUG DURING CODE REVIEW
`GIT BLAME` SHOWS ME AS AUTHOR
imgflip.com

GIT BLAME

Recimo da nas zanima, kdo vse in kdaj je spreminjal datoteko `prva_datoteka.txt`:

```
λ git blame -l prva_datoteka.txt
```

```
^e44d7 (Gašper Spagnolo 2024-12-12 22:02:03 +0100 1) #TO JE PRVA
634d79 (Gašper Spagnolo 2024-12-12 23:06:49 +0100 2)
b4fac5 (Gašper Spagnolo 2024-12-12 23:35:06 +0100 3)  1337 featu
```

GIT BLAME NAPREDNO

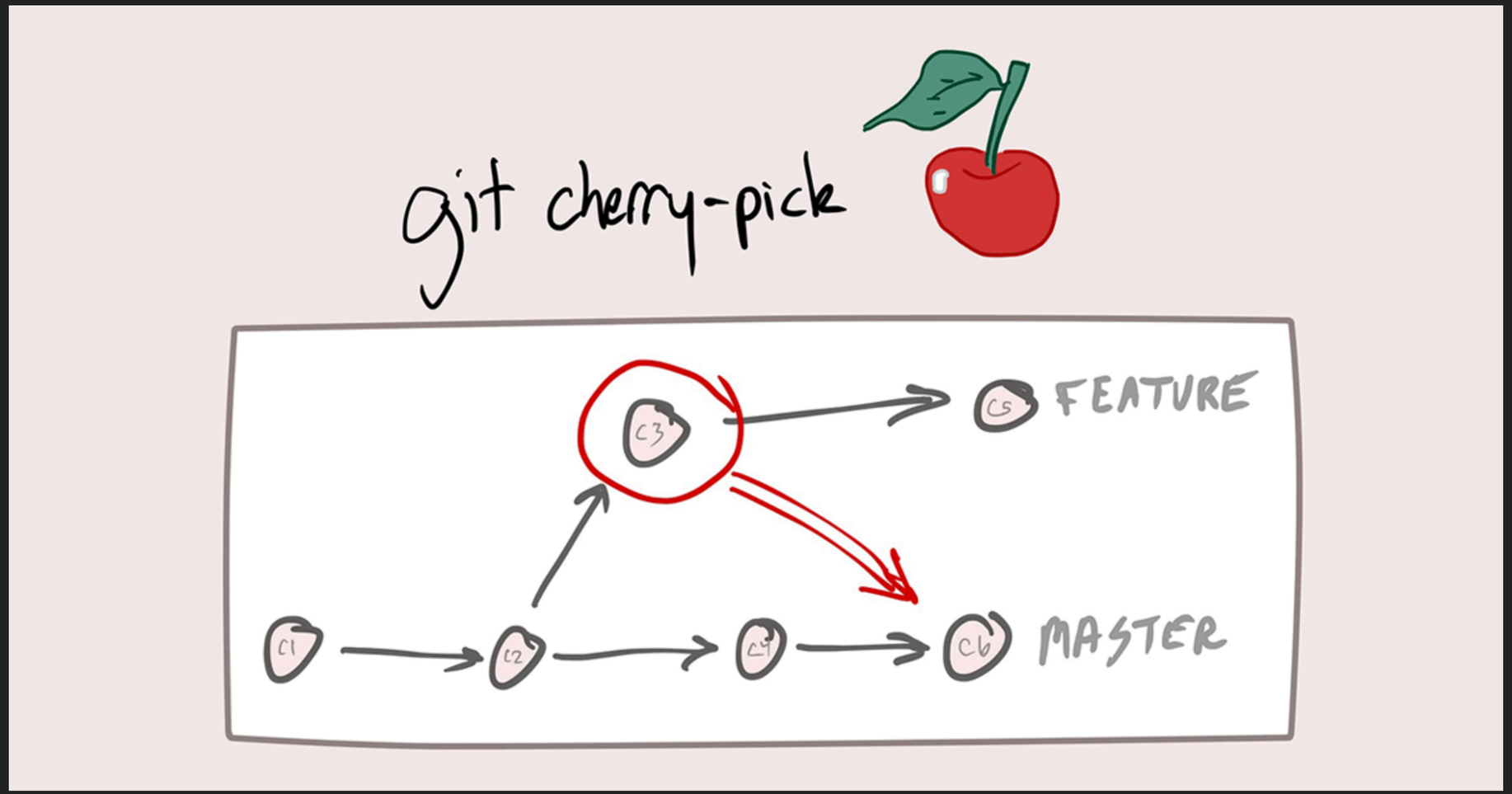
```
λ git -l blame <commit-hash>..HEAD prva_datoteka.txt
```

```
λ git blame -l --author="Author Name" prva_datoteka.txt
```

```
λ git blame -l -w prva_datoteka.txt
```

CHERRY PICK

Cherry pick in a nutshell:



CHERRY PICK PRIMER

```
λ git checkout main
λ git checkout -b cpp
```

```
λ echo "This is the first change." > file.txt
λ git add file.txt
λ git commit -m "First change in cpp branch"
```

```
λ echo "This is the second change." >> file.txt
λ git add file.txt
λ git commit -m "Second change in cpp branch"
```

CHERRY PICK PRIMER

Izberemo commit, ki ga želimo cherry pickati:

```
λ git log
```

```
commit 88b2b262d71fbf82b5ceccd5cbd71a1f7fc0e605 (HEAD -> cpp)
Author: Gašper Spagnolo <gasper.spagnolo@outlook.com>
Date:   Wed Dec 13 00:52:32 2024 +0100

    Second change in cpp branch

commit b18a08c1602c623df9aed2bd245a3928e01a3edb
Author: Gašper Spagnolo <gasper.spagnolo@outlook.com>
Date:   Wed Dec 13 00:52:22 2024 +0100

    First change in cpp branch
```

Recimo da želimo prvo spremembo: `b18a...`

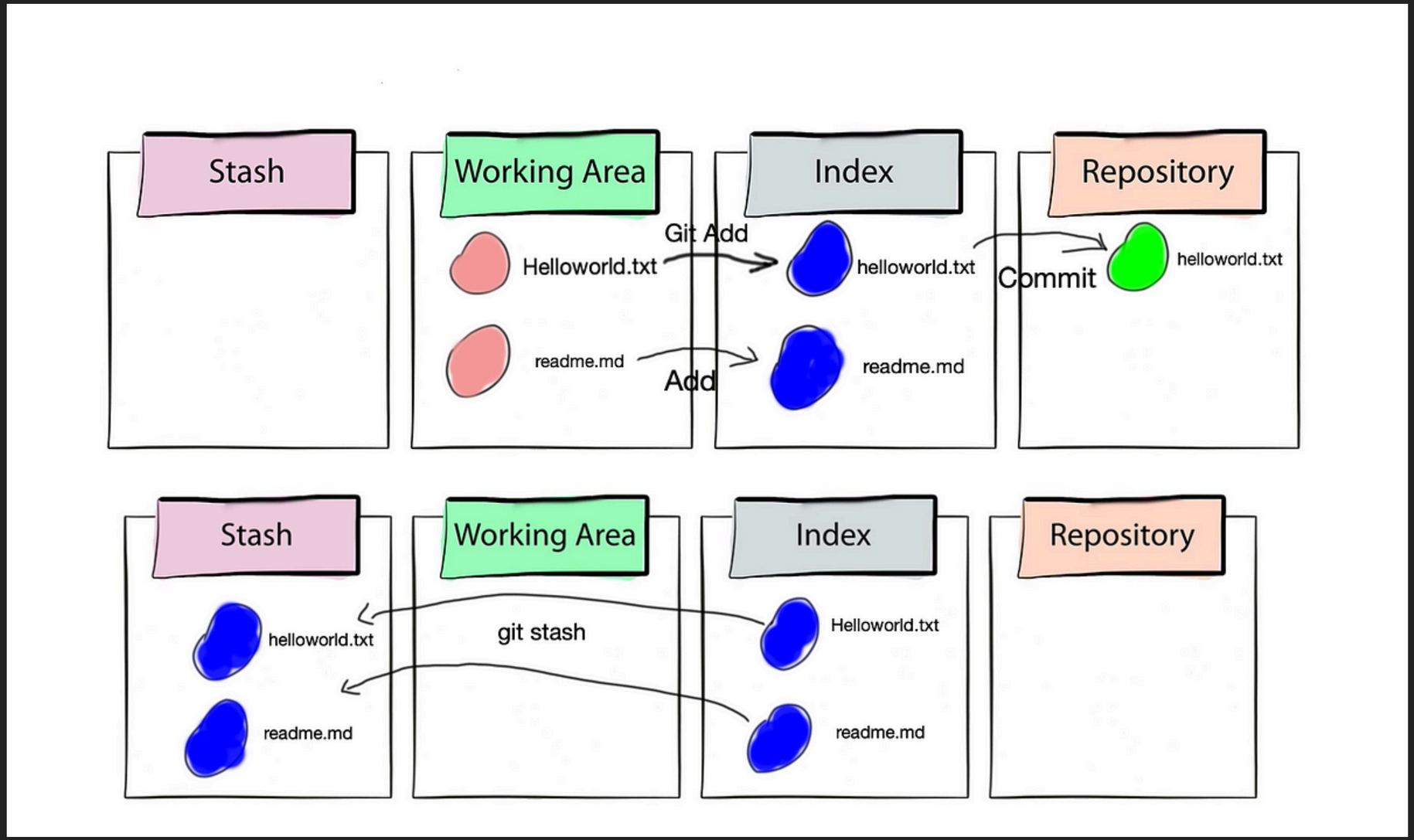
CHERRY PICK PRIMER

```
λ git checkout main
```

```
 git cherry-pick b18a08c1602c623df9aed2bd245a3928e01a3edb
[main 4e030e9] First change in cpp branch
 Date: Wed Dec 13 00:52:22 2024 +0100
 1 file changed, 1 insertion(+)
 create mode 100644 file.txt
```

```
λ git merge cpp
```

GIT STASH


Stash
Working Area
Index
Repository
Git Add
Helloworld.txt
helloworld.txt
Commit
helloworld.txt
readme.md
Add
readme.md
Stash
Working Area
Index
Repository
helloworld.txt
git stash
Helloworld.txt
readme.md
readme.md

GIT STASH

```
λ echo "To je datoteka na kateri zelim se delati pozneje" \
    > datoteka.txt
λ git add datoteka.txt
λ git stash
# or named stash
λ git stash save "stash name"
```

Potem pa nadaljujemo z našim delom.

GIT STASH

Ce zelimo shraniti tudi datoteke, ki jih nismo dodali v repozitorij, uporabimo:

```
λ git stash -u
```

Pregled stash-ov:

```
λ git stash list
```

GIT STASH

Ko pa želimo sedaj nadaljevati z delom na tej datoteki(ah), pa jo lahko pridobimo iz stash-a:

```
λ git stash pop
```

Ali ce imamo vec stash-ov:

```
λ git stash apply stash@{index}
```

GIT STASH

Brsianje specifičnega stash-a:

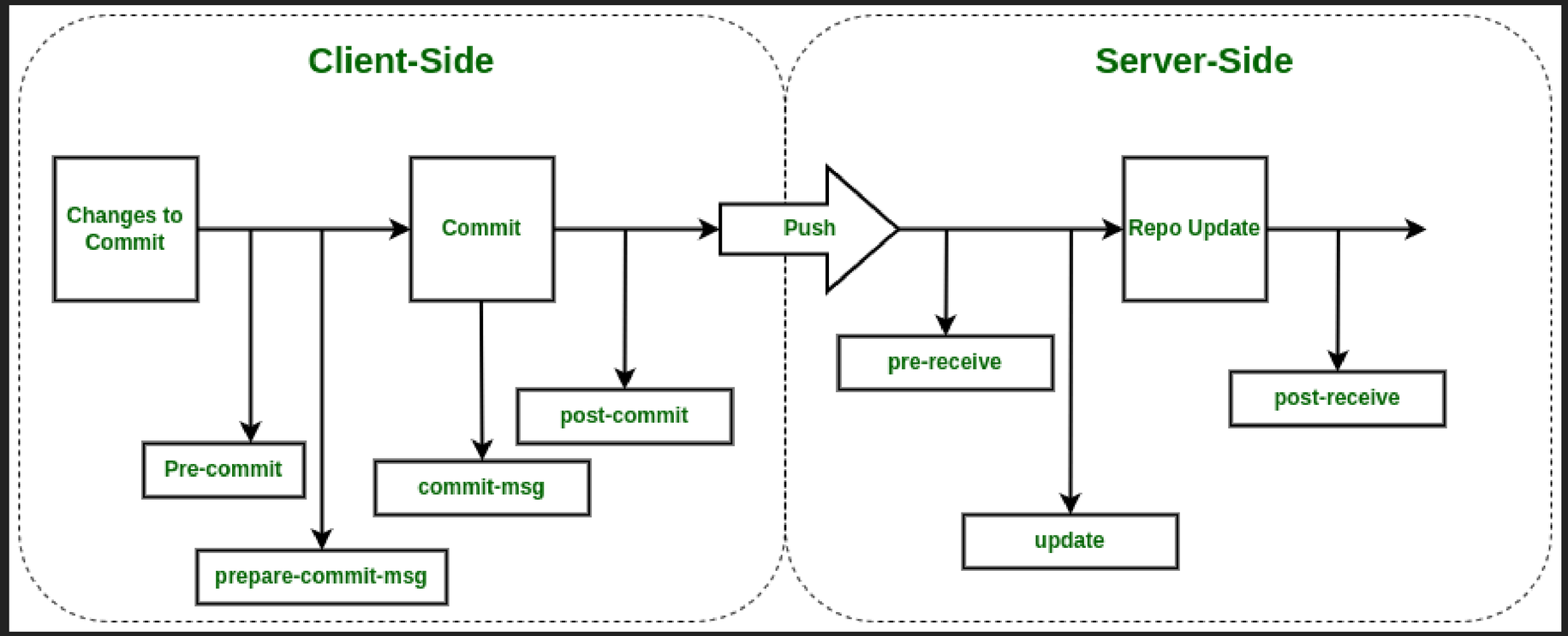
```
λ git stash drop stash@{index}
```

Brisanje vseh stash-ov:

```
λ git stash clear
```

ADVANCED GIT

HOOKS


Client-Side
Server-Side
Changes to Commit
Commit
Push
Repo Update
Pre-commit
prepare-commit-msg
commit-msg
post-commit
pre-receive
update
post-receive

EXAMPLE HOOK

vi .git/hooks/pre-commit

```
#!/bin/sh
# pre-commit

if git diff --cached | grep -i 'TODO'
then
  echo "Please remove TODOs before committing."
  exit 1
fi
```

chmod +x .git/hooks/pre-commit

UPORABA HOOKOV

- Preverjanje kode pred commitom (zagon testov, linterjev, formaterjev,..)
- Preverjanje commit sporočil
- Obvescanje kolaboratorjev ob commitu
- Avtomatizacija dela po prejemu sprememb (npr. buildanje, deployanje,..)
- Zascita glavne veje pred neposrednim commitom

HOOKS


Always remember to do things well
Use continuous integration to notice wrong stuff
Use pre-commit hooks to not even commit wrong stuff
src: @ma_salmon

GIT CONFIG

Pomaga nam izboljšati uporabniško izkušnjo z Gitom.

```
λ git config --global user.name "Your Name"
λ git config --global user.email "your_email@example.com"
λ git config --global init.defaultBranch main
λ git config --global core.editor "code --wait" # set vscode as
```

`--global` populira `~/.gitconfig`.

ALIASES


Add loads of aliases to ~/.bashrc
Save loads of time using aliases
Log into another machine
Can't remember how to use the regular commands

GIT ALIASES

```
λ git config --global alias.a "add ."
λ git config --global alias.cm "commit -m"
λ git config --global alias.s "status"
λ git config --global alias.lg "log --graph --oneline --decorate
λ git config --global alias.co "checkout"
λ git config --global alias.br "branch"
λ git config --global alias.undo "reset HEAD~1 --mixed"
```

GIT FSCK

Preveri integriteto repozitorija. Ali vse imajo vse datoteke pravilni checksum, ali je prislo do kakšne napake pri commitu, itd.

```
λ git fsck -v
Checking object directory
Checking blob 00ec0a7b0e8facc65b1935d8727920a6e265567c
Checking commit 05985f140657c479622df038c40ddf7f08ea84da
Checking tree 0d4eeab809bbf3fe1581e2149e6b314c2ac9ddfa
Checking HEAD link
Checking reflog 0000000000000000000000000000000000000000->e44d7e
Checking cache tree
Checking connectivity (128 objects)
Checking 00ec0a7b0e8facc65b1935d8727920a6e265567c
....
```

ZABAVNI PRIMER IZ PRAKSE

```
git fsck --full --no-reflogs --unreachable \
    --lost-found | grep commit | cut -d " " -f 3 | xargs -n 1 \
    git log -n 1 --pretty=oneline | grep "<commit-message>"
```

GIT GC

Git garbage collector. Počisti ravno tisto, kar smo na prejsnem slide-u reševali ;).

```
λ git gc
Enumerating objects: 50, done.
Counting objects: 100% (50/50), done.
Delta compression using up to 4 threads
Compressing objects: 100% (33/33), done.
Writing objects: 100% (50/50), done.
Total 50 (delta 10), reused 0 (delta 0), pack-reused 0
```

Počisti vse nepotrebne objekte, ko se jih nabere 256.

```
git config --global gc.auto 256
```

WRAP IT UP

HVALA ZA POZORNOST

DragonSec SI

@DragonSec_SI

info@dragonsec.si

JOIN US

join@dragonsec.si

DragonSec SI