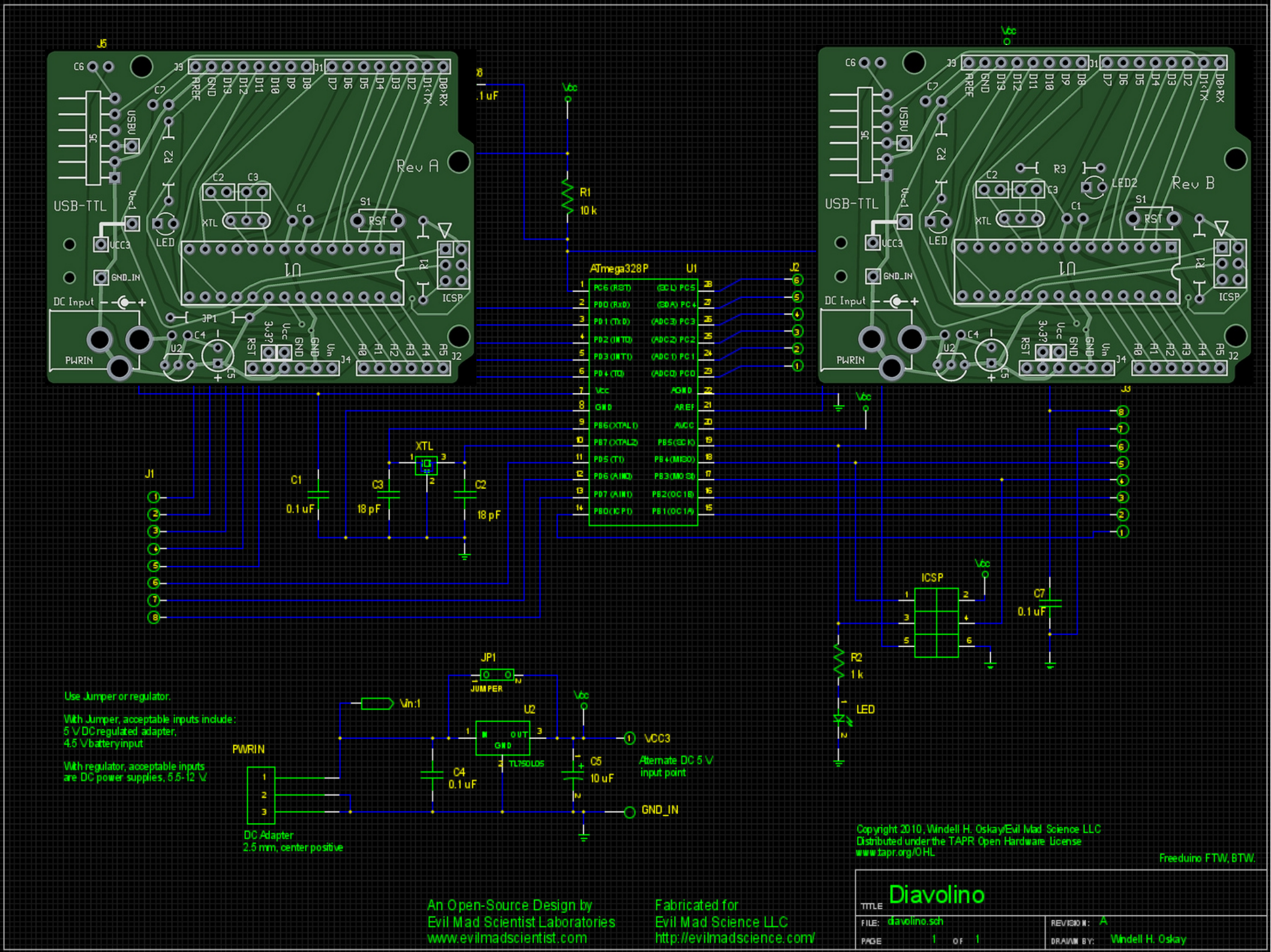
Správa verzí + GIT

Petr Škoda

Faculty of Information Technology, Brno University of Technology

Praktické aspekty vývoje software, léto 2016

14. 3. 2016


Rev A
Rev B
USB-TTL
DC Input
PWRIN
ICSP
LED
LED2
XTL
RST
ATmega328P
U1
J1
J2
J3
J4
J5
C1
0.1 uF
C2
18 pF
C3
18 pF
C4
0.1 uF
C5
10 uF
C7
0.1 uF
R1
10 k
R2
1 k
JP1
JUMPER
U2
TL750L05
Vin:1
VCC3
GND_IN
Alternate DC 5 V
input point
Use Jumper or regulator.
With Jumper, acceptable inputs include:
5 V DC regulated adapter,
4.5 V battery input
With regulator, acceptable inputs
are DC power supplies, 5.5-12 V.
DC Adapter
2.5 mm, center positive
Copyright 2010, Windell H. Oskay/Evil Mad Science LLC
Distributed under the TAPR Open Hardware License
www.tapr.org/OHL
Freeduino FTW, BTW.
An Open-Source Design by
Evil Mad Scientist Laboratories
www.evilmadscientist.com
Fabricated for
Evil Mad Science LLC
http://evilmadscience.com
Diavolino
FILE: diavolino.sch
REVISION: A
PAGE 1 OF 1
DRAWN BY: Windell H. Oskay

Obsah

Pojmy a motivace

Pojmy

- RCS – Revision Control system
- VCS – Version Control system
- Česky – systém pro správu verzí (SSV)

Motivace

- Historie změn
- Paralelní vývoj
- Spolupráce více vývojářů
- Záloha

Historie

Lokální

1972 SCCS (verzování v jednom adresáři)

1980 RCS (základní podpora pro paralelní vývoj)

Centralizované

1986 CVS (podpora spolupráce)

1999 Subversion SVN (odstranění hlavních chyb CVS)

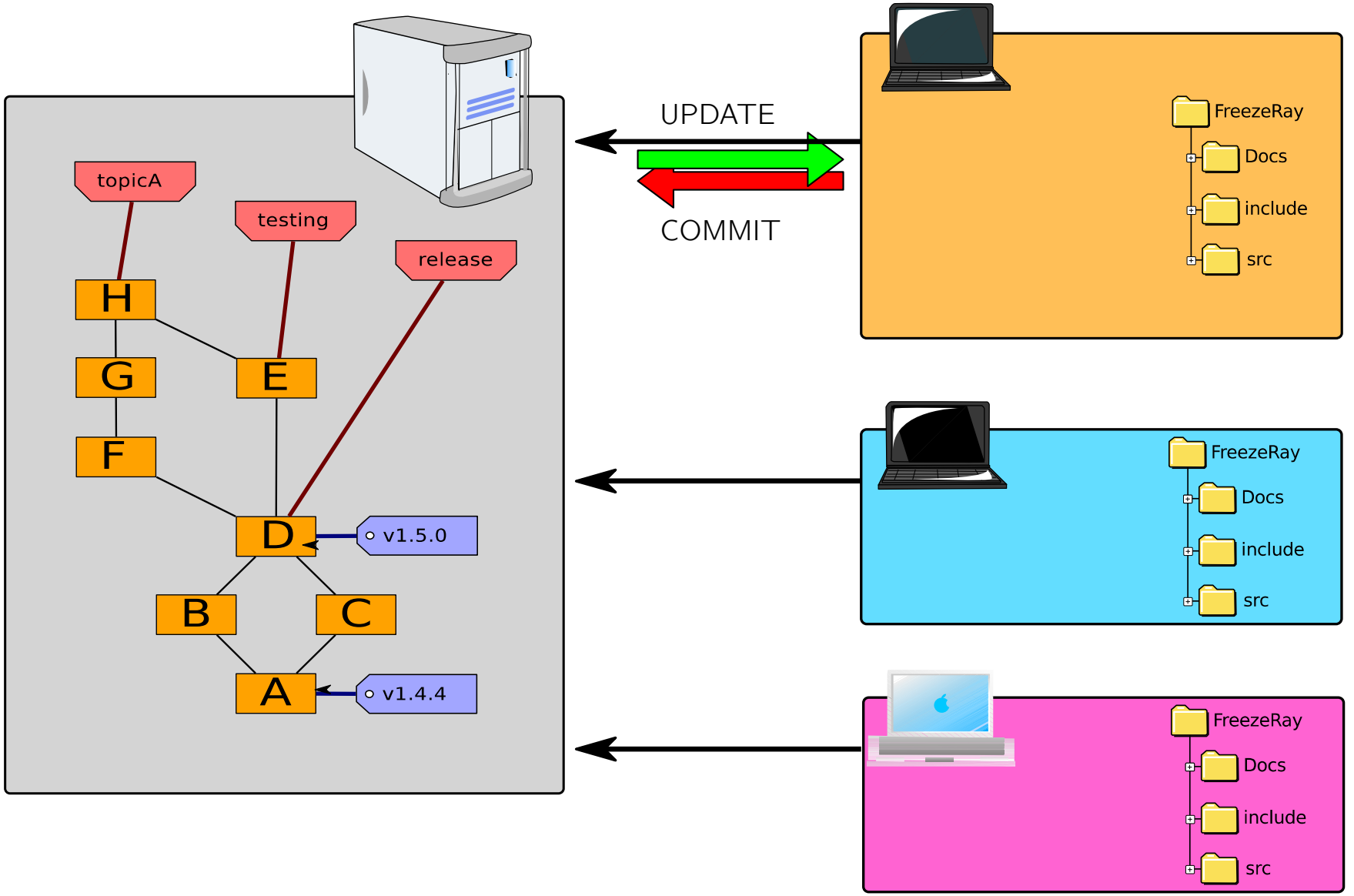
Distribuované

2001 Arch, Monotone

2002 Darcs

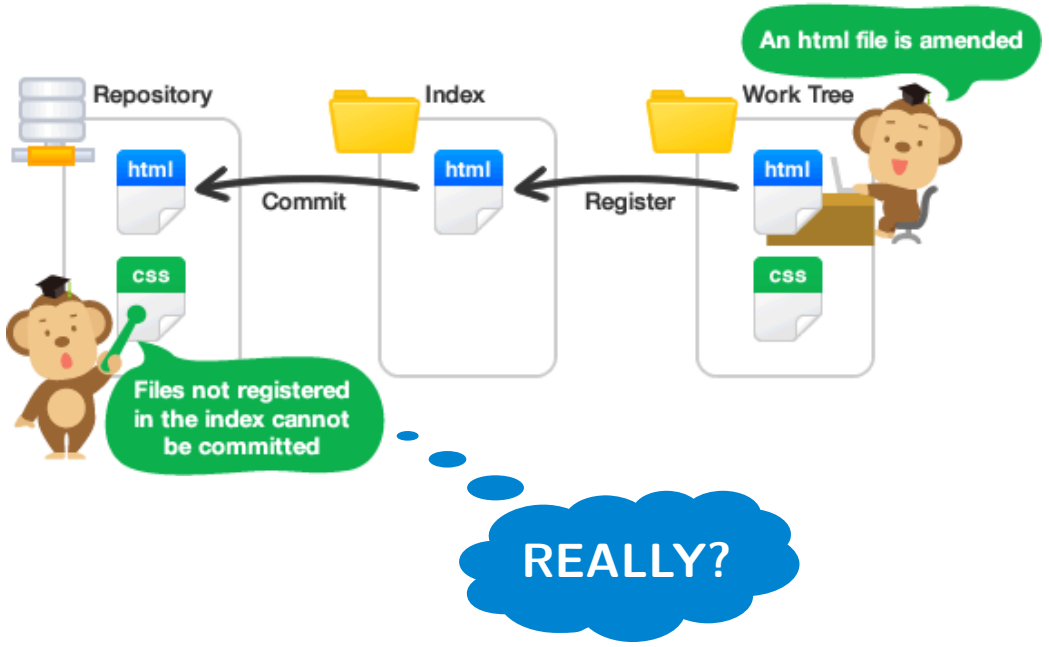
2005 **Git**, Mercurial (hg), Bazar (bzr)

Základní myšlenky



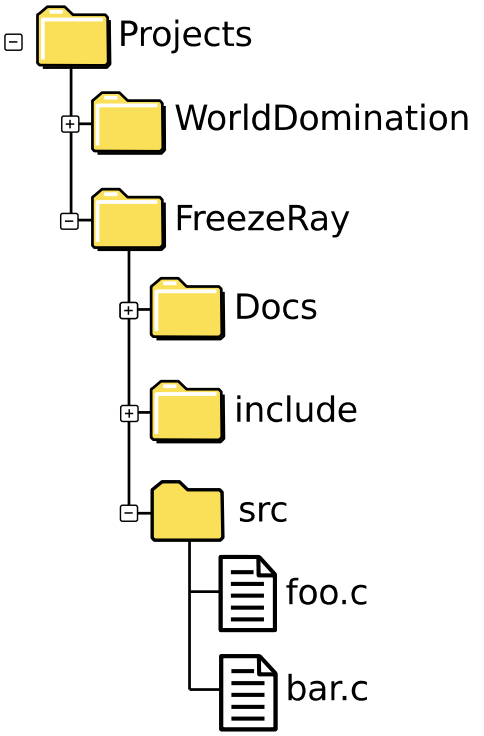
Součásti SSV

- Pracovní strom (working tree/working copy)
- Vyčkávací prostor (staging area/index)
- Repozitář/e (repository)



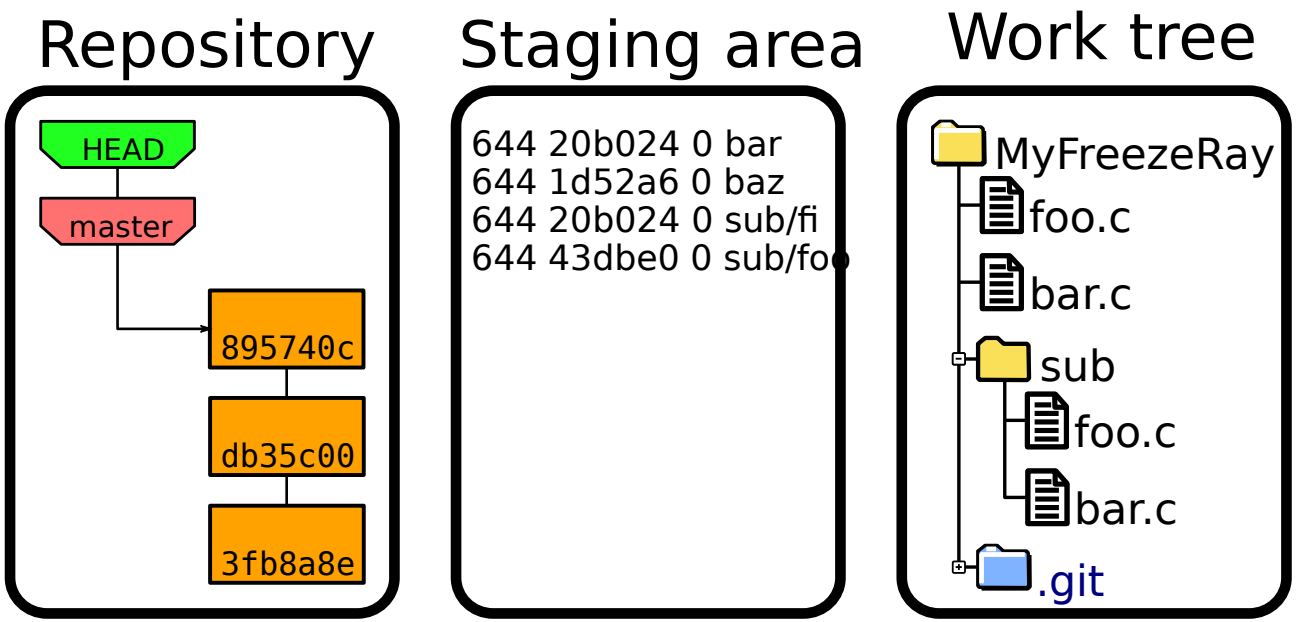
Součásti SSV – Pracovní strom

- Working tree/working copy
- Adresářová struktura projektu
- Na lokálním disku uživatele
- Zde jsou prováděny úpravy
- Obvykle obsahuje také metadata SSV
- Je odvozen z některé verze v repozitáři



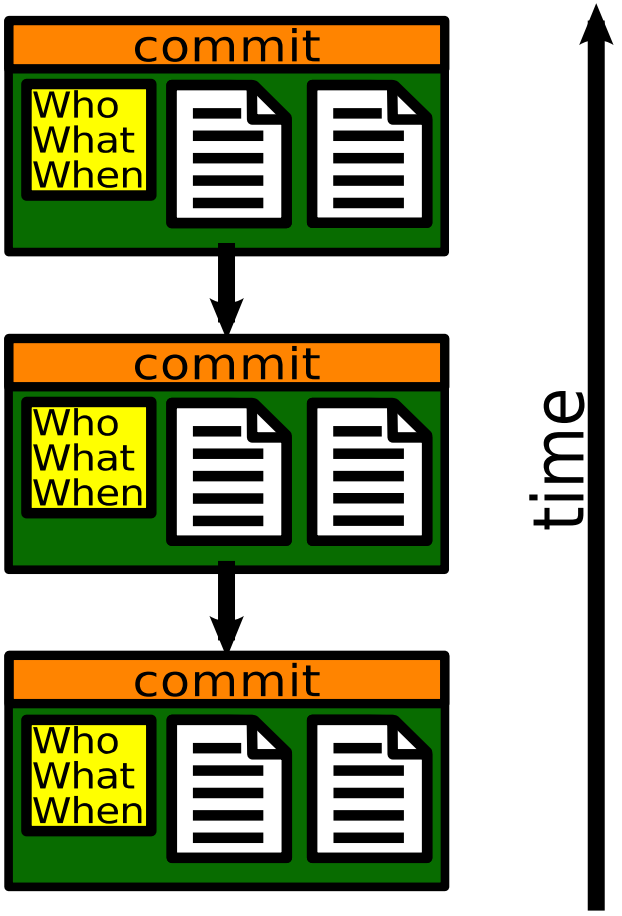
Součásti SSV – Vyčkávací prostor

- Staging area/index
- Objekty a operace určené k uložení do repozitáře
  - Přidané soubory
  - Odstranění souborů
  - Přejmenování souborů
  - Upravené soubory
- Tvoří jeden balík změn – jednu úpravu/verzi/revizi
- Po odeslání do repozitáře je obvykle vyprázdněn



Součásti SSV – Repozitář

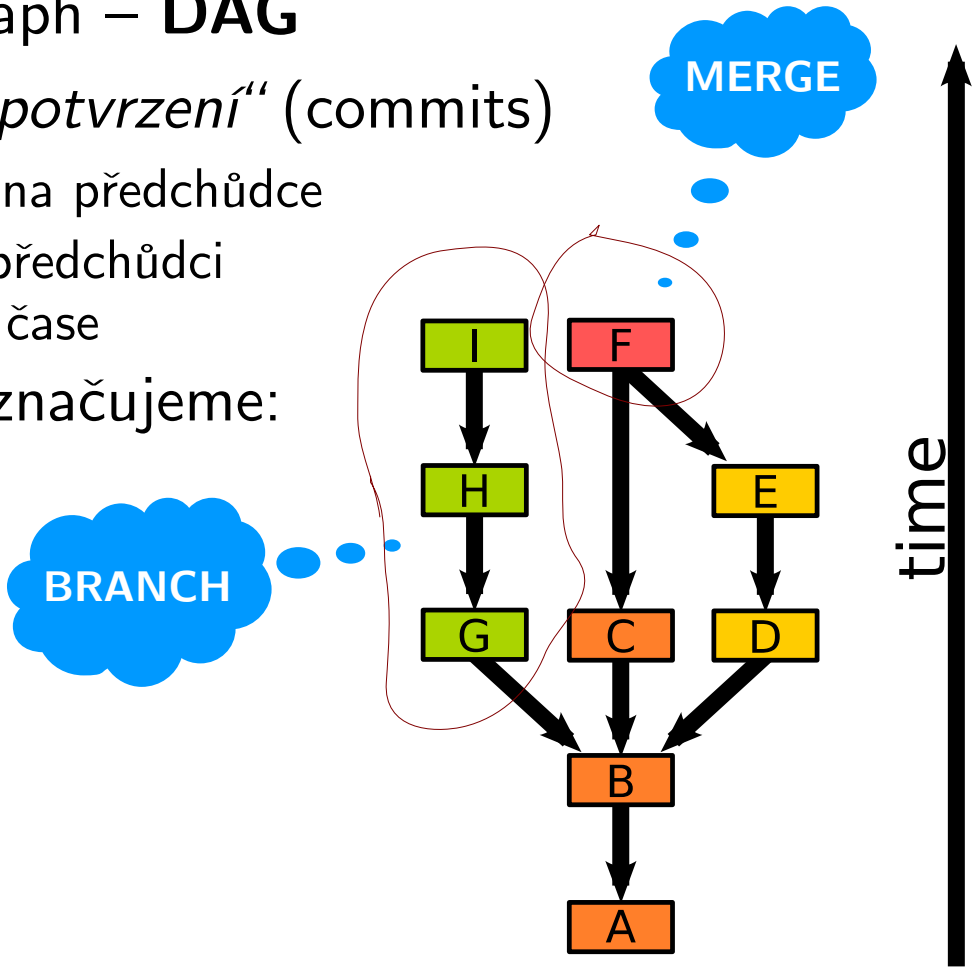
- Nejdůležitější součást
- Podstatou je DB
- Obsahuje
  - Obsahy souborů
  - Potvrzení (balíky změn/commits)
  - **Historii projektu**



Součásti SSV – Repozitář

Historie

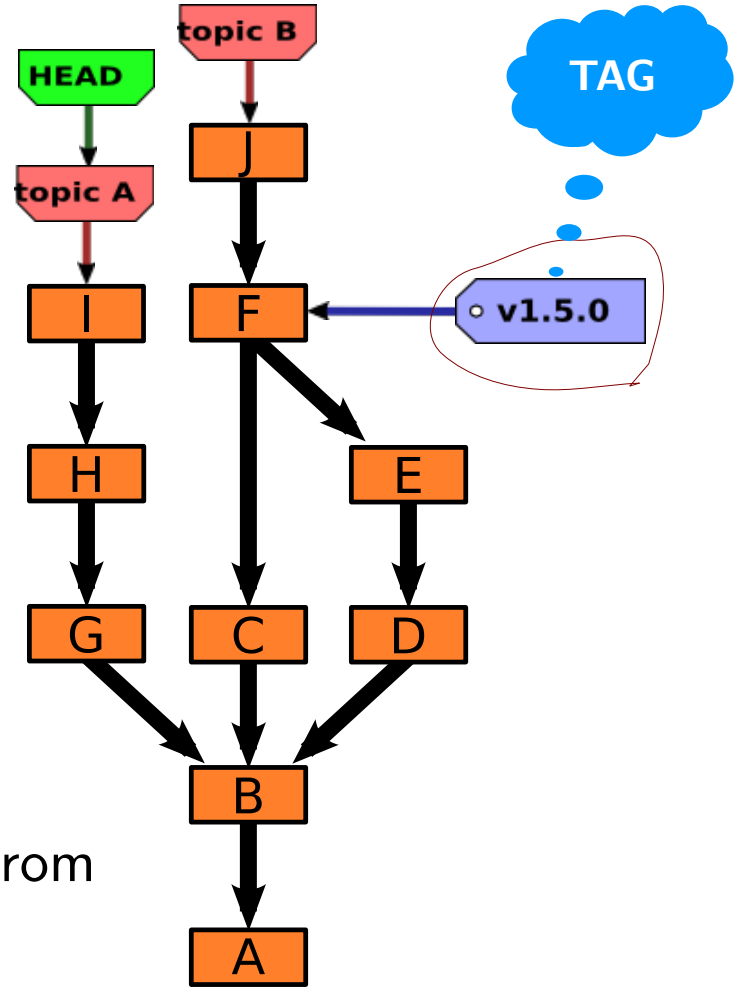
- Tvoří Directed Acyclic Graph – **DAG**
- Skládá se z jednotlivých „*potvrzení*“ (commits)
  - Propojeny pomocí odkazů na předchůdce
  - Každé potvrzení se svými předchůdci tvoří jeden stav projektu v čase
- Vybrané „vzory“ v grafu označujeme:
  - Větve (*branches*)
    - „Rozdvojení“ od přímé posloupnosti
  - Spojení (*merges*)
    - *Spojení dvou větví*



Součásti SSV – Repozitář

Reference

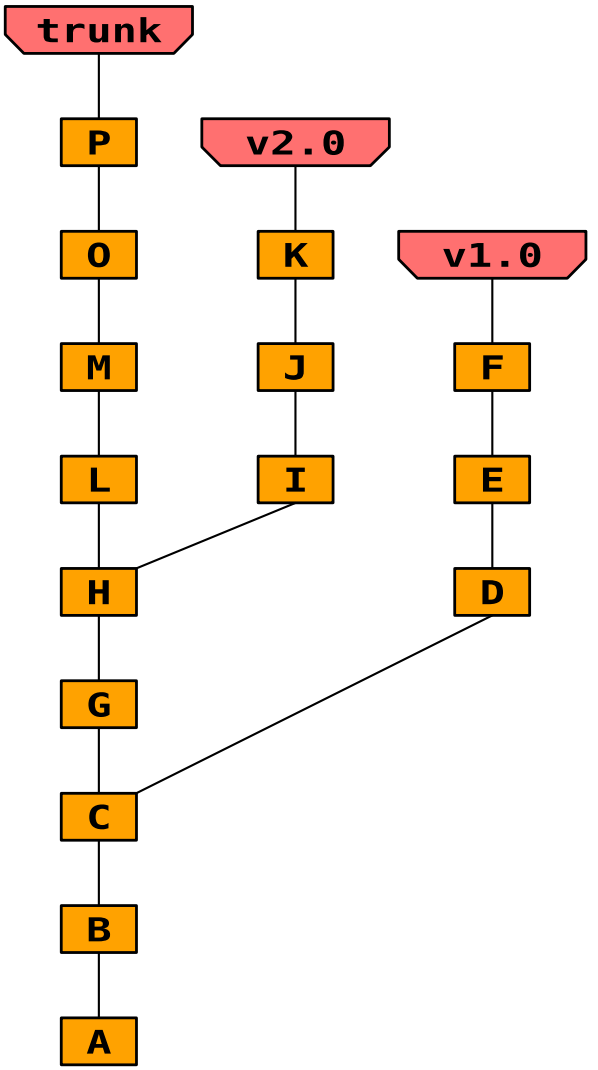
- Ukazují na *potvrzení* (commit) v rámci DAG
- Značky (tags)
  - Uživatelem definované
  - Například jména verzí
- Větve (branches)
  - „**topic A**“, „**topic B**“
  - Ukazuje na aktuálně poslední potvrzení (commit) dané větve
- HEAD
  - Ukazuje na potvrzení (commit), jež byl zdrojem pro aktuální pracovní strom



Pracovní postupy – tradiční

Tradiční

- **v1.0 a v2.0**
  - Staré verze aplikace
  - Musíme po jistou dobu udržovat opravami chyb
- **trunk**
  - Nové funkce
  - Opravy chyb



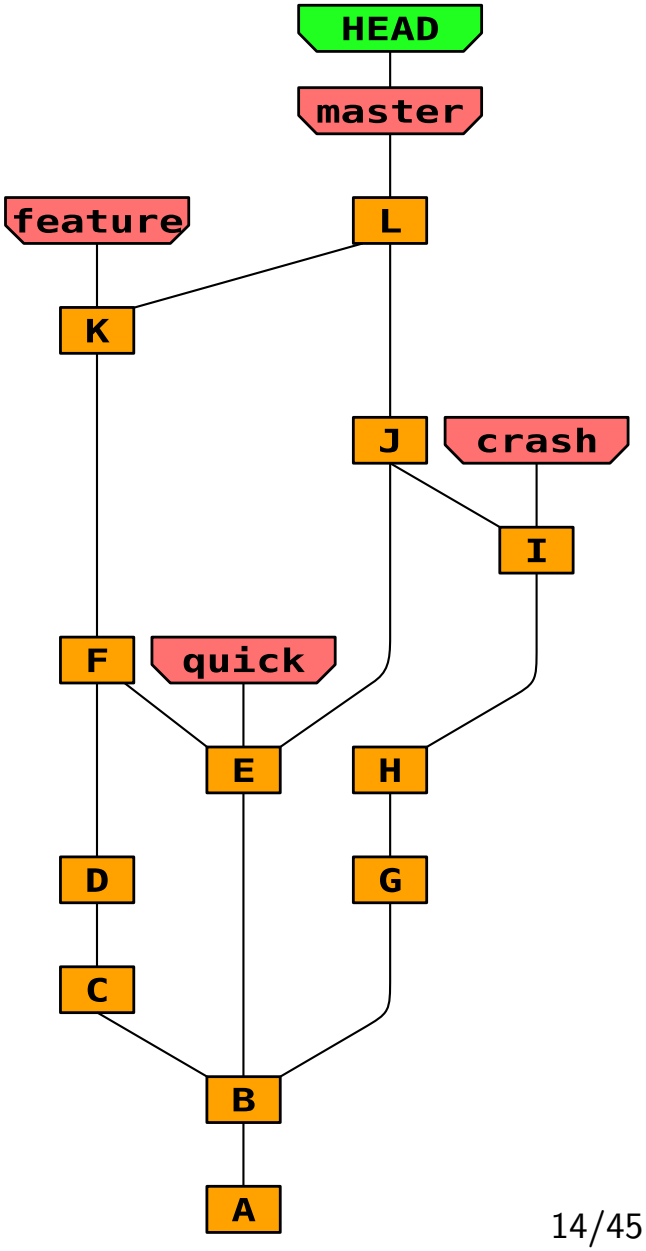
Pracovní postupy – tématické větve

Tématické větve

- Jedna větev pro každé téma
- Časté potvrzování (commits)
- Krátká životnost větví
- Dokončené téma spojíme (merge) do hlavní větve (master)

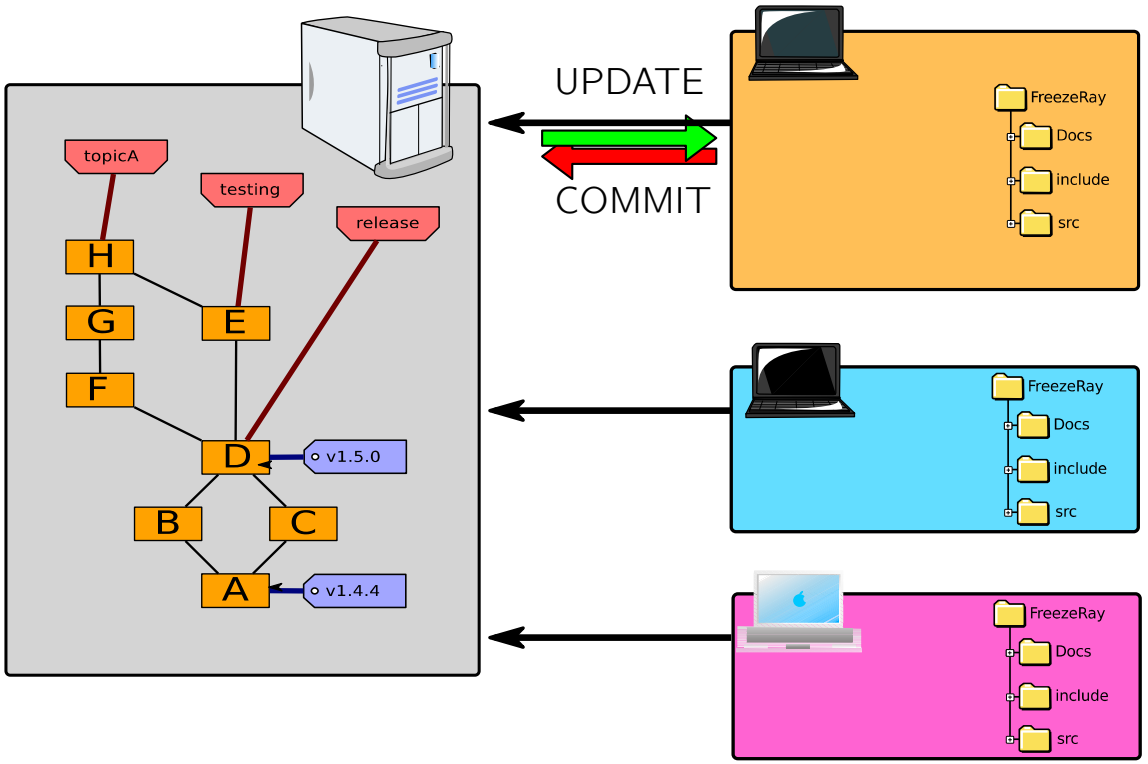
Porovnání s tradičním postupem

- Přehlednější
- Možná práce na více funkcích zároveň
- Spojení (merge) lze odkládat
- Vyžaduje rychlé přepínání větví



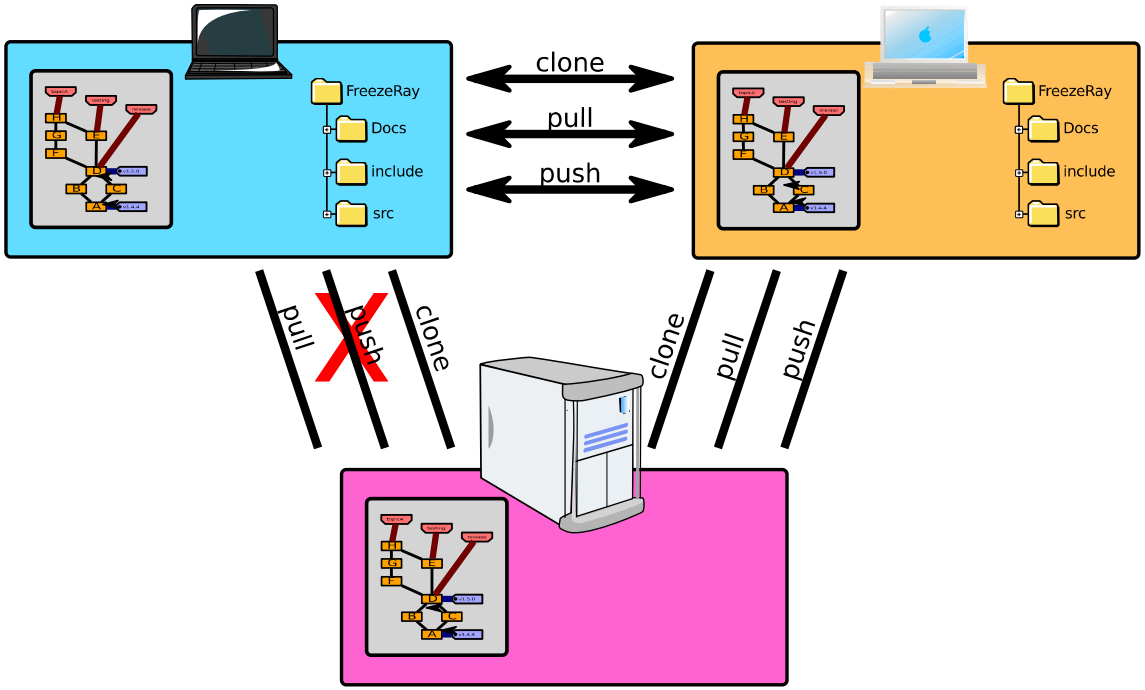
Centralizovaná správa verzí

- Většina operací vyžaduje server
- Vzdálené operace bývají pomalé
- Potvrzené změny MUSEJÍ být plně funkční a otestované
- Nutí odkládat potvrzení (commits)
- Je nutné řešit různá oprávnění uživatelů
- Server je úzké místo
- Obsahuje „*single point of failure*“



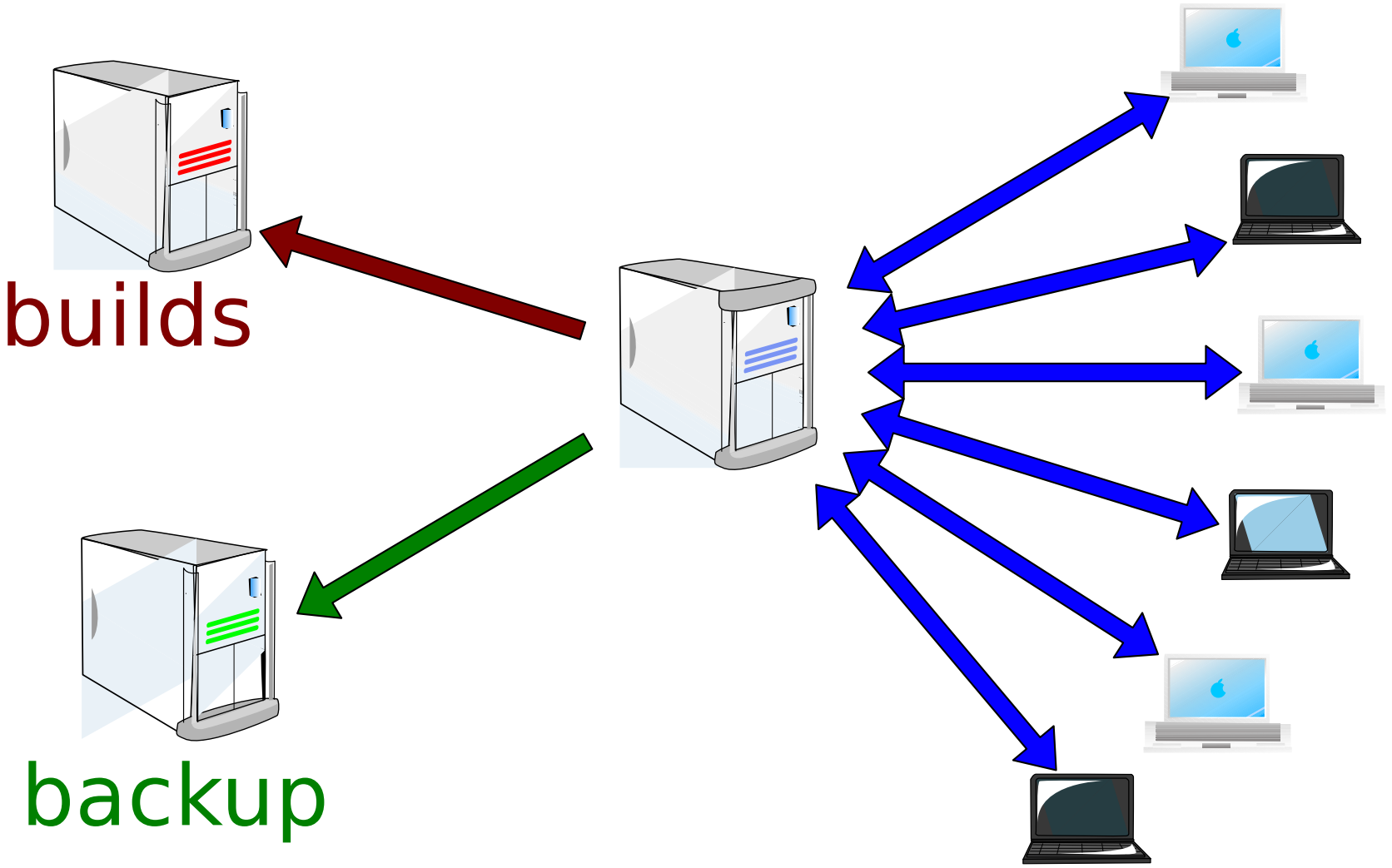
Distribuovaná správa verzí

- Každý může být server
- Neinvazivní „mikro“ potvrzení
  - Potvrzovat často
  - Potvrzovat malé změny
  - Nerozbijeme sestavení
- Práce plně offline
- Většina operací je bleskově rychlá
- Flexibilnější rozdělení zodpovědností
- Každý vývojář je záloha repozitáře
- Z principu nemá „*single point of failure*“



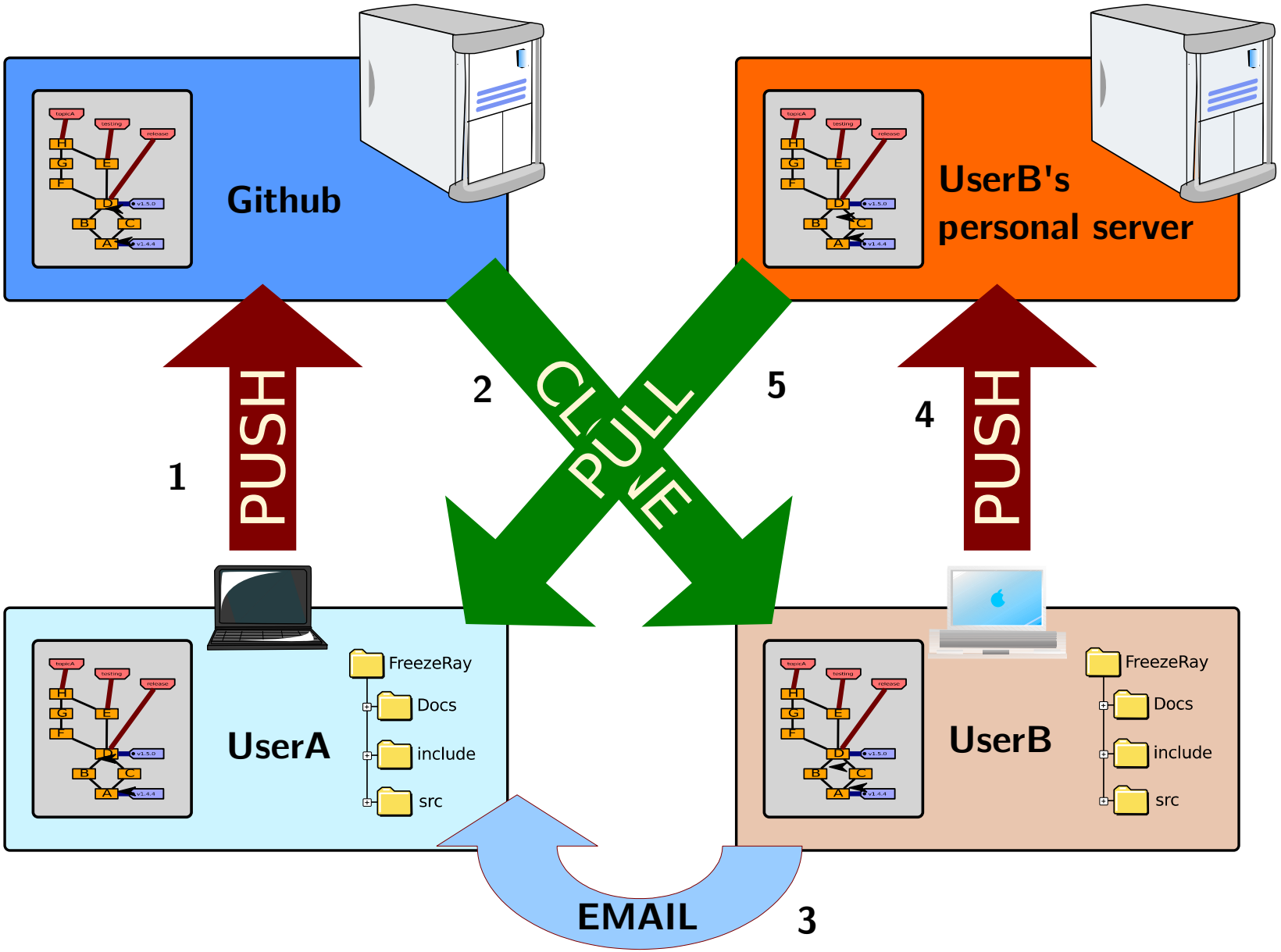
Jak probíhá spolupráce

- Centralizovaná topologie



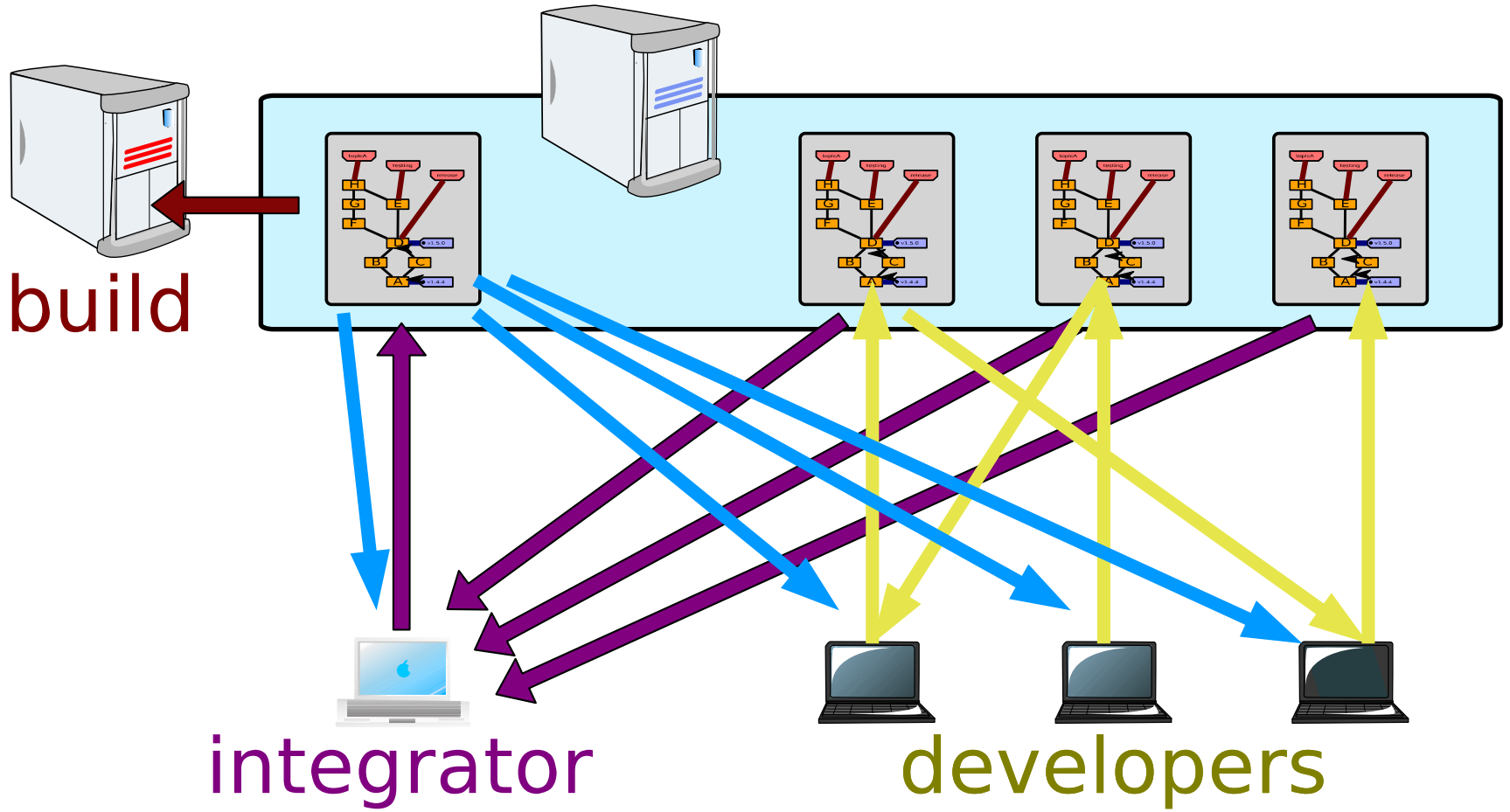
Jak probíhá spolupráce

- Distribuovaná topologie



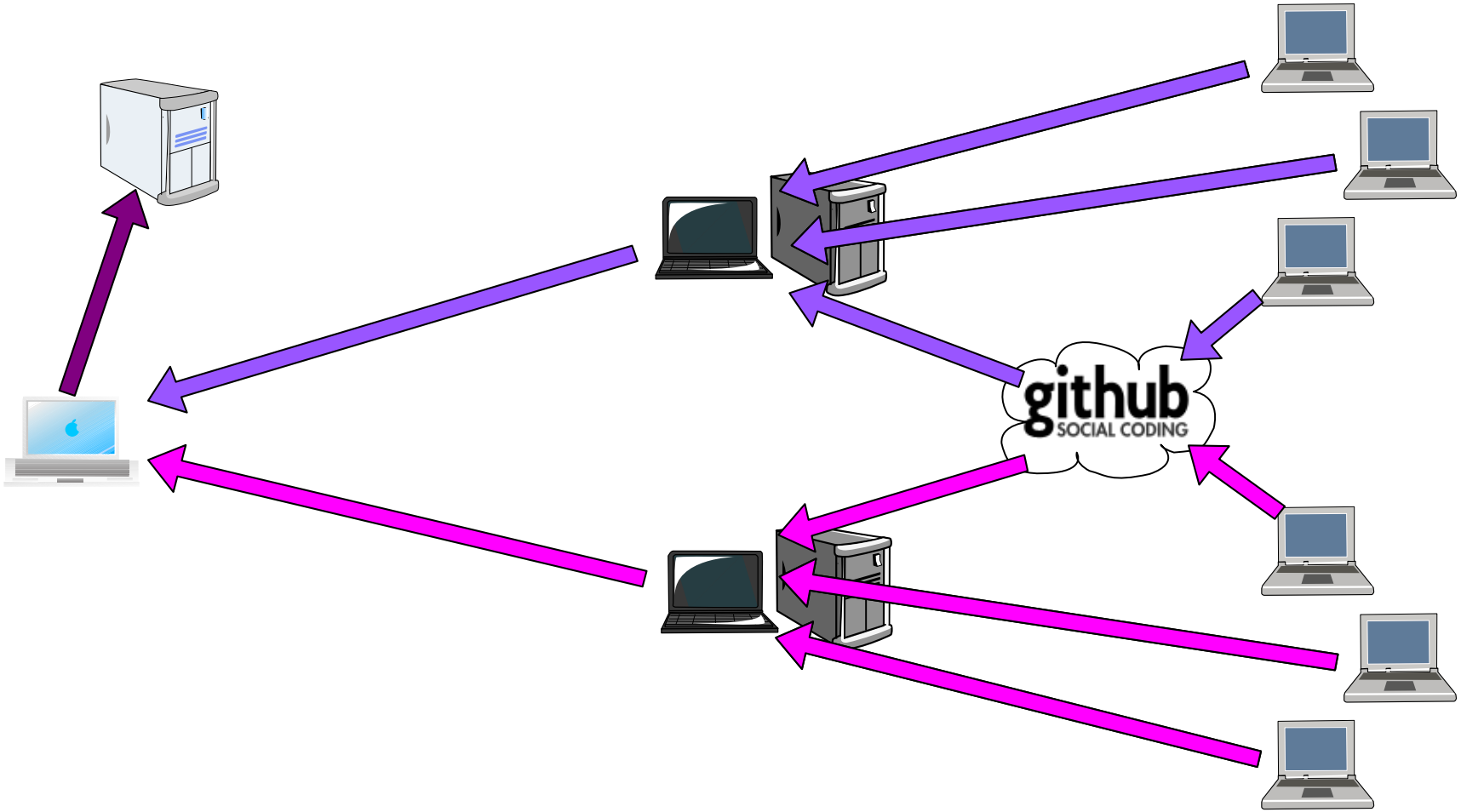
Topologie „Integrátor“

- Jeden zodpovědný integruje změny od vývojářů



Topologie „diktátor a poručníci“

- Vícevrstvá struktura zodpovědných
- Používá se při vývoji jádra Linuxu



Git

Základy práce s GIT vs. SVN

Založení repozitáře a pracovního stromu

- Inicializace (nový repozitář)

  `svnadmin create` (obvykle musí provést správce)

  `git init`

- Pracovní strom

  `svn checkout <URL>`

  `git clone <URL>`

Změny

- Soubor nový/zrušit/přejmenovat

  `svn [add|rm] <soubory>` (! přidává a odebírá z repozitáře)

  `git [add|rm|mv] <soubory>` (přidává a odebírá ze staging area)

- Commit

  `svn commit [<soubory>]` (změny jdou přímo na server)

  `git commit [-a|<soubory>]` (změny se uloží jen lokálně)

Základy práce s GIT vs. SVN

Informace o stavu

- Stav pracovního stromu

  ```
  svn status
  git status
  ```

  - Vypíše soubory:
    - Upravené (i neverzované) ve vyčkávacím prostoru
    - Upravené nepřidané do vyčkávacího prostoru
    - Konfliktní soubory po slučování
- Změny v pracovní kopii oproti repozitáři

  ```
  svn diff
  git diff
  ```
- Historie potvrzení (commits)

  ```
  svn log
  git log
  ```

  - Zobrazí seznam provedených potvrzení, umí různé formáty (např. přepínač --graph)

Základy práce s GIT vs. SVN

Odkazy

- Značkování (tagging)

```
svn copy .. <URL>/tags/<značka>
git tag <značka>
```

- Větvení (branching)

```
svn copy .. <URL>/branches/<větev>
git branch <větev>
```

Větve

- Přepínání

```
svn switch <URL>/branches/<větev>
git checkout <větev>
```

- Spojení

```
svn merge -r<rev>:<rev> <URL>/branches/<větev>
git merge <větev>
```

Základy práce s GIT vs. ~~SVN~~

Distribuované operace

- Podporuje pouze GIT (SVN je centralizovaný)
- Klonování (získání kopie celého repozitáře)

```
git clone <URL>
```

- Aktualizace ze vzdáleného repozitáře do aktuální větve

```
git pull [remote [remote_branch]]
```

- Sdílení (aktualizace do vzdáleného/jiného repozitáře)

```
git push [remote [remote_branch]]
```

Základní nastavení

- Jméno uživatele a email (nastavíme globálně pro aktuálního uživatele)

```
git config --global user.name "Firstname Lastname"
git config --global user.email "someone@foo.com"
```

Git

Vyčkávací prostor (staging area/index)

- Obsahuje změny v repozitáři připravené pro potvrzení
- Soubory jsou do vyčkávacího prostoru okopírovány
- Přidání změn pro příští potvrzení (commit):

  ```
  git add | git mv | git rm
  ```
- Potvrzení do lokálního repozitáře:

  ```
  git commit
  ```

Bez použití vyčkávacího prostoru

```
git commit -a | git commit <soubory>
```

(-a potvrdí všechny nové a upravené soubory v prac. stromu)

- Přepínání větví

  ```
  git checkout <branch>
  ```
- Získání starých revizí

  ```
  git checkout [-b <new_branch>] <commit>
  ```

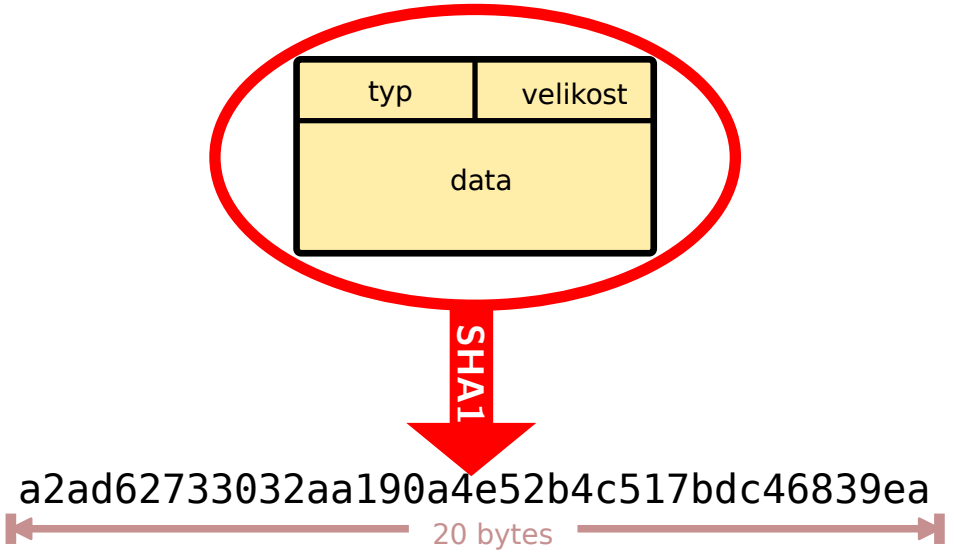
  (-b <new_branch> provede checkout do nové větve)

Git – metadata a repozitář

- Adresář v pracovním stromu *.git*
- Důležité soubory
  - *HEAD* – odkaz na zdroj aktuálního pracovního stromu v repozitáři
  - *config* – nastavení Gitu pro aktuální repozitář
  - *description* – popis repozitáře
  - *index* – odkaz na strom vyčkávacího prostoru (stromy viz dále)
- Důležité adresáře
  - *objects* – objekty repozitáře (commits, trees, blobs, tags)
    - Soubory pojmenované SHA1 jejich obsahu
    - Uspořádané v adresářích začínajících prvními znaky SHA1
  - *refs* – odkazy na větve
  - *hooks* – skripty pro vykonání před či po různých akcích

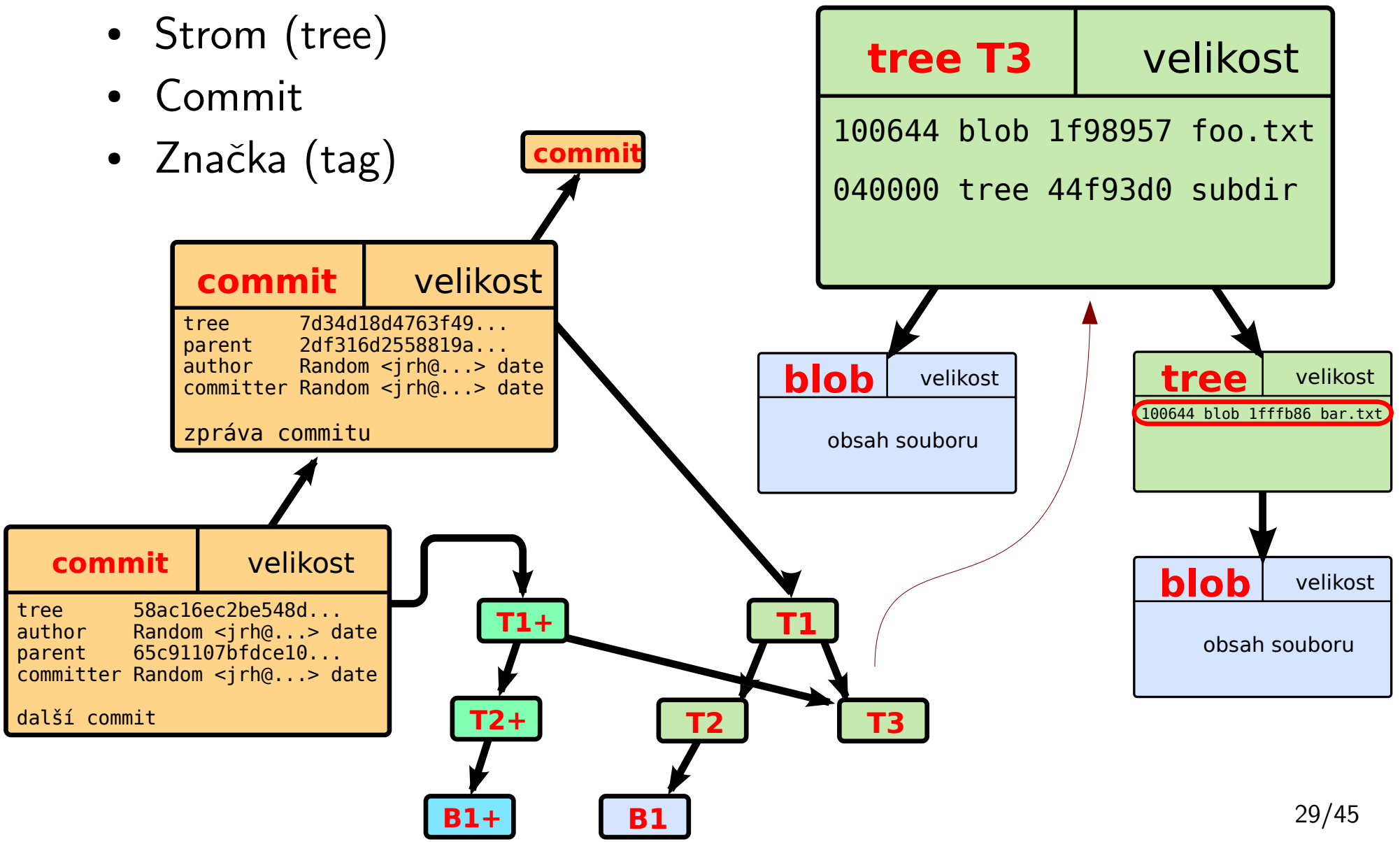
Git – objekty

- Soubory uložené v .git/objects
- Adresovatelné obsahem
- Stejný objekt nikdy neexistuje vícekrát
- Soubor objektu se jmenuje jeho SHA1 hašem
- Každý objekt obsahuje identifikátor typu (4 typy), velikost a data
- Obsah objektu nelze změnit, lze jen smazat celý objekt
- Potvrzení (commit) je také objekt
- *Potvrzení identifikujeme jakýmkoli unikátním prefixem jeho SHA1 haše*



Git – typy objektů

- Blob
- Strom (tree)
- Commit
- Značka (tag)



Git – commit

- *Otázka: Jaký je rozdíl mezi potvrzením v SVN a Git?*

- Co potvrzujeme
  - Vše co je ve vyčkávacím prostoru
    `git commit`
  - Všechny změny pracovního stromu proti repozitáři
    `git commit -a`
  - *Jak přidáváme soubory do vyčkávacího prostoru?*
    Pomocí git add nebo git mv.

- Co nepotvrzujeme
  - Můžeme nastavit soubory, které mají být ignorovány
  - Soubor **.gitignore** v pracovním stromu
  - Zadáváme relativní cesty k souborům/adresářům, masky s *

Git – nápověda

Když něco nevím, najdu to v nápovědě!

The GIT man pages are one almighty "f**k you". [Steve Bennett, 10 things I hate about Git]

- Dostupné příkazy Gitu

  `git help [--all]`
- Nápověda konkrétního příkazu

  `git <command> -h`

  `man git-<command>`

  `git help <command>`
- Dostupná nastavení pro *.git/config*

  `git help config`

```
git help
usage: git [--version] [--exec-path[=<path>]] [--html-path] [--man-path] [--info-
path]
           [-p|--paginate|--no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           [-c name=value] [--help]
           <command> [<args>]

The most commonly used git commands are:
   add        Add file contents to the index
   bisect     Find by binary search the change that introduced a bug
   branch     List, create, or delete branches
   checkout   Checkout a branch or paths to the working tree
   clone      Clone a repository into a new directory
   commit     Record changes to the repository
   diff       Show changes between commits, commit and working tree, etc
   fetch      Download objects and refs from another repository
   grep       Print lines matching a pattern
   init       Create an empty git repository or reinitialize an existing one
   log        Show commit logs
   merge      Join two or more development histories together
   mv         Move or rename a file, a directory, or a symlink
   pull       Fetch from and merge with another repository or a local branch
   push       Update remote refs along with associated objects
   rebase     Forward-port local commits to the updated upstream head
   reset      Reset current HEAD to the specified state
   rm         Remove files from the working tree and from the index
   show       Show various types of objects
   status     Show the working tree status
   tag        Create, list, delete or verify a tag object signed with GPG

See 'git help <command>' for more information on a specific command.
```

Git Cheat Sheet

Remember!
`git <COMMAND> --help`

Global configuration is stored in ~/.gitconfig.
`git config --help`

master is the default development branch.
origin is the default upstream repository.

Create

From existing data
```
cd ~/my_project_directory
git init
git add .
```

From existing repository
```
git clone ~/existing_repo ~/new/repo
git clone git://host.org/project.git
git clone ssh://user@host.org/project.git
```

Show

Files changed in working directory
`git status`

Changes made to tracked files
`git diff`

What changed between ID1 and ID2
`git diff <ID1> <ID2>`

History of changes
`git log`

History of changes for file with diffs
`git log -p <FILE> <DIRECTORY>`

Who changed what and when in a file
`git blame <FILE>`

A commit identified by ID
`git show <ID>`

A specific file from a specific ID
`git show <ID>:<FILE>`

All local branches
`git branch`
star (*) marks the current branch

Revert

Return to the last commited state
`git reset --hard`
This cannot be undone!

Revert the last commit
`git revert HEAD`
Creates a new commit

Revert specific commit
`git revert <ID>`
Creates a new commit

Fix the last commit
`git commit -a --amend`
(after editing the broken files)

Checkout the ID version of a file
`git checkout <ID> <FILE>`

Update

Fetch latest changes from origin
`git fetch`
(this does not merge them)

Pull latest changes from origin
`git pull`
(does a fetch followed by a merge)

Apply a patch that someone sent you
`git am -3 patch.mbox`
In case of conflict, resolve the conflict and
`git am --resolved`

Publish

Commit all your local changes
`git commit -a`

Prepare a patch for other developers
`git format-patch origin`

Push changes to origin
`git push`

Make a version or milstone
`git tag v1.0`

Branch

Switch to a branch
`git checkout <BRANCH>`

Merge BRANCH1 into BRANCH2
```
git checkout <BRANCH2>
git merge <BRANCH1>
```

Create branch BRANCH based on HEAD
`git branch <BRANCH>`

Create branche BRANCH based on OTHER and switch to it
`git checkout -b <BRANCH> <OTHER>`

Delete branch BRANCH
`git branche -d <BRANCH>`

Resolve merge conflicts

View merge conflicts
`git diff`

View merge conflicts against base file
`git diff --base <FILE>`

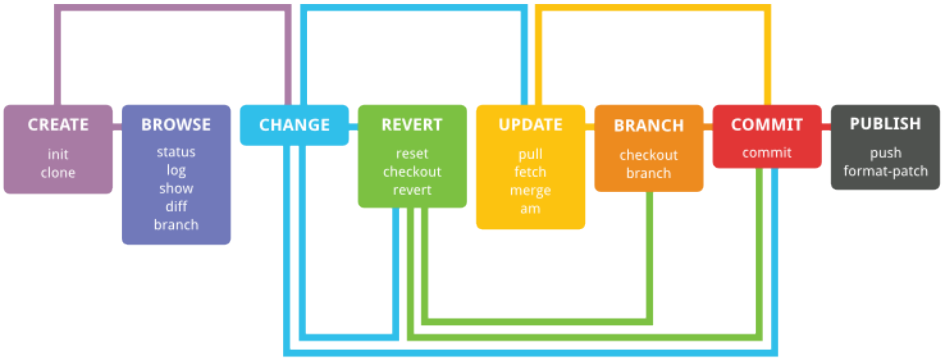
View merge conflicts against your changes
`git diff --ours <FILE>`

View merge conflicts against other changes
`git diff --theirs <FILE>`

Discard a conflicting patch
```
git reset --hard
git rebase --skip
```

After resolving conflicts, merge with
```
git add <CONFLICTING_FILE>
git rebase --continue
```

Workflow



Git – (mírně) pokročilé použití

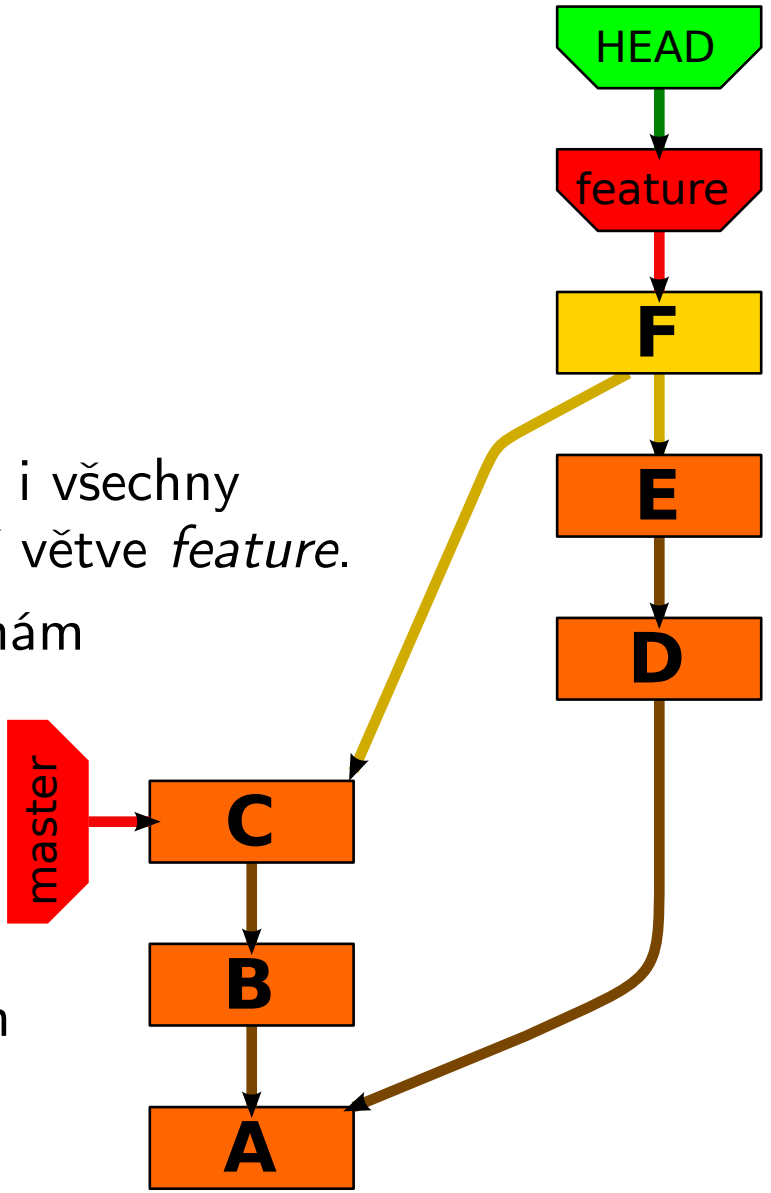
- Stále se jedná o základy, které je nutné ovládat
- **Merge**
  - Spojí dvě větve tak, že obě větve zůstanou nezměněné
  - Přidáním nového commitu
  - Po předání do sdíleného repository vidí větev všichni
  - Hodí se na větve vývoje větších funkcí a oprav
- **Rebase**
  - Spojí dvě větve tak, že změny jedné větve přidá na konec druhé
  - Jedna větev přestane existovat
  - Jsou vytvořeny nové commity v původní větvi
  - Fakticky není ztracen žádný commit
  - Hodí se pro malé lokální pracovní větve
  - *NEPOUŽÍVÁME na sdílených repozitářích!*

Git – merge

- Ve větvi „*feature*“ jsme provedli
  `git merge master`
- Vzniklo nové potvrzení „**F**“
- **F** obsahuje změny z potvrzení
  B, C, D, E
- **F** má dva předchůdce (E, C)
- Výsledkem je, že větev *feature* obsahuje i všechny úpravy provedené v *master* po vytvoření větve *feature*.
- Ve větvi master nedošlo k žádným změnám
- Posloupnost příkazů:

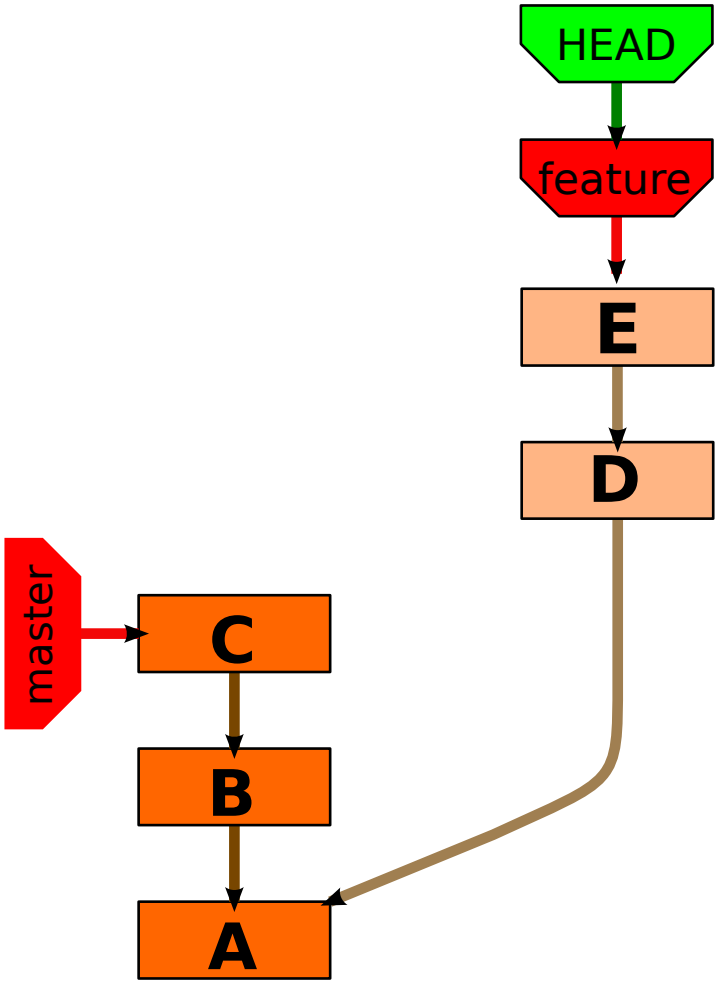
  ```
  git checkout feature
  git merge master
  ```

- Pokud nastanou konflikty, vyřešíme je a provedeme nové potvrzení konfliktních souborů



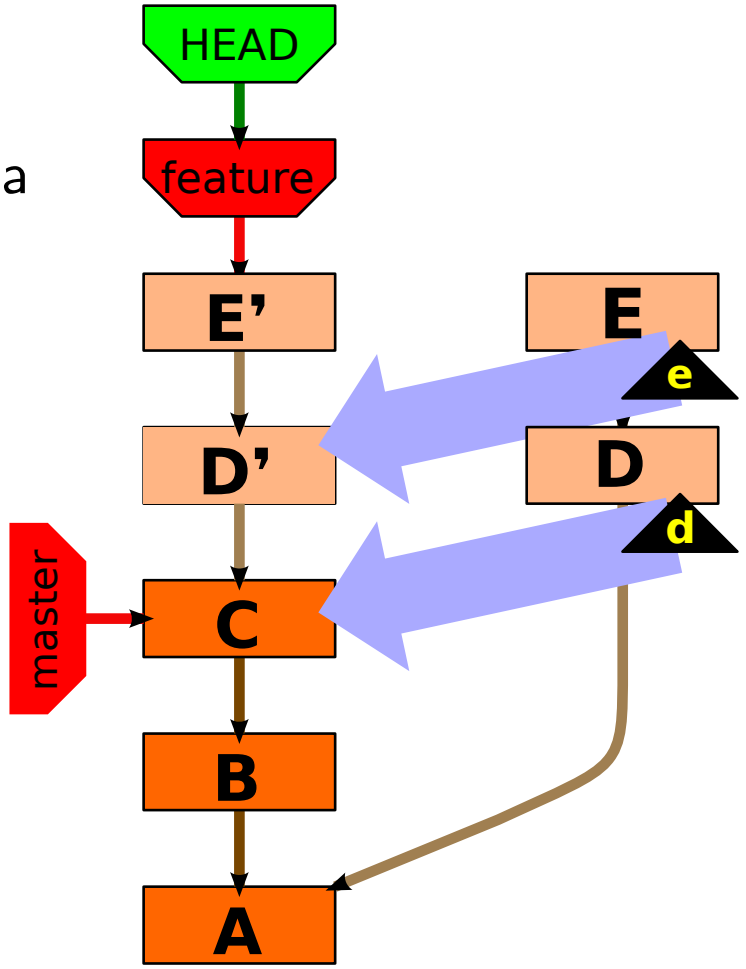
Git – rebase

- Ve větvi „*feature*“ provedeme
  `git rebase master`



Git – rebase

- Ve větvi „*feature*“ jsme provedli
  `git rebase master`
- Výsledkem je připojení změn z *feature* na konec *master*
- Původní potvrzení E a D bodou odstraněna
- E' a D' vznikly vytvořením rozdílů s větví *master* a jejich aplikováním na *master (postupně)*



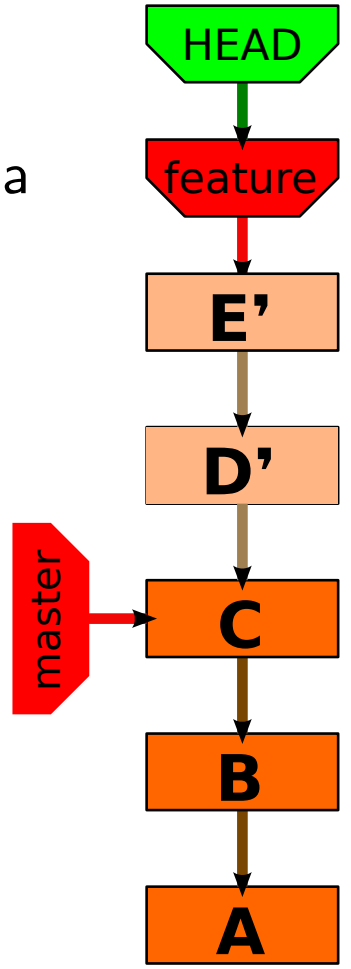
Git – rebase

- Ve větvi „*feature*“ jsme provedli
  `git rebase master`
- Výsledkem je připojení změn z *feature* na konec *master*
- Původní potvrzení E a D budou odstraněna
- E' a D' vznikly vytvořením rozdílů s větví *master* a jejich aplikováním na *master (postupně)*
- Pozor, větev *master* stále končí potvrzením C
- Konec *master* „posuneme nahoru“ například pomocí

  `git checkout master`

  `git rebase feature`
- Pokud během rebase nastane konflikt, můžeme po jeho vyřešení pokračovat

  `git rebase --continue`



Git – práce se vzdálenými repozitáři

- Git remote

```
git remote -v
git remote add <remote_alias> <URL>
git push <remote_alias> <branch>
git pull <remote_alias> <branch>
```

- Vzdálený repozitář
  - Může být zpřístupněn pomocí *místního přístupu*, *SSH*, *protokolu Git a HTTP*
  - Můžeme vytvořit publikováním lokálního repozitáře

```
git clone --bare ssh://user@server:/home/name/project.git
git remote add origin ssh://user@server:/home/name/project.git
```

  - Vytvoří vzdálenou kopii lokálního repozitáře a nastaví jej jako *origin*
  - Vzdálený repozitář je vytvořen bez pracovního stromu (--bare)
  - Nyní můžeme lokální změny publikovat na server příkazem

`git push` nebo `git push origin`

Git – hosting repozitářů

- Několik volných hostingů
- Obvykle zdarma jen pro nekomerční užití
- Hodí se pro hlavní repozitář projektu i jako osobní repozitář
- *http://git.or.cz/gitwiki/GitHosting*

repo.or.cz

Git – v praxi

Navrácení změn

- **Reset** – provede návrat na zadanou revizi (změna v index; --hard mění i prac. strom; bez udání commit vrátí změny provedené v prac. stromu)
  `git reset [--hard] [<commit>] | git reset <soubor>`
- **Revert** – pokusí se odstranit změny provedené zadanou revizí
  `git revert <commit>`
- Následně je nutné provést potvrzení změn do repozitáře

Otázka: Jak je identifikováno potvrzení (commit) v Gitu?

Zvláštní nastavení

- *Autocrlf* – automaticky ošetřit konce řádků na *nix like nastavíme v .git/config; core.autocrlf=true

Oddělená hlava (detached head)

- Aktuální poslední potvrzení není nejvyšší potvrzení větve
- Pokud se přepneme (checkout) do staršího potvrzení
- Napravíme `git checkout <branch>`
- Pokud jsme již omylem potvrdili nějaké změny
  `git checkout -b <new_branch>`
  - Tím jsme vytvořili novou větev a musíme provést *merge* nebo *rebase*

Git – v praxi

Inicializace nového repozitáře

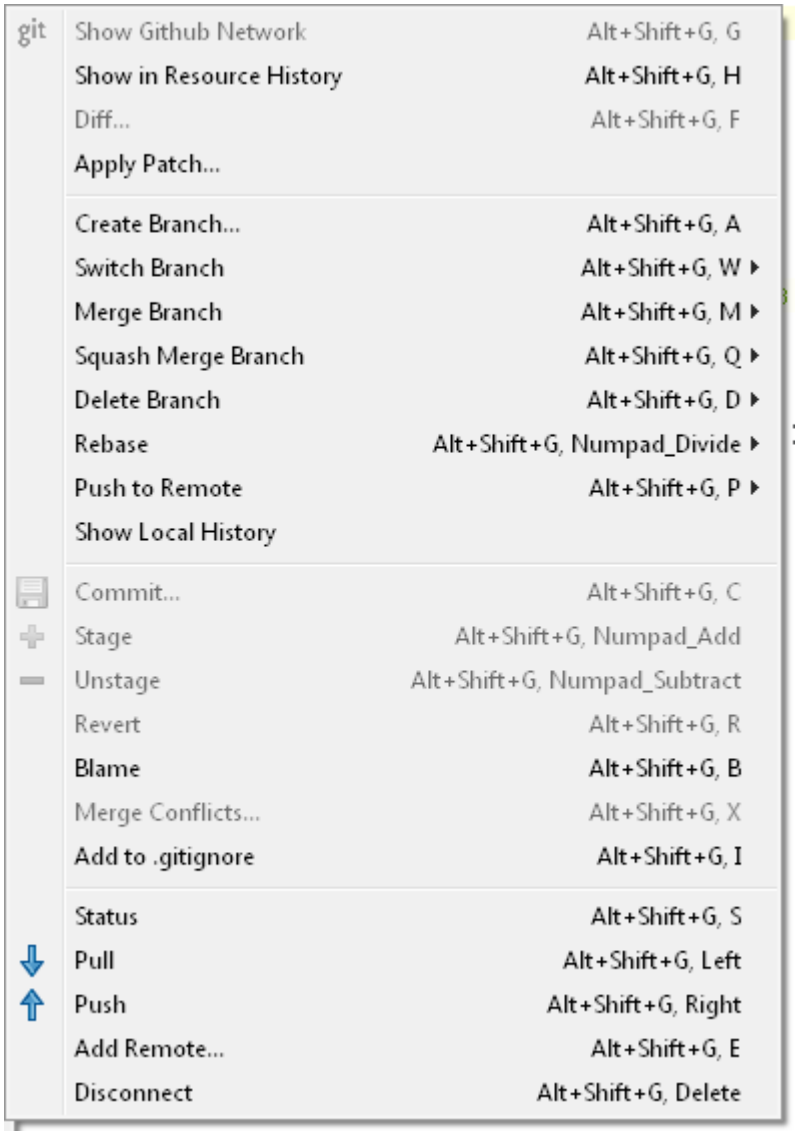
```
mkdir muj_projekt
cd muj_projekt
git init
echo test>foo.bar
git add foo.bar
git commit -m "první commit"

echo test>foo.baz
git commit -a -m "druhy commit"
```

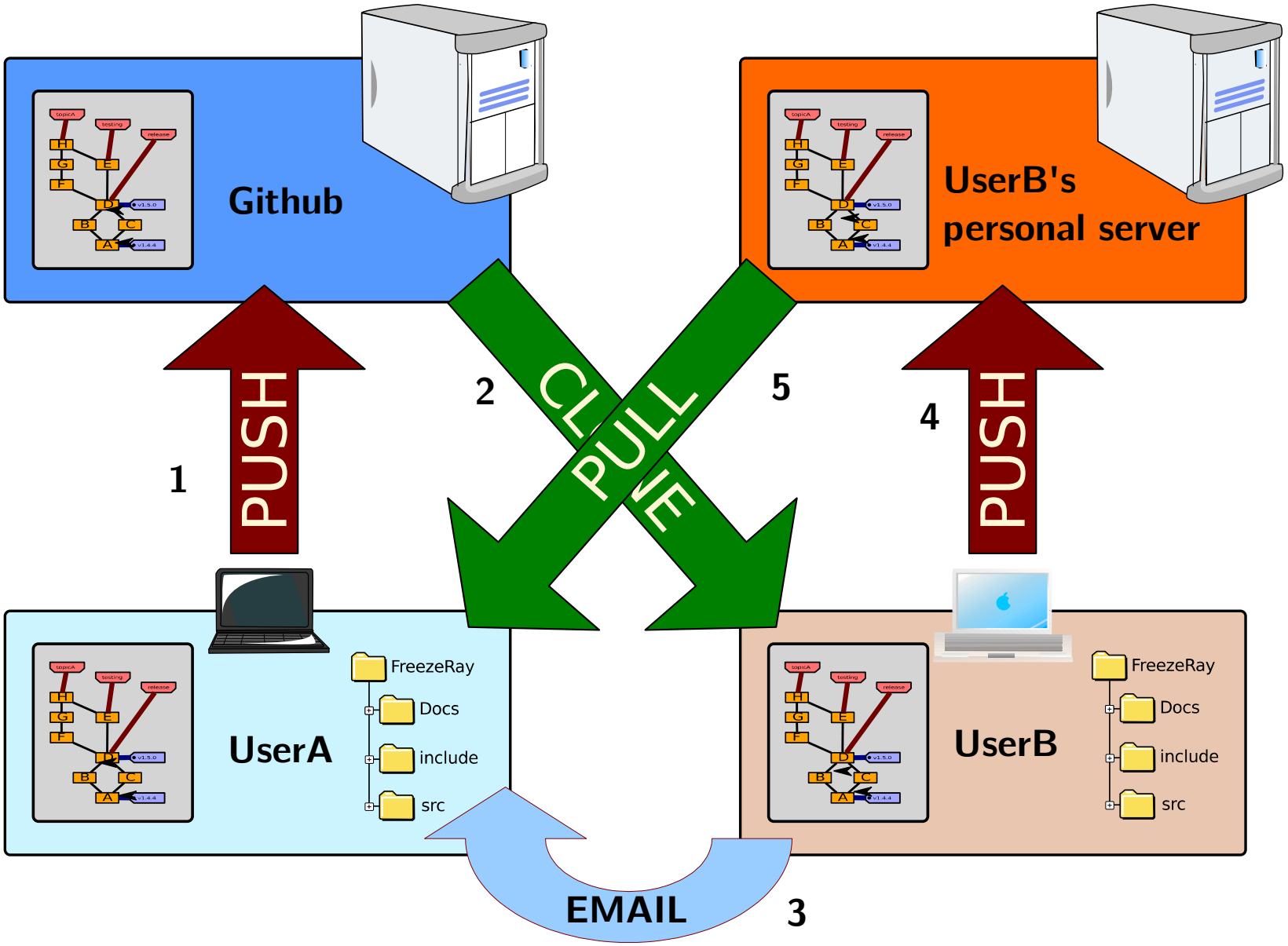
Další užitečné příkazy

- **git stash**
- **git fetch**
- **git show**
- **git blame**

Git je vhodné používat z IDE



Git – opakování



Git – zhodnocení

- Skvělý pro Open Source vývoj
- Snadná spolupráce roztříštěných skupin vývojářů
- Provádí operace bleskurychle
- Je možné neustále přepínat větve
- Perfektně sleduje závislosti mezi commity
- Na svém písečku si každý může bezpečně dělat opravdu cokoli
- Zaručuje konzistenci dat

Nevýhody

- Složitější na počáteční pochopení
- Vždy nakonec potřebujete i ty neběžné příkazy
- Repozitář musí mít každý uživatel úplně celý lokálně
- Problém s velkými a obřími repozitáři (řešeno jen částečně)
  - Linux Kernel a Android každý zatím pod 900 MB, ideální velikost repo. do 100 MB
- Revize nejsou číslovány v uspořádání (jen označovány SHA1 hašem)
- Cesta od vývojáře ke skutečnému sdílení je extrémně dlouhá

Otázky

?

Zdroje

- The Git Community Book (obsahuje vše a ještě víc)

  http://book.git-scm.com/
- Learn Git in 20 Minutes (video)

  https://www.youtube.com/watch?v=Y9XZQO1n_7c
- Git screencast (video)

  http://www.ralfebert.de/blog/tools/git_screencast/
- M. Schmidt, *Správa verzí*, přednáška IVS 2011

  http://pcivs.fit.vutbr.cz/06_schmidt_ivs-git_2011.pdf
- Steve Bennett, 10 things I hate about Git, 2012

  https://steveko.wordpress.com/2012/02/24/10-things-i-hate-about-git/