

Verzovací systémy

DELTA - Střední škola informatiky a ekonomie, s.r.o.

Ing. Luboš Zápotočný

14.09.2025

CC BY-NC-SA 4.0

Verzovací systémy

Motivace a přínosy verzování

Jaké vás napadají přínosy verzování?

Motivace a přínosy verzování

Jaké vás napadají přínosy verzování?

- Prokazatelná historie změn, audit a možnost návratu v čase

Motivace a přínosy verzování

Jaké vás napadají přínosy verzování?

- Prokazatelná historie změn, audit a možnost návratu v čase
- Paralelní práce více lidí, bezpečné experimenty ve větvích

Motivace a přínosy verzování

Jaké vás napadají přínosy verzování?

- Prokazatelná historie změn, audit a možnost návratu v čase
- Paralelní práce více lidí, bezpečné experimenty ve větvích
- Code review

Motivace a přínosy verzování

Jaké vás napadají přínosy verzování?

- Prokazatelná historie změn, audit a možnost návratu v čase
- Paralelní práce více lidí, bezpečné experimenty ve větvích
- Code review
- Bezpečnost a „zálohy“ díky vzdálenému repozitáři (remote)

Motivace a přínosy verzování

Jaké vás napadají přínosy verzování?

- Prokazatelná historie změn, audit a možnost návratu v čase
- Paralelní práce více lidí, bezpečné experimenty ve větvích
- Code review
- Bezpečnost a „zálohy“ díky vzdálenému repozitáři (remote)
- Traceability: kdo, co, kdy a proč něco změnil

Motivace a přínosy verzování

Jaké vás napadají přínosy verzování?

- Prokazatelná historie změn, audit a možnost návratu v čase
- Paralelní práce více lidí, bezpečné experimenty ve větvích
- Code review
- Bezpečnost a „zálohy“ díky vzdálenému repozitáři (remote)
- Traceability: kdo, co, kdy a proč něco změnil
- Automatizace: CI/CD, testy, buildy, releasy

Definice

Verzování je způsob správy změn v souborech a projektech v čase

Umožňuje uchovávat jednotlivé verze, porovnávat je, vracet změny a koordinovat práci více lidí

Definice

Verzovací systém (Version Control System - VCS) je softwarový nástroj pro správu verzí souborů

Definice

Verzovací systém (Version Control System - VCS) je softwarový nástroj pro správu verzí souborů

- Zaznamenává historii změn v souborech

Definice

Verzovací systém (Version Control System - VCS) je softwarový nástroj pro správu verzí souborů

- Zaznamenává historii změn v souborech
- Usnadňuje práci více lidí

Definice

Verzovací systém (Version Control System - VCS) je softwarový nástroj pro správu verzí souborů

- Zaznamenává historii změn v souborech
- Usnadňuje práci více lidí
- Podporuje větvení (**branching**) a slučování změn (**merge**)

Definice

Verzovací systém (Version Control System - VCS) je softwarový nástroj pro správu verzí souborů

- Zaznamenává historii změn v souborech
- Usnadňuje práci více lidí
- Podporuje větvení (**branching**) a slučování změn (**merge**)
- Umožňuje bezpečný návrat změn (**revert**)

Definice

Verzovací systém (Version Control System - VCS) je softwarový nástroj pro správu verzí souborů

- Zaznamenává historii změn v souborech
- Usnadňuje práci více lidí
- Podporuje větvení (**branching**) a slučování změn (**merge**)
- Umožňuje bezpečný návrat změn (**revert**)
- Umožňuje provádět **audit**: kdy, kdo, co a proč něco změnil

Definice

Verzovací systém (Version Control System - VCS) je softwarový nástroj pro správu verzí souborů

- Zaznamenává historii změn v souborech
- Usnadňuje práci více lidí
- Podporuje větvení (**branching**) a slučování změn (**merge**)
- Umožňuje bezpečný návrat změn (**revert**)
- Umožňuje provádět **audit**: kdy, kdo, co a proč něco změnil
- Chrání **integritu** historie pomocí kryptografických hashů a referencí

Typy verzovacích systémů

Jaké znáte typy verzovacích systémů?

Typy verzovacích systémů

Jaké znáte typy verzovacích systémů?

Centralizované

Typy verzovacích systémů

Jaké znáte typy verzovacích systémů?

Centralizované (SVN, Perforce)

Typy verzovacích systémů

Jaké znáte typy verzovacích systémů?

Centralizované (SVN, Perforce)

- Jeden centrální server, který spravuje aktuální stav

Typy verzovacích systémů

Jaké znáte typy verzovacích systémů?

Centralizované (SVN, Perforce)

- Jeden centrální server, který spravuje aktuální stav
 - **Single Point of Failure**

Typy verzovacích systémů

Jaké znáte typy verzovacích systémů?

Centralizované (SVN, Perforce)

- Jeden centrální server, který spravuje aktuální stav
 - **Single Point of Failure**
- Pracovat offline je téměř nemožné

Typy verzovacích systémů

Jaké znáte typy verzovacích systémů?

Centralizované (SVN, Perforce)

- Jeden centrální server, který spravuje aktuální stav
 - **Single Point of Failure**
- Pracovat offline je téměř nemožné

Distribuované

Typy verzovacích systémů

Jaké znáte typy verzovacích systémů?

Centralizované (SVN, Perforce)

- Jeden centrální server, který spravuje aktuální stav
 - **Single Point of Failure**
- Pracovat offline je téměř nemožné

Distribuované (**Git**, Mercurial)

Typy verzovacích systémů

Jaké znáte typy verzovacích systémů?

Centralizované (SVN, Perforce)

- Jeden centrální server, který spravuje aktuální stav
 - **Single Point of Failure**
- Pracovat offline je téměř nemožné

Distribučované (Git, Mercurial)

- Každý **klon** repozitáře je sám o sobě plnohodnotný

Typy verzovacích systémů

Jaké znáte typy verzovacích systémů?

Centralizované (SVN, Perforce)

- Jeden centrální server, který spravuje aktuální stav
 - **Single Point of Failure**
- Pracovat offline je téměř nemožné

Distribuované (Git, Mercurial)

- Každý **klon** repozitáře je sám o sobě plnohodnotný
- Lepší práce offline, **rychlost**, flexibilita větvení

Pojmy

- Repozitář:

Pojmy

- **Repozitář:** pracovní adresář + metadata v .git
- **Remote:**

Pojmy

- **Repozitář:** pracovní adresář + metadata v .git
- **Remote:** hlavní (vzdálený) repozitář
- **Commit:**

Pojmy

- **Repozitář:** pracovní adresář + metadata v .git
- **Remote:** hlavní (vzdálený) repozitář
- **Commit:** snapshot změn + autor, čas, zpráva, hash
- **Branch:**

Pojmy

- **Repozitář:** pracovní adresář + metadata v .git
- **Remote:** hlavní (vzdálený) repozitář
- **Commit:** snapshot změn + autor, čas, zpráva, hash
- **Branch:** ukazatel na commit (pohyblivý)
- **HEAD:**

Pojmy

- **Repozitář**: pracovní adresář + metadata v .git
- **Remote**: hlavní (vzdálený) repozitář
- **Commit**: snapshot změn + autor, čas, zpráva, hash
- **Branch**: ukazatel na commit (pohyblivý)
- **HEAD**: aktuální pozice v historii
- **Diff**:

Pojmy

- **Repozitář**: pracovní adresář + metadata v .git
- **Remote**: hlavní (vzdálený) repozitář
- **Commit**: snapshot změn + autor, čas, zpráva, hash
- **Branch**: ukazatel na commit (pohyblivý)
- **HEAD**: aktuální pozice v historii
- **Diff**: rozdíl mezi dvěma stavy
- **Patch**:

Pojmy

- **Repozitář**: pracovní adresář + metadata v .git
- **Remote**: hlavní (vzdálený) repozitář
- **Commit**: snapshot změn + autor, čas, zpráva, hash
- **Branch**: ukazatel na commit (pohyblivý)
- **HEAD**: aktuální pozice v historii
- **Diff**: rozdíl mezi dvěma stavy
- **Patch**: aplikované rozdíly mezi stavy

Základy práce s Gitem (lokálně)

Repozitář

Git repozitář je rozšířením projektového adresáře o speciální adresář `.git`, který obsahuje všechna metadata, konfiguraci a historii

Založení repozitáře

Repozitář

Git repozitář je rozšířením projektového adresáře o speciální adresář `.git`, který obsahuje všechna metadata, konfiguraci a historii

Založení repozitáře

- `git init` - založí nový (lokální) repozitář

Repozitář

Git repozitář je rozšířením projektového adresáře o speciální adresář `.git`, který obsahuje všechna metadata, konfiguraci a historii

Založení repozitáře

- `git init` - založí nový (lokální) repozitář
- `git clone <url>` - naklonuje existující (vzdálený) repozitář

Nastavení identity

Repozitář

Git repozitář je rozšířením projektového adresáře o speciální adresář `.git`, který obsahuje všechna metadata, konfiguraci a historii

Založení repozitáře

- `git init` - založí nový (lokální) repozitář
- `git clone <url>` - naklonuje existující (vzdálený) repozitář

Nastavení identity

- `git config [--global] user.name "Tvé Jméno"`
- `git config [--global] user.email "tvuj@email.cz"`

Stavy souborů v repozitáři

Soubor může být v několika stavech:

Stavy souborů v repozitáři

Soubor může být v několika stavech:

- **Untracked** - Git o souboru ještě nic neví

Stavy souborů v repozitáři

Soubor může být v několika stavech:

- **Untracked** - Git o souboru ještě nic neví
- **Tracked** - Git o souboru ví a sleduje ho

Stavy souborů v repozitáři

Soubor může být v několika stavech:

- **Untracked** - Git o souboru ještě nic neví
- **Tracked** - Git o souboru ví a sleduje ho
 - **Unmodified** - soubor se nezměnil od jeho posledního commitu

Stavy souborů v repozitáři

Soubor může být v několika stavech:

- **Untracked** - Git o souboru ještě nic neví
- **Tracked** - Git o souboru ví a sleduje ho
 - **Unmodified** - soubor se nezměnil od jeho posledního commitu
 - **Modified** - soubor se změnil vůči jeho poslednímu commitu

Stavy souborů v repozitáři

Soubor může být v několika stavech:

- **Untracked** - Git o souboru ještě nic neví
- **Tracked** - Git o souboru ví a sleduje ho
 - **Unmodified** - soubor se nezměnil od jeho posledního commitu
 - **Modified** - soubor se změnil vůči jeho poslednímu commitu
 - **Staged** - soubor je připraven k zabalení do commitu

Stavy souborů v repozitáři

Soubor může být v několika stavech:

- **Untracked** - Git o souboru ještě nic neví
- **Tracked** - Git o souboru ví a sleduje ho
 - **Unmodified** - soubor se nezměnil od jeho posledního commitu
 - **Modified** - soubor se změnil vůči jeho poslednímu commitu
 - **Staged** - soubor je připraven k zabalení do commitu

Stavy souborů v repozitáři

Untracked soubory můžete zobrazit například pomocí příkazu

Stavy souborů v repozitáři

Untracked soubory můžete zobrazit například pomocí příkazu `git status`

```
$ echo "hello" >hello.txt
```

```
$ git status
```

```
...
```

```
Untracked files:
```

```
(...)
```

```
hello.txt
```

Stavy souborů v repozitáři

Soubory můžete **začít sledovat** pomocí příkazu

Stavy souborů v repozitáři

Soubory můžete **začít sledovat** pomocí příkazu `git add [file/directory]`

```
$ git add hello.txt
```

Poté můžete **zobrazit stav** souborů pomocí příkazu `git status`

```
$ git status
```

```
...
```

```
Changes to be committed:
```

```
(...)
```

```
    new file:   hello.txt
```

Stavy souborů v repozitáři

Aktuální změny v souborech můžete zabalit do **commitu** pomocí příkazu

Stavy souborů v repozitáři

Aktuální změny v souborech můžete zabalit do **commitu** pomocí příkazu `git commit`

- `-m` označuje zprávu commitu (message) / popis změn

```
$ git commit -m "Add hello.txt"
[main (root-commit) b18730d] Add hello.txt
1 file changed, 1 insertion(+)
create mode 100644 hello.txt
```

Stavy souborů v repozitáři

Provedením změny se automaticky změní stav souboru na **modified**

```
$ echo "zmena" >hello.txt
```

```
$ git status
```

```
...
```

```
Changes not staged for commit:
```

```
  (...)
```

```
    modified:   hello.txt
```

Stavy souborů v repozitáři

Změněný obsah si můžete zobrazit pomocí příkazu

Stavy souborů v repozitáři

Změněný obsah si můžete zobrazit pomocí příkazu `git diff [file]`

- Pokud ne zadáte parametr `file`, zobrazí se rozdíly ve všech **sledovaných** souborech

```
$ git diff
--- a/hello.txt
+++ b/hello.txt
@@ -1,1 @@
-hello
+zmena
```


Stavy souborů v repozitáři

Tuto změnu musíme znovu označit a připravit pro zabalení do **commitu** pomocí příkazu

Stavy souborů v repozitáři

Tuto změnu musíme znovu označit a připravit pro zabalení do **commitu** pomocí příkazu `git add`

```
$ git add hello.txt
```

```
$ git status
```

```
...
```

```
Changes to be committed:
```

```
  (...)
```

```
    modified:   hello.txt
```

Stavy souborů v repozitáři

Změny připravené pro commit se **nezobrazí** příkazem `git diff`, ale je nutné použít příkaz

Stavy souborů v repozitáři

Změny připravené pro commit se **nezobrazí** příkazem `git diff`, ale je nutné použít příkaz `git diff --staged`

```
$ git diff --staged
```

```
...  
--- a/hello.txt  
+++ b/hello.txt  
@@ -1,1 @@  
-hello  
+zmena
```

Stavy souborů v repozitáři

Odpřipravit soubor můžeme pomocí příkazu

Stavy souborů v repozitáři

Odpřipravit soubor můžeme pomocí příkazu `git restore --staged [file]`

```
$ git restore --staged hello.txt
```

```
$ git diff --staged
```

```
<empty output>
```

```
$ git status
```

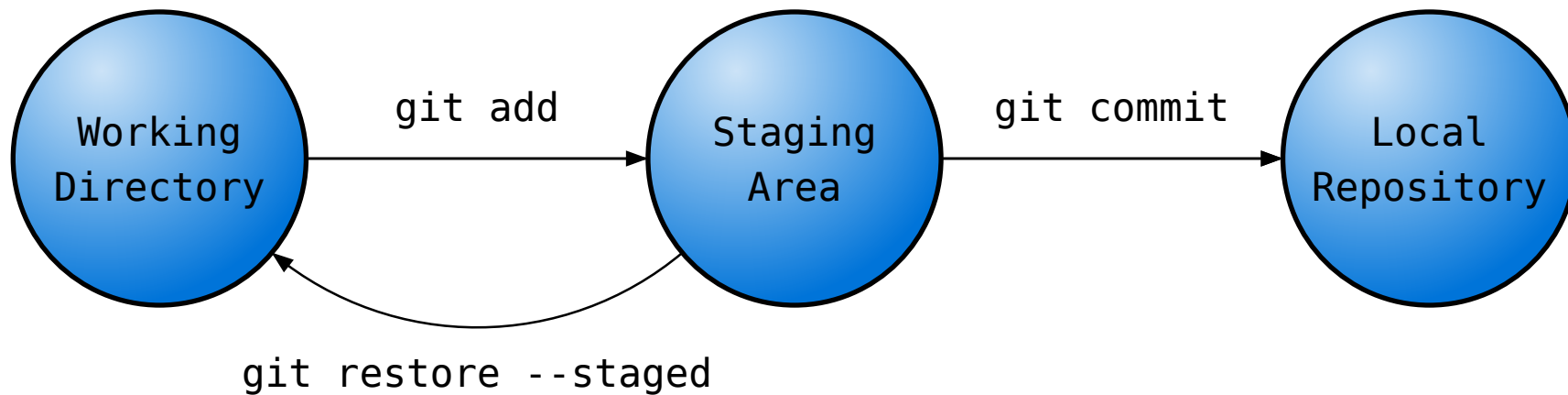
```
...
```

```
Changes to be committed:
```

```
  (...)
```

```
    modified:   hello.txt
```

Stavy souborů v repozitáři



Stavy souborů v repozitáři

Příprava základní historie

```
$ git add hello.txt  
$ git commit -m "Modified hello.txt"
```

```
$ echo "test" >test.txt  
$ git add test.txt  
$ git commit -m "Add test.txt"
```

```
$ echo "test2" >>test.txt  
$ git add test.txt  
$ git commit -m "Improve test.txt"
```


Ukázka historie

Historii změn lze zobrazit
příkazem

Ukázka historie

Historii změn lze zobrazit

příkazem `git log --oneline`

```
$ git log --oneline
```

```
3f02147 (HEAD -> main) Improve test.txt
```

```
f1112de Add test.txt
```

```
1494583 Modified hello.txt
```

```
b18730d Add hello.txt
```

Zde vidíme, že aktuální stav repozitáře, na který ukazuje ukazatel HEAD, je 3f02147 Improve test.txt

Ukázka historie

Detailní pohled na konkrétní commit lze zobrazit
příkazem

Ukázka historie

Detailní pohled na konkrétní commit lze zobrazit

příkazem `git show <hash>`

```
$ git show 3f02147
```

```
commit 3f02147110c48535b4687efa1945ef6a812d27fd (HEAD ->
main)
```

```
Author: Luboš Zápotočný <lubos.zapotocny@delta-skola.cz>
```

```
Date:    Wed Sep 3 16:17:22 2025 +0200
```

```
    Improve test.txt
```

```
...
```

Ukázka historie

```
....  
diff --git a/test.txt b/test.txt  
index 9daeafb..b02def2 100644  
--- a/test.txt  
+++ b/test.txt  
@@ -1,2 @@  
    test  
+test2
```

Ukázka historie

Příkaz `git show` lze použít i s parametrem `:<cesta>` k zobrazení obsahu konkrétního souboru v časovém bodě commitu, i když daný soubor nebyl přímo obsažen v daném commitu

```
$ git show f1112de:hello.txt  
zmena
```

Ignorování souborů

Soubory, které nechcete sledovat v repozitáři, můžete **ignorovat** pomocí souboru

Ignorování souborů

Soubory, které nechcete sledovat v repozitáři, můžete **ignorovat** pomocí souboru `.gitignore`

Připravíme si demo adresář s externími závislostmi

```
$ mkdir externals
$ for i in {1..10}; do mkdir externals/external-$i; done
$ for i in {1..10}; do echo "echo 'external-$i'" > externals/
external-$i/data.txt; done
```


Ignorování souborů

Git aktuálně nesleduje tyto soubory

Zjistíme to příkazem `git status`

```
$ git status
```

```
...
```

```
Untracked files:
```

```
(...)
```

```
externals/
```

Ignorování souborů

Pro detailní výpis nových souborů můžeme použít příkaz

Ignorování souborů

Pro detailní výpis nových souborů můžeme použít příkaz `git status --untracked-files`

```
$ git status --untracked-files
```

```
...
```

```
Untracked files:
```

```
(...)
```

```
externals/external-1/data.txt
```

```
externals/external-10/data.txt
```

```
externals/external-2/data.txt
```

```
...
```

Ignorování souborů

Přidání všech aktuálních změn v adresáři (rekurzivně) je možné pomocí příkazu

Ignorování souborů

Přidání všech aktuálních změn v adresáři (rekurzivně) je možné pomocí příkazu `git add .`

Je vhodné tento příkaz použít?

Ignorování souborů

Přidání všech aktuálních změn v adresáři (rekurzivně) je možné pomocí příkazu `git add .`

Je vhodné tento příkaz použít?

V tomto případě to je **nechtěné chování**, protože přidáváme do repozitáře i adresář `externals`, obsahující externí závislosti

Ty se ale **standardně nepřidávají do repozitáře** kvůli jejich velikosti (většinou jednotky GB)

Ignorování souborů

Pro automatické ignorování těchto adresářů a souborů je potřeba dané adresáře a soubory vyjmenovat v souboru `.gitignore`

```
$ echo "externals/" >> .gitignore
```

Git nyní bude adresář `externals` automaticky ignorovat

```
$ git status
```

```
...
```

```
Untracked files:
```

```
(...)
```

```
  .gitignore
```

Ignorování souborů

Soubor `.gitignore` by **měl být součástí zdrojového kódu repozitáře** a je tedy nutné ho commitnout

```
$ git add .gitignore  
$ git commit -m "Add .gitignore"
```

Proč to dělat?

Ignorování souborů

Soubor `.gitignore` by **měl být součástí zdrojového kódu repozitáře** a je tedy nutné ho commitnout

```
$ git add .gitignore  
$ git commit -m "Add .gitignore"
```

Proč to dělat?

Přidáním souboru `.gitignore` do repozitáře se všem spolupracovníkům nastaví stejná pravidla ignorování, což podporuje **jednotné workflow**

Ignorování souborů

Do souboru `.gitignore` se **typicky přidává**:

Ignorování souborů

Do souboru `.gitignore` se **typicky přidává**:

- `node_modules/` a `vendor/` - externí balíčky a knihovny

Ignorování souborů

Do souboru `.gitignore` se **typicky přidává**:

- `node_modules/` a `vendor/` - externí balíčky a knihovny
- `build/` - generované soubory (binární soubory, artifacty)

Ignorování souborů

Do souboru `.gitignore` se **typicky přidává**:

- `node_modules/` a `vendor/` - externí balíčky a knihovny
- `build/` - generované soubory (binární soubory, artifacty)
- `.env` - soubory s citlivými údaji (klíče, tokeny)

Ignorování souborů

Do souboru `.gitignore` se **typicky přidává**:

- `node_modules/` a `vendor/` - externí balíčky a knihovny
- `build/` - generované soubory (binární soubory, artifacty)
- `.env` - soubory s citlivými údaji (klíče, tokeny)
- `*.log` - logovací soubory

Ignorování souborů

Do souboru `.gitignore` se **typicky přidává**:

- `node_modules/` a `vendor/` - externí balíčky a knihovny
- `build/` - generované soubory (binární soubory, artifacty)
- `.env` - soubory s citlivými údaji (klíče, tokeny)
- `*.log` - logovací soubory
- `.idea/` - soubory s konfigurací IDE (např. JetBrains IDE)

Ignorování souborů

Do souboru `.gitignore` se **typicky přidává**:

- `node_modules/` a `vendor/` - externí balíčky a knihovny
- `build/` - generované soubory (binární soubory, artifacty)
- `.env` - soubory s citlivými údaji (klíče, tokeny)
- `*.log` - logovací soubory
- `.idea/` - soubory s konfigurací IDE (např. JetBrains IDE)
- `.DS_Store` - či jiné soubory, které se automaticky vytvářejí na určitých platformách (např. macOS)

Ignorování souborů

Do souboru `.gitignore` se **typicky přidává**:

- `node_modules/` a `vendor/` - externí balíčky a knihovny
- `build/` - generované soubory (binární soubory, artifacty)
- `.env` - soubory s citlivými údaji (klíče, tokeny)
- `*.log` - logovací soubory
- `.idea/` - soubory s konfigurací IDE (např. JetBrains IDE)
- `.DS_Store` - či jiné soubory, které se automaticky vytvářejí na určitých platformách (např. macOS)
- ...

Větvení

Větev v gitu je

Větvení

Větev v gitu je **pojmenovaný ukazatel** na commit

Například v následující historii vidíme pouze jednu větev `main`

```
$ git log --oneline
04e7a01 (HEAD -> main) Add .gitignore
3f02147 Improve test.txt
f1112de Add test.txt
1494583 Modified hello.txt
b18730d Add hello.txt
```

Větvení

Větev lze **vytvořit na pozici aktuálního commitu (HEAD)** pomocí příkazu

Větvení

Větev lze **vytvořit na pozici aktuálního commitu** (HEAD) pomocí příkazu `git branch <jméno>`

```
$ git branch new-branch
```

```
$ git log --oneline
```

```
04e7a01 (HEAD -> main, new-branch) Add .gitignore
```

```
3f02147 Improve test.txt
```

```
f1112de Add test.txt
```

```
1494583 Modified hello.txt
```

```
b18730d Add hello.txt
```

Větvení

Větev lze **vytvořit na konkrétním commitu** pomocí příkazu

Větvení

Větev lze **vytvořit na konkrétním commitu** pomocí příkazu `git branch <jméno> <hash>`

```
$ git branch second-branch 3f02147
```

```
$ git log --oneline
```

```
04e7a01 (HEAD -> main, new-branch) Add .gitignore
```

```
3f02147 (second-branch) Improve test.txt
```

```
f1112de Add test.txt
```

```
1494583 Modified hello.txt
```

```
b18730d Add hello.txt
```

Větvení

Aktuální větev lze **přepnout** pomocí příkazu

Větvení

Aktuální větev lze **přepnout** pomocí příkazu `git switch <jméno>`

```
$ git switch second-branch
```

```
Switched to branch 'second-branch'
```

```
$ git log --oneline
```

```
3f02147 (HEAD -> second-branch) Improve test.txt
```

```
f1112de Add test.txt
```

```
1494583 Modified hello.txt
```

```
b18730d Add hello.txt
```

Větvení

Aktuální větev lze **přepnout** pomocí příkazu `git switch <jméno>`

```
$ git switch second-branch
Switched to branch 'second-branch'
$ git log --oneline
3f02147 (HEAD -> second-branch) Improve test.txt
f1112de Add test.txt
1494583 Modified hello.txt
b18730d Add hello.txt
```

Pozor! - co se na výpisu logu změnilo a proč?

Větvení

Aktuální větev lze **přepnout** pomocí příkazu `git switch <jméno>`

```
$ git switch second-branch
Switched to branch 'second-branch'
$ git log --oneline
3f02147 (HEAD -> second-branch) Improve test.txt
f1112de Add test.txt
1494583 Modified hello.txt
b18730d Add hello.txt
```

Pozor! - co se na výpisu logu změnilo a proč?

Větvení

Výpis logu obsahuje historii **pouze aktuální větve** second-branch

V historii tedy například **nevidíme** commit Add .gitignore, který byl vytvořen později v historii a tato větev ho neobsahuje

Větvení

Aktuální větev lze vyčíst z výpisu příkazu

Větvení

Aktuální větev lze vyčíst z výpisu
příkazu `git branch` (označena hvězdičkou)

```
$ git branch  
  main  
  new-branch  
* second-branch
```

Nebo přímo pomocí příkazu

Větvení

Aktuální větev lze vyčíst z výpisu příkazu `git branch` (označena hvězdičkou)

```
$ git branch
  main
  new-branch
* second-branch
```

Nebo přímo pomocí příkazu `git branch --show-current`

```
$ git branch --show-current
second-branch
```

Větvení

Úpravy v jiné větvi **neovlivňují stav jiných větví**

Větvení

Úpravy v jiné větvi **neovlivňují stav jiných větví**

Přepnuli jsme se na větev `second-branch`, která je v historii ještě před přidáním souboru `.gitignore`, proto v následujících příkladech budeme přidávat nové soubory selektivně

Vytvoříme nový soubor a commitněme ho do aktuální větve

```
$ echo "branching" >branching.txt  
$ git add branching.txt  
$ git commit -m "Add branching.txt"
```

Větvení

Obsah aktuální větve (second-branch) je tedy následující:

```
$ git log --oneline
f9fbf77 (HEAD -> second-branch) Add branching.txt
3f02147 Improve test.txt
f1112de Add test.txt
1494583 Modified hello.txt
b18730d Add hello.txt
```

Větvení

Příkazu `git log` v základním nastavení zobrazuje pouze historii aktuální větve

Přidáme-li parametr `--branches`, zobrazí se historie všech větví

Větvení

Příkazu `git log` v základním nastavení zobrazuje pouze historii aktuální větve

Přidáme-li parametr `--branches`, zobrazí se historie všech větví

Výsledek ale není moc přehledný a hlavně nejsou vidět vztahy mezi větvemi, proto je vhodné také přidat parametr `--graph`

Větvení

```
$ git log --oneline --branches --graph
* f9fbf77 (HEAD -> second-branch) Add branching.txt
| * 04e7a01 (new-branch, main) Add .gitignore
|/
* 3f02147 Improve test.txt
* f1112de Add test.txt
* 1494583 Modified hello.txt
* b18730d Add hello.txt
```

Nový commit ve větvi second-branch vychází ze stejného commitu jako commit ve hlavní větvi, který přidával soubor .gitignore

Sloučení větví

Pokud jsme s výsledkem naší práce ve větvi second-branch spokojeni, můžeme se vrátit do hlavní větve main a **sloučit naše změny** pomocí příkazu

Sloučení větví

Pokud jsme s výsledkem naší práce ve větvi second-branch spokojeni, můžeme se vrátit do hlavní větve main a **sloučit naše změny** pomocí příkazu `git merge <jméno_větve_ke_sloučení>`

Sloučení větví

Pokud jsme s výsledkem naší práce ve větví second-branch spokojeni, můžeme se vrátit do hlavní větve main a **sloučit naše změny** pomocí příkazu `git merge <jméno_větve_ke_sloučení>`

Otevře se nám defaultní editor (vim nebo nano) s možností přidat popisek pro takzvaný **merge commit**

Editor stačí uložit a zavřít

- Pro vim -

Sloučení větví

Pokud jsme s výsledkem naší práce ve větví second-branch spokojeni, můžeme se vrátit do hlavní větve main a **sloučit naše změny** pomocí příkazu `git merge <jméno_větve_ke_sloučení>`

Otevře se nám defaultní editor (vim nebo nano) s možností přidat popisek pro takzvaný **merge commit**

Editor stačí uložit a zavřít

- Pro vim - `:wq`
- Pro nano -

Sloučení větví

Pokud jsme s výsledkem naší práce ve větvi `second-branch` spokojeni, můžeme se vrátit do hlavní větve `main` a **sloučit naše změny** pomocí příkazu `git merge <jméno_větve_ke_sloučení>`

Otevře se nám defaultní editor (`vim` nebo `nano`) s možností přidat popisek pro takzvaný **merge commit**

Editor stačí uložit a zavřít

- Pro `vim` - `:wq`
- Pro `nano` - `Ctrl+O` a `Ctrl+X`

Sloučení větví

```
$ git switch main  
Switched to branch 'main'  
$ git merge second-branch  
Merge made by the 'ort' strategy.  
  branching.txt | 1 +  
  1 file changed, 1 insertion(+)  
  create mode 100644 branching.txt
```

Sloučení větví

Zobrazení historie po sloučení větví vypadá následovně:

```
$ git log --oneline
43ed9a7 (HEAD -> main) Merge branch 'second-branch'
f9fbf77 (second-branch) Add branching.txt
04e7a01 (new-branch) Add .gitignore
3f02147 Improve test.txt
f1112de Add test.txt
1494583 Modified hello.txt
b18730d Add hello.txt
```

Sloučení větví

Důležitá pozorování:

Sloučení větví

Důležitá pozorování:

- Commit hashe se **nezměnily** - zajištění **integrity**

Sloučení větví

Důležitá pozorování:

- Commit hashe se **nezměnily** - zajištění **integrity**
- Větev second-branch **stále existuje**

Sloučení větví

Důležitá pozorování:

- Commit hashe se **nezměnily** - zajištění **integrity**
- Větev second-branch **stále existuje**
- Ve větvi main jsou nyní oba commity + jeden merge commit

Sloučení větví

Důležitá pozorování:

- Commit hashe se **nezměnily** - zajištění **integrity**
- Větev second-branch **stále existuje**
- Ve větvi main jsou nyní oba commity + jeden merge commit
- Větev new-branch je nyní **pozadu**

Sloučení větví

Ještě zajímavější je ale grafový pohled na historii

```
$ git log --oneline --graph
*    43ed9a7 (HEAD -> main) Merge branch 'second-branch'
|\
| * f9fbf77 (second-branch) Add branching.txt
* | 04e7a01 (new-branch) Add .gitignore
|/
* 3f02147 Improve test.txt
* f1112de Add test.txt
* 1494583 Modified hello.txt
* b18730d Add hello.txt
```

Sloučení větví

Zde vidíme, že speciální commit pro sloučení větví (merge commit) má dva rodičovské odkazy

Sloučení větví

Zde vidíme, že speciální commit pro sloučení větví (merge commit) má dva rodičovské odkazy

- Jeden na commit z původní větve `main` - `04e7a01`

Sloučení větví

Zde vidíme, že speciální commit pro sloučení větví (merge commit) má dva rodičovské odkazy

- Jeden na commit z původní větve `main` - `04e7a01`
- Druhý na commit z větve `second-branch` - `f9fbf77`

Sloučení větví

Zde vidíme, že speciální commit pro sloučení větví (merge commit) má dva rodičovské odkazy

- Jeden na commit z původní větve `main` - `04e7a01`
- Druhý na commit z větve `second-branch` - `f9fbf77`

Sloučení větví

Metoda slučování větví s merge commitem (**merge**) je jedním ze dvou hlavních způsobů, jak sloučit změny ze dvou větví

Druhým způsobem je **rebase**, pro který si teď připravíme data

Sloučení větví

```
$ git switch new-branch  
Switched to branch 'new-branch'  
$ echo "rebase" >rebase.txt  
$ git add rebase.txt  
$ git commit -m "Add rebase.txt"  
[new-branch 9106e41] Add rebase.txt  
1 file changed, 1 insertion(+)  
create mode 100644 rebase.txt
```


Sloučení větví

Přepnuli jsme se na větev new-branch, která je **pozadu** vůči větvi main a vytvořili jsme nový soubor rebase.txt, který jsme commitnuli

Pojďme se podívat na grafový pohled na historii

Sloučení větví

```
$ git log --oneline --branches --graph
* 9106e41 (HEAD -> new-branch) Add rebase.txt
| * 43ed9a7 (main) Merge branch 'second-branch'
|/|
| * f9fbf77 (second-branch) Add branching.txt
* | 04e7a01 Add .gitignore
|/
* 3f02147 Improve test.txt
* f1112de Add test.txt
* 1494583 Modified hello.txt
* b18730d Add hello.txt
```

Sloučení větví

Začlenění změny z větve new-branch do větve main lze provést slučováním stejně jako v předchozím příkladu, ale teď předvedeme metodu pomocí příkazu **rebase**

```
$ git branch --show-current  
new-branch
```

```
$ git rebase main
```

```
Successfully rebased and updated refs/heads/new-branch.
```

Zkontrolujeme grafový pohled na historii

Sloučení větví

```
$ git log --oneline --branches --graph
* 7cb77bd (HEAD -> new-branch) Add rebase.txt
*    43ed9a7 (main) Merge branch 'second-branch'
|\
| * f9fbf77 (second-branch) Add branching.txt
* | 04e7a01 Add .gitignore
|/
* 3f02147 Improve test.txt
* f1112de Add test.txt
* 1494583 Modified hello.txt
* b18730d Add hello.txt
```

Sloučení větví

Co se změnilo?

Sloučení větví

Co se změnilo?

- Commit Add rebase.txt nyní **navazuje na jiný commit**
 - Původně navazoval na commit 3f02147 - Improve test.txt

Sloučení větví

Co se změnilo?

- Commit Add rebase.txt nyní **navazuje na jiný commit**
 - Původně navazoval na commit 3f02147 - Improve test.txt
 - Nyní navazuje na commit 43ed9a7 - Merge branch ...

Sloučení větví

Co se změnilo?

- Commit Add rebase.txt nyní **navazuje na jiný commit**
 - Původně navazoval na commit 3f02147 - Improve test.txt
 - Nyní navazuje na commit 43ed9a7 - Merge branch ...
- Větev new-branch je nyní **dopředu** vůči větvi main

Sloučení větví

Co se změnilo?

- Commit Add rebase.txt nyní **navazuje na jiný commit**
 - Původně navazoval na commit 3f02147 - Improve test.txt
 - Nyní navazuje na commit 43ed9a7 - Merge branch ...
- Větev new-branch je nyní **dopředu** vůči větvi main
 - Větev main je nyní **pozadu** vůči větvi new-branch

Sloučení větví

Co se změnilo?

- Commit `Add rebase.txt` nyní **navazuje na jiný commit**
 - Původně navazoval na commit `3f02147 - Improve test.txt`
 - Nyní navazuje na commit `43ed9a7 - Merge branch ...`
- Větev `new-branch` je nyní **dopředu** vůči větvi `main`
 - Větev `main` je nyní **pozadu** vůči větvi `new-branch`
- Commit hash commitu `Add rebase.txt` se **změnil**

Sloučení větví

Co se změnilo?

- Commit `Add rebase.txt` nyní **navazuje na jiný commit**
 - Původně navazoval na commit `3f02147 - Improve test.txt`
 - Nyní navazuje na commit `43ed9a7 - Merge branch ...`
- Větev `new-branch` je nyní **dopředu** vůči větvi `main`
 - Větev `main` je nyní **pozadu** vůči větvi `new-branch`
- Commit hash commitu `Add rebase.txt` se **změnil**
 - Z původního `9106e41` na nový `7cb77bd`

Sloučení větví

Co se změnilo?

- Commit `Add rebase.txt` nyní **navazuje na jiný commit**
 - Původně navazoval na commit `3f02147 - Improve test.txt`
 - Nyní navazuje na commit `43ed9a7 - Merge branch ...`
- Větev `new-branch` je nyní **dopředu** vůči větvi `main`
 - Větev `main` je nyní **pozadu** vůči větvi `new-branch`
- Commit hash commitu `Add rebase.txt` se **změnil**
 - Z původního `9106e41` na nový `7cb77bd`

Sloučení větví

Proč se změnil commit hash?

Sloučení větví

Proč se změnil commit hash?

Protože commit hash je **unikátní identifikátor** commitu a součástí commitu je také odkaz na rodičovský commit

Sloučení větví

Proč se změnil commit hash?

Protože commit hash je **unikátní identifikátor** commitu a součástí commitu je také odkaz na rodičovský commit

Pokud se tedy **změní obsah commitu**, v tomto případě odkaz na rodičovský commit, musí se to **projevit i na jeho hashi**

Sloučení větví

No, a proč toto dělat?

Sloučení větví

No, a proč toto dělat?

Pamatujete, co jsou větve?

Sloučení větví

No, a proč toto dělat?

Pamatujete, co jsou větve?

Větve jsou **pojmenované ukazatele** na commit

Když jsme zajistili, že new-branch je **dopředu** vůči main, tak můžeme pouze

Sloučení větví

No, a proč toto dělat?

Pamatujete, co jsou větve?

Větve jsou **pojmenované ukazatele** na commit

Když jsme zajistili, že new-branch je **dopředu** vůči main, tak můžeme pouze **přesunout ukazatel** pro main na tento nový commit a bude hotovo

Jinými slovy, změna z větve new-branch bude začleněna i do větve main **bez nutnosti vytvoření merge commitu**

Sloučení větví

```
$ git switch main
Switched to branch 'main'
$ git merge new-branch
Updating 43ed9a7..7cb77bd
Fast-forward
  rebase.txt | 1 +
  1 file changed, 1 insertion(+)
  create mode 100644 rebase.txt
```

Co se změnilo, oproti ne-rebasnuté verzi?

Sloučení větví

Fast-forward merge mód je přesně to, o co jsme se snažili

Ve FF módu se pouze **posouvají ukazatele**

Nevytváří se nové merge commity

Zkontrolujeme grafový pohled na historii

Sloučení větví

```
$ git log --oneline --branches --graph
* 7cb77bd (HEAD -> main, new-branch) Add rebase.txt
*    43ed9a7 Merge branch 'second-branch'
|\
| * f9fbf77 (second-branch) Add branching.txt
* | 04e7a01 Add .gitignore
|/
* 3f02147 Improve test.txt
* f1112de Add test.txt
* 1494583 Modified hello.txt
* b18730d Add hello.txt
```

Sloučení větví

Co se změnilo, oproti ne-rebasnuté verzi?

Sloučení větví

Co se změnilo, oproti ne-rebasnuté verzi?

- Větve `main` a `new-branch` ukazují na stejný commit - `7cb77bd`

Sloučení větví

Co se změnilo, oproti ne-rebasnuté verzi?

- Větve `main` a `new-branch` ukazují na stejný commit - `7cb77bd`
- Nemáme „zbytečný“ merge commit v historii

Konflikty

Merge conflict je

Konflikty

Merge conflict je situace, kdy se změny ze vzdáleného repozitáře nebo jiné větve liší natolik, že Git neví, jak je korektně sloučit

Konflikty

Merge conflict je situace, kdy se změny ze vzdáleného repozitáře nebo jiné větve liší natolik, že Git neví, jak je korektně sloučit

Vytvoříme situaci, která povede k merge conflictu

Konflikty

```
$ git branch --show-current
```

```
main
```

```
$ echo "content from main" >conflict.txt
```

```
$ git add conflict.txt
```

```
$ git commit -m "Add conflict.txt from main"
```

```
$ git switch second-branch
```

```
$ echo "content from second-branch" >conflict.txt
```

```
$ git add conflict.txt
```

```
$ git commit -m "Add conflict.txt from second-branch"
```

```
$ git switch main
```

Konflikty

```
* a5497c5 (second-branch) Add conflict.txt from second-branch
| * e2186ba (HEAD -> main) Add conflict.txt from main
| * 7cb77bd (new-branch) Add rebase.txt
| * 43ed9a7 Merge branch 'second-branch'
| |\
| |/
|/|
* | f9fbf77 Add branching.txt
| * 04e7a01 Add .gitignore
|/
* 3f02147 Improve test.txt
```

Konflikty

Nyní se pokusíme sloučit změny z větve

Konflikty

Nyní se pokusíme sloučit změny z větve `second-branch` do větve

Konflikty

Nyní se pokusíme sloučit změny z větve `second-branch` do větve `main` pomocí příkazu

Konflikty

Nyní se pokusíme sloučit změny z větve `second-branch` do větve `main` pomocí příkazu `git merge second-branch`

```
$ git merge second-branch
```

```
Auto-merging conflict.txt
```

```
CONFLICT (add/add): Merge conflict in conflict.txt
```

```
Automatic merge failed; fix conflicts and then commit the  
result.
```

Konflikty

Git nám zhlásil konflikt a řekl nám, že musíme konflikt v souboru `conflict.txt` **vyřešit manuálně**

Konflikty se řeší **přímo v obsahu** daného souboru

```
$ cat conflict.txt
<<<<<< HEAD
content from main
=====
content from second-branch
>>>>>> second-branch
```

Konflikty

Pro vyřešení je nutné pouze otevřít soubor v oblíbeném editoru a upravit obsah tak, aby vyhovoval požadavkům z obou větví

Speciální znaky, pomocí kterých nám Git označil, kde se nachází konflikt je nutné před pokračováním odstranit

Konflikty

Pro vyřešení je nutné pouze otevřít soubor v oblíbeném editoru a upravit obsah tak, aby vyhovoval požadavkům z obou větví

Speciální znaky, pomocí kterých nám Git označil, kde se nachází konflikt je nutné před pokračováním odstranit

```
$ cat conflict.txt  
content from main  
content from second-branch
```

Konflikty

Tato změna bude nyní vidět jako změněný soubor

```
$ git status
```

```
...
```

```
Unmerged paths:
```

```
(...)
```

```
    both added:    conflict.txt
```

A tedy stejně jako při jiných úpravách je potřeba tento soubor **připravit** pro commit pomocí příkazu

Konflikty

Tato změna bude nyní vidět jako změněný soubor

```
$ git status
```

```
...
```

```
Unmerged paths:
```

```
(...)
```

```
    both added:    conflict.txt
```

A tedy stejně jako při jiných úpravách je potřeba tento soubor **připravit** pro commit pomocí příkazu `git add`

```
$ git add conflict.txt
```

Konflikty

A tuto úpravu následně **commitnout**

```
$ git commit -m "Resolve conflict in conflict.txt"  
[main 79203b8] Resolve conflict in conflict.txt
```

Zkontrolujeme grafový pohled na historii

Konflikty

```
*    79203b8 (HEAD -> main) Resolve conflict in conflict.txt
|\
| * a5497c5 (second-branch) Add conflict.txt from second-
branch
* | e2186ba Add conflict.txt from main
* | 7cb77bd (new-branch) Add rebase.txt
* | 43ed9a7 Merge branch 'second-branch'
|\|
| * f9fbf77 Add branching.txt
* | 04e7a01 Add .gitignore
|/
```

Konflikty

Obsah tohoto speciálního commitu je následující:

```
$ git show 79203b8
...
Merge: e2186ba a5497c5
...
--- a/conflict.txt
+++ b/conflict.txt
@@@ -1,1 -1,1 +1,2 @@@
+content from main
+ content from second-branch
```

Konflikty

Zkontrolujeme aktuální stav souboru `conflict.txt`

```
$ cat conflict.txt  
content from main  
content from second-branch
```

A máme **hotovo!**

Pomocné příkazy

Kolaborativní vývoj více lidí na jedné větvi může vést k problematickým situacím

Jedním z nejdůležitějších principů je

Pomocné příkazy

Kolaborativní vývoj více lidí na jedné větvi může vést k problematickým situacím

Jedním z nejdůležitějších principů je
nepřepisovat historii ve sdílených větvích

Přepisování historie si možná ukážeme v budoucnosti, ale ti, kteří se o to zajímají mohou prostudovat příkazy `git rebase -i` a `git push --force`

Pomocné příkazy

My si ale ještě ukážeme příkaz na vrácení změn z commitu

Pomocné příkazy

My si ale ještě ukážeme příkaz na vrácení změn z commitu
`git revert <commithash>`, který pomáhá při vývoji na sdílených
větvích a hlavně při opravě změn na hlavní větvi

Pomocné příkazy

My si ale ještě ukážeme příkaz na vrácení změn z commitu `git revert <commithash>`, který pomáhá při vývoji na sdílených větvích a hlavně při opravě změn na hlavní větvi

Použijeme ho na commit `f9fbf77 Add branching.txt`, který jsme vytvořili ve větvi `second-branch` a který přidával soubor `branching.txt`, který jsme poté začlenili do větve `main`

Pomocné příkazy

My si ale ještě ukážeme příkaz na vrácení změn z commitu `git revert <commithash>`, který pomáhá při vývoji na sdílených větvích a hlavně při opravě změn na hlavní větvi

Použijeme ho na commit `f9fbf77 Add branching.txt`, který jsme vytvořili ve větvi `second-branch` a který přidával soubor `branching.txt`, který jsme poté začlenili do větve `main`

Otevře se nám defaultní editor (`vim` nebo `nano`) s možností přidat popisek pro takzvaný **revert commit**

Pomocné příkazy

```
$ git revert f9fbf77
```

```
$ git log --oneline --branches --graph
```

```
* 3eb526c (HEAD -> main) Revert "Add branching.txt"
```

```
*    79203b8 Resolve conflict in conflict.txt
```

```
|\
```

```
| * a5497c5 (second-branch) Add conflict.txt from second-  
branch
```

```
* | e2186ba Add conflict.txt from main
```

Pomocné příkazy

```
$ git show 3eb526c
```

```
    This reverts commit
```

```
f9fbf7707afe58ca5cf2970ee52085fd2cb4bcae.
```

```
diff --git a/branching.txt b/branching.txt
```

```
deleted file mode 100644
```

```
index 0742518..0000000
```

```
--- a/branching.txt
```

```
+++ /dev/null
```

```
@@ -1,0 @@
```

```
-branching
```

Pomocné příkazy

Ve volných chvílích se také podívejte na příkaz `git mergetool`, který vám pomůže podobné konflikty řešit

Naučte se také ve vašem oblíbeném IDE používat Git integraci, která vám pomůže mimo jiné i s řešením konfliktů

Často přínosným příkazem je také `git stash`, který vám pomůže uchovat změny, které nechcete commitnout, ale nechcete je také ztratit

Dalším důležitým a lehce nebezpečným příkazem je `git reset`, který vám pomůže vrátit se na určitý commit v historii

Práce se vzdáleným repozitářem

Klonování

Pokud je lokální repozitář inicializovaný **klonováním** vzdáleného repozitáře (**remote**), tak Git automaticky vytvoří referenci na daný vzdálený repozitář

Příkaz pro klonování je

Klonování

Pokud je lokální repozitář inicializovaný **klonováním** vzdáleného repozitáře (**remote**), tak Git automaticky vytvoří referenci na daný vzdálený repozitář

Příkaz pro klonování je `git clone <adresa>`

Klonování je možné provést pomocí protokolů

Klonování

Pokud je lokální repozitář inicializovaný **klonováním** vzdáleného repozitáře (**remote**), tak Git automaticky vytvoří referenci na daný vzdálený repozitář

Příkaz pro klonování je `git clone <adresa>`

Klonování je možné provést pomocí protokolů

- HTTPS
- **SSH** (nejlepší)
- Git (málo podporovaný)

Klonování

Jaký je rozdíl v práci se vzdáleným repozitářem pomocí **HTTPS** a **SSH**?

Klonování

Jaký je rozdíl v práci se vzdáleným repozitářem pomocí **HTTPS** a **SSH**?

HTTP protokol je **bezstavový**, zatímco SSH je stavový protokol

Klonování

Jaký je rozdíl v práci se vzdáleným repozitářem pomocí **HTTPS** a **SSH**?

HTTP protokol je **bezstavový**, zatímco SSH je stavový protokol

Při **klonování** nebo **synchronizaci** pomocí HTTPS je potřeba **vždy** zadat přístupové údaje (neplatí pro veřejné repozitáře), ale při SSH je potřeba zadat **pouze jednou** do vzdáleného systému svůj veřejný SSH klíč

Klonování

Jaký je rozdíl v práci se vzdáleným repozitářem pomocí **HTTPS** a **SSH**?

HTTP protokol je **bezstavový**, zatímco SSH je stavový protokol

Při **klonování** nebo **synchronizaci** pomocí HTTPS je potřeba **vždy** zadat přístupové údaje (neplatí pro veřejné repozitáře), ale při SSH je potřeba zadat **pouze jednou** do vzdáleného systému svůj veřejný SSH klíč

Při **nahrávání změn** pomocí HTTPS je potřeba **vždy** zadat přístupové údaje, ale při SSH je autentizace **automatická pomocí SSH klíče**

Klonování

```
$ git clone https://github.com/delta-cs/vyuka.git
Cloning into 'vyuka'...
remote: Enumerating objects: 25, done.
remote: Counting objects: 100% (25/25), done.
remote: Compressing objects: 100% (13/13), done.
remote: Total 25 (delta 5), reused 23 (delta 3), pack-reused
0 (from 0)
Receiving objects: 100% (25/25), done.
Resolving deltas: 100% (5/5), done.

$ cd vyuka
```

Klonování

Seznam vzdálených repozitářů lze zobrazit příkazem

Klonování

Seznam vzdálených repozitářů lze zobrazit příkazem `git remote -v`

```
$ git remote -v  
origin  https://github.com/delta-cs/vyuka.git (fetch)  
origin  https://github.com/delta-cs/vyuka.git (push)
```

Výstup příkazu potvrzuje, že v adresáři vyuka je lokální repozitář, který má referenci na vzdálený repozitář pojmenovaný origin

Klonování

Seznam vzdálených repozitářů lze zobrazit příkazem `git remote -v`

```
$ git remote -v  
origin  https://github.com/delta-cs/vyuka.git (fetch)  
origin  https://github.com/delta-cs/vyuka.git (push)
```

Výstup příkazu potvrzuje, že v adresáři vyuka je lokální repozitář, který má referenci na vzdálený repozitář pojmenovaný origin

Fetch a **push** jsou dva základní příkazy pro

Klonování

Seznam vzdálených repozitářů lze zobrazit příkazem `git remote -v`

```
$ git remote -v  
origin  https://github.com/delta-cs/vyuka.git (fetch)  
origin  https://github.com/delta-cs/vyuka.git (push)
```

Výstup příkazu potvrzuje, že v adresáři `vyuka` je lokální repozitář, který má referenci na vzdálený repozitář pojmenovaný `origin`

Fetch a **push** jsou dva základní příkazy pro synchronizaci mezi lokálním a vzdáleným repozitářem, které si probereme později

Přidání vzdáleného repozitáře

Manuální přidání vzdáleného repozitáře lze provést příkazem

Přidání vzdáleného repozitáře

Manuální přidání vzdáleného repozitáře lze provést příkazem `git remote add <jméno> <adresa>`

Jméno hlavního vzdáleného repozitáře je obvykle `origin`, nicméně lze ho pojmenovat libovolně

```
$ git remote add origin https://github.com/delta-cs/vyuka.git
```

Synchronizace

Stažení změn ze vzdáleného repozitáře lze provést příkazem

Synchronizace

Stažení změn ze vzdáleného repozitáře lze provést příkazem `git fetch [<jméno>]`

```
$ git fetch
```

Synchronizace

Stažení změn ze vzdáleného repozitáře lze provést příkazem `git fetch [<jméno>]`

```
$ git fetch
```

Tento příkaz stáhne historii ze vzdáleného repozitáře (defaultně stahuje z origin), ale **pouze do lokálního repozitáře** - adresáře `.git`

Změny se ještě neprojeví ve working directory

Pro začlenění změn ze vzdáleného repozitáře do lokálního repozitáře je potřeba použít příkaz

Synchronizace

Stažení změn ze vzdáleného repozitáře lze provést příkazem `git fetch [<jméno>]`

```
$ git fetch
```

Tento příkaz stáhne historii ze vzdáleného repozitáře (defaultně stahuje z origin), ale **pouze do lokálního repozitáře** - adresáře `.git`

Změny se ještě neprojeví ve working directory

Pro začlenění změn ze vzdáleného repozitáře do lokálního repozitáře je potřeba použít příkaz `git merge <remote>/<branch>`

Synchronizace

```
$ git merge origin/master
```

Tento příkaz začlení změny ze vzdáleného repozitáře (`origin`) a jeho větve `master` do lokálního repozitáře a aktuální větve

Synchronizace

Příkaz `git pull` je zkratka pro `git fetch` a `git merge`

Synchronizace

Lokální větev s novými commity můžete nahrát na vzdálený repozitář pomocí příkazu

Synchronizace

Lokální větev s novými commity můžete nahrát na vzdálený repozitář pomocí příkazu `git push <remote> <branch>`

Pro ulehčení lze použít příkaz

Synchronizace

Lokální větev s novými commity můžete nahrát na vzdálený repozitář pomocí příkazu `git push <remote> <branch>`

Pro ulehčení lze použít příkaz `git push --set-upstream <remote> <branch>`, který nastaví vzdálený repozitář a větev automaticky

Synchronizace

Lokální větev s novými commity můžete nahrát na vzdálený repozitář pomocí příkazu `git push <remote> <branch>`

Pro ulehčení lze použít příkaz `git push --set-upstream <remote> <branch>`, který nastaví vzdálený repozitář a větev automaticky

Poté je možné používat pouze příkaz

Synchronizace

Lokální větev s novými commity můžete nahrát na vzdálený repozitář pomocí příkazu `git push <remote> <branch>`

Pro ulehčení lze použít příkaz `git push --set-upstream <remote> <branch>`, který nastaví vzdálený repozitář a větev automaticky

Poté je možné používat pouze příkaz `git push`

Best practices

Zprávy commitů a konvence

Best practices

Zprávy commitů a konvence

- **Imperativní styl:** „Add login validation“ (ne „Added“ / „Adds“)

Best practices

Zprávy commitů a konvence

- **Imperativní styl:** „Add login validation“ (ne „Added“ / „Adds“)
- **Stručný předmět** (≤ 50 znaků), prázdný řádek, případně delší popis

Best practices

Zprávy commitů a konvence

- **Imperativní styl:** „Add login validation“ (ne „Added“ / „Adds“)
- **Stručný předmět** (≤ 50 znaků), prázdný řádek, případně delší popis
- Vysvětluje „proč“ a kontext, ne jen „co“

Best practices

Zprávy commitů a konvence

- **Imperativní styl:** „Add login validation“ (ne „Added“ / „Adds“)
- **Stručný předmět** (≤ 50 znaků), prázdný řádek, případně delší popis
- Vysvětluje „proč“ a kontext, ne jen „co“

Best practices

Obsah commitů

Best practices

Obsah commitů

- **Atomické** úpravy: jeden logický krok = jeden commit

Best practices

Obsah commitů

- **Atomické** úpravy: jeden logický krok = jeden commit
- **Časté commity**, které jsou srozumitelně popsané

Best practices

Obsah commitů

- **Atomické** úpravy: jeden logický krok = jeden commit
- **Časté commity**, které jsou srozumitelně popsané
- **Otestovaná lokální práce** před odesláním do vzdáleného repozitáře

Best practices

Obsah commitů

- **Atomické** úpravy: jeden logický krok = jeden commit
- **Časté commity**, které jsou srozumitelně popsané
- **Otestovaná lokální práce** před odesláním do vzdáleného repozitáře
- Tvorba **revert commitů**, **neupravovat historii sdílených větví**

Best practices

Obsah commitů

- **Atomické** úpravy: jeden logický krok = jeden commit
- **Časté commity**, které jsou srozumitelně popsané
- **Otestovaná lokální práce** před odesláním do vzdáleného repozitáře
- Tvorba **revert commitů**, **neupravovat historii sdílených větví**
- **Nesdílet citlivé údaje** (klíče, tokeny) do vzdáleného repozitáře