

# Nástroje pro porovnání obsahu dvou textových souborů

Pavel Tišnovský 🕒 před 20 hodinami

Nástroj `diff` patří k základním utilitám, s nímž se setká většina administrátorů i vývojářů používajících Linux. Ovšem kromě `diffu` můžeme při porovnávání dvou souborů, popř. i pro jejich synchronizaci použít další nástroje.

---

## Obsah

- [1. Nástroje pro porovnání obsahu dvou textových souborů \(#k01\)](#)
- [2. Klasický unixový nástroj `diff` \(#k02\)](#)
- [3. Způsoby zobrazení rozdílů mezi soubory nástrojem `diff` \(#k03\)](#)
- [4. Formáty, v nichž není zobrazen kontext \(#k04\)](#)
  - [4.1 Výchozí formát \(#k041\)](#)
  - [4.2 Formát pro RCS \(#k042\)](#)
  - [4.3 Formát pro `ed`, `vi` či `Vim` \(#k043\)](#)
- [5. Formáty s volitelným kontextem \(#k05\)](#)
  - [5.1 Základní formát s kontextem \(#k051\)](#)
  - [5.2 Unifikovaný formát \(#k052\)](#)
  - [5.3 Zobrazení rozdílů ve dvou sloupcích \(#k053\)](#)
- [6. Vylepšujeme výstup z `diffu` \(#k06\)](#)
- [7. Filtre `idiff` aneb barevné zobrazení rozdílů \(#k07\)](#)
- [8. Filtre `diff-so-fancy` \(#k08\)](#)
- [9. Použití filtru `diff-so-fancy` s `Git`em \(#k09\)](#)
- [10. Filtre `diffh` určený pro vygenerování HTML stránky se zobrazením rozdílů \(#k10\)](#)
- [11. Další utility pro porovnání souborů s textovým rozhraním \(#k11\)](#)
- [12. Nástroj `sdiff` \(#k12\)](#)
- [13. Nástroj `wdiff` \(#k13\)](#)
- [14. Diff režim `Vim`u \(#k14\)](#)
- [15. Nástroj `mcdiff` \(#k15\)](#)

[16. Utility s plnohodnotným GUI \(#k16\)](#)[17. TkDiff \(#k17\)](#)[18. xxdiff \(#k18\)](#)[19. Další užitečné nástroje popsané příště \(#k19\)](#)[20. Odkazy na Internetu \(#k20\)](#)

## 1. Nástroje pro porovnání obsahu dvou textových souborů

Nedávno na Rootu vyšla [zprávička](https://www.root.cz/zpravicky/jak-hledat-rozdily-v-textovych-souborech/) (<https://www.root.cz/zpravicky/jak-hledat-rozdily-v-textovych-souborech/>) o článku s tématem, jak v Linuxu hledat rozdíly v textových souborech (<https://www.maketecheasier.com/use-diff-compare-files-linux/>). Zmíněný článek se zabýval základním nástrojem **diff** a taktéž aplikací *Meld* s plnohodnotným grafickým uživatelským rozhraním. Ovšem porovnání dvou souborů, popř. i jejich synchronizace je tak často používaná operace, že postupně vzniklo několik desítek dalších nástrojů, které buď doplňují možnosti **diffu** (skvělý [diff-so-fancy \(#k09\)](#)), nebo je lze použít zcela samostatně. V tomto článku si některé z těchto nástrojů představíme.

Poznámka: samostatnou kapitolu tvoří utility pro třicestný merge. Těm se budeme podrobněji věnovat příště.

## 2. Klasický unixový nástroj diff

První nástroj, o kterém se v tomto článku zmíníme, možná ani není nutné čtenářům Rootu podrobně představovat, protože se s ním už pravděpodobně setkali. Tento nástroj se jmenuje **diff** a jedním z důvodů, proč se o něm zmiňujeme hned v úvodních kapitolách, je fakt, že první verze **diffu** vznikla již na začátku sedmdesátých let minulého století (v roce 1974 již například vyšla verze založená na stále používaném [Hunt-McIlroyově algoritmu](https://en.wikipedia.org/wiki/Hunt%E2%80%93McIlroy_algorithm) ([https://en.wikipedia.org/wiki/Hunt%E2%80%93McIlroy\\_algorithm](https://en.wikipedia.org/wiki/Hunt%E2%80%93McIlroy_algorithm))). Tento nástroj samozřejmě prošel poměrně dlouhým vývojem a různými rozšířeními, ať se to již týká vlastního interního algoritmu pro hledání rozdílů, tak i formátů výstupu, tj. způsobů, jakým **diff** zobrazuje rozdíly mezi porovnávanými soubory. V současné verzi dokáže **diff** porovnat dva soubory, obsah dvou adresářů, popř. rekurzivně procházet a navzájem porovnat zvolenou dvojici adresářů. Pokud výstup produkováný **diffem** uložíme do souboru, získáme tzv. *patch*, který je možné (například na jiném počítači) aplikovat na původní soubor a vlastně tak znovu provést historii editace (podle použitého formátu lze použít nástroje **patch**, **ed**, **vi** či **Vim**).

Již v předchozím odstavci jsme se zmínili o tom, že současné verze nástroje **diff** podporují několik způsobů zobrazení rozdílů mezi dvěma soubory. Vyžadovaný formát se volí pomocí přepínačů zadaných na příkazové řádce:

Přepínač	Dlouhá verze	Význam
	--normal	<u>výchozí formát</u>
-n	--rcs	<u>formát používaný</u> v dnes již archaickém systému <b>RCS</b>
-e	--ed	vytváří <u>skript</u> spustitelný v editorech <b>ed</b> , <b>vi</b> a samozřejmě i <b>Vim</b>
-c, -C	--context	<u>základní formát</u> , v němž se kromě změn zobrazuje i kontext
-u, -U	--unified	tzv. <u>unifikovaný formát</u> , v němž se také zobrazuje kontext
-y	--side-by-side	zobrazení rozdílů mezi soubory <u>ve dvou sloupcích</u>

## 3. Způsoby zobrazení rozdílů mezi soubory nástrojem diff

Podívejme se nyní na způsob zobrazení rozdílů mezi dvěma verzemi jednoho zdrojového souboru. Oba soubory, které se budou porovnávat, naleznete na adresách:

1. [starší verze](#)
2. [novější verze](#)

Nejprve (#k04) si popíšeme ty formáty, v nichž není explicitně zobrazen kontext, tj. „okolí změn“ a posléze si ukážeme formáty, v nichž naopak kontext nalezneme. Kontext není důležitý pouze pro čtenáře vytvořených rozdílových souborů, ale například i pro nástroj **patch**, který dokáže změnu aplikovat i ve chvíli, kdy byl soubor mezitím editován a došlo tedy například k posunu řádků apod.

## 4. Formáty, v nichž není zobrazen kontext

### 4.1 Výchozí formát

Pokud nástroj **diff** spustíme jen se specifikací dvou souborů, popř. souboru a adresáře, v němž se nachází soubor stejného jména, dostaneme následující výstup:

```
105a106,109
>     # three new lines
>     # three new lines
>     # three new lines
>
118,120d121
<     # TODO: delete these three lines
<     # TODO: delete these three lines
<     # TODO: delete these three lines
125d125
< @then('I should see 0 components')
127c127
< def check_components(context, num=0, components='', ecosystem=''):
---
> def check_components(context, num, components='', ecosystem=''):
131a132
>     assert json_data is not None
133c134
<     search_results = json_data['result']
---
>     search_results = json_data['analysis']
```

Vidíme, že **diff** zobrazil pouze ty části souborů, které se od sebe odlišují. Do těchto částí pak vložil přesné informace o tom, k jakým změnám došlo. To je důležité, protože výsledek musí být strojově (tudíž jednoznačně) zpracovatelný. Ve výstupu nalezneme tři typy příkazů, přičemž každý příkaz obsahuje číslo řádku popř. rozsah změněných řádků, dále jméno příkazu a potom číslo řádku (či rozsah řádků), ve druhém souboru. Všechna čísla řádků odpovídají původním souborům. Mezi tři podporované příkazy patří:

Příkaz	Mnemotechnická pomůcka	Význam
a	append	rozdíl spočívá v přidání řádků ve druhém souboru
c	change	řádek či řádky byly změněny, následovat bude seznam rozdílů oddělený ---
d	delete	rozdíl spočívá v řádcích, které ve druhém souboru chybí

Příklady:

Celý příkaz	Význam
127c127	došlo ke změně na řádku 127

105a106,109	změna na řádku 105 (v prvním souboru), ve druhém souboru jsou nové čtyři řádky 106 až 109
125d125	ve druhém souboru chybí (oproti souboru prvnímu) řádek číslo 125
118,120d121	ve druhém souboru chybí (oproti souboru prvnímu) tři řádky 118 až 120

## 4.2 Formát pro RCS

Ještě stručnější (a velmi těžce čitelný) je formát používaný v RCS. Ten získáme jednoduše příkazem:

```
diff -n old.py new.py
```

Ve výstupu najdeme pouze dva typy příkazů, a to **d** (delete) a **a** (append). Změna je tedy představována vymazáním řádku a jeho nahrazením jiným řádkem. Každý příkaz navíc obsahuje i počítadlo opakování, tj. kolik řádků se má vložit nebo vymazat:

```
a105 4
    # three new lines
    # three new lines
    # three new lines

d118 3
d125 1
d127 1
a127 1
def check_components(context, num, components='', ecosystem=''):
a131 1
    assert json_data is not None
d133 1
a133 1
    search_results = json_data['analysis']
```

## 4.3 Formát pro ed, vi či Vim

Z technického hlediska je zajímavější přepínač **-e**, protože ten produkuje skripty, které je možné spustit jak v editoru **ed** (ten velmi pravděpodobně máte nainstalovaný, i když jste ho možná nikdy nepoužili), tak i ve Vimu. Na řádcích označených **<Esc>** se nachází jediný znak – escape (kód 27 v ASCII), protože právě tímto znakem se v ed i Vim ukončují příkazy „append“, „change“ i „delete“:

```
133c
    search_results = json_data['analysis']
<Esc>
131a
    assert json_data is not None
<Esc>
127c
def check_components(context, num, components='', ecosystem=''):
<Esc>
125d
118,120d
105a
    # three new lines
    # three new lines
    # three new lines

<Esc>
```

Pokud si vytvoříte skript příkazem:

```
diff -e old.py new.py > patch.vim
```

Je možné ve Vim otevřít původní soubor **old.py** a pomocí příkazu:

```
:source patch.vim
```

aplikovat jednotlivé editační příkazy uložené ve skriptu.

Popř. lze vše provést z příkazového řádku:

```
vim old.py -S patch.vim
```

Poznámka: ve skutečnosti je mnohem častější i praktičtější použití dále popsaných formátů, které jsou zpracovatelné utilitou **patch**.

## 5. Formáty s volitelným kontextem

Výše uvedené (#k04) tři formáty zobrazují pouze rozdíly mezi soubory, ale nepřidávají do vytvořeného výstupu žádný kontext, tj. oblast kódu, v níž ke změně došlo. Proto nejsou takové rozdílové soubory příliš čitelné, což nám však ve skutečnosti nemusí příliš vadit, protože **diff** podporuje i další výstupní formáty. Velkou předností existence kontextu je fakt, že případné změny je možné aplikovat i ve chvíli, kdy se (další editací) změní počet řádků nebo dojde k dalším změnám v souboru, na který je *patch* aplikován (to se ovšem týká utility **patch** a nikoli nástroje **diff**).

### 5.1 Základní formát s kontextem

Jedním ze základních formátů, který zobrazuje i kontextové informace, je formát zapnutý přepínačem **-c**. Výstup při použití příkazu:

```
diff -c old.py new.py
```

vypadá značně odlišně, protože se namísto jednopísmenných příkazů do prvního sloupce zapisují značky, které označují přidané či naopak smazané řádky (z pohledu historie). Dále si povšimněte, že změny jsou (pokud je to možné) sloučené do skupin pojmenovaných *hunk*. Každý *hunk* začíná řádkem s hvězdičkami, za ním následují údaje o řádcích v prvním i druhém souboru, jichž se *hunk* týká a poté jsou již jednotlivé řádky z *hunku* vypsané. V prvním sloupci znak mezery znamená, že řádek nebyl změněn, ! značí změněný řádek, – řádek vymazaný (chybí ve druhém souboru) a + řádek přidaný (do druhého souboru oproti souboru prvnímu):

```
*** old.py      2018-01-23 17:19:26.424398470 +0100
--- new.py      2018-01-23 17:19:40.856322400 +0100
*****
*** 103,108 ****
--- 103,112 ----

        use_token = parse_token_clause(token)

+   # three new lines
+   # three new lines
+   # three new lines
+
        url = component_analysis_url(context, ecosystem, component, version)

        for _ in range(timeout // sleep_amount):
*****
*** 115,136 ****
        elif status_code != 404:
            raise Exception('Bad HTTP status code {c}'.format(c=status_code))
            time.sleep(sleep_amount)
-   # TODO: delete these three lines
-   # TODO: delete these three lines
-   # TODO: delete these three lines
        else:
            raise Exception('Timeout waiting for the component analysis results')

- @then('I should see 0 components')
+ @then('I should see {num:d} components ({components}), all from {ecosystem} ecosystem')
! def check_components(context, num=0, components='', ecosystem=''):
    """Check that the specified number of components can be found."""
    components = split_comma_separated_list(components)

    json_data = context.response.json()

!     search_results = json_data['result']
    assert len(search_results) == num
    for search_result in search_results:
        assert search_result['ecosystem'] == ecosystem
--- 119,137 ----
```

```

        elif status_code != 404:
            raise Exception('Bad HTTP status code {c}'.format(c=status_code))
        time.sleep(sleep_amount)
    else:
        raise Exception('Timeout waiting for the component analysis results')

    @then('I should see {num:d} components ({components}), all from {ecosystem} ecosystem')
! def check_components(context, num, components='', ecosystem=''):
    """Check that the specified number of components can be found."""
    components = split_comma_separated_list(components)

    json_data = context.response.json()
+   assert json_data is not None

!   search_results = json_data['analysis']
    assert len(search_results) == num
    for search_result in search_results:
        assert search_result['ecosystem'] == ecosystem

```

Namísto volby **-c** můžeme použít i volbu **-C číslo**, kterou se určuje, kolik nezměněných řádků se má zahrnout do jednotlivých hunků. Pokud budeme například vyžadovat pět řádků na začátku a na konci, použijeme příkaz:

```
diff -C 5 old.py new.py
```

s následujícím výsledkem:

```

*** old.py      2018-01-23 17:19:26.424398470 +0100
--- new.py      2018-01-23 17:19:40.856322400 +0100
*****
*** 101,110 ****
--- 101,114 ----
        timeout = context.component_analysis_timeout # in seconds
        sleep_amount = 10 # we don't have to overload the API with too many calls

        use_token = parse_token_clause(token)

+       # three new lines
+       # three new lines
+       # three new lines
+
        url = component_analysis_url(context, ecosystem, component, version)

        for _ in range(timeout // sleep_amount):
            if use_token:
                status_code = requests.get(url, headers=authorization(context)).status_code
*****
*** 113,138 ****
        if status_code == 200:
            break
        elif status_code != 404:
            raise Exception('Bad HTTP status code {c}'.format(c=status_code))
        time.sleep(sleep_amount)
-       # TODO: delete these three lines
-       # TODO: delete these three lines
-       # TODO: delete these three lines
    else:
        raise Exception('Timeout waiting for the component analysis results')

- @then('I should see 0 components')
    @then('I should see {num:d} components ({components}), all from {ecosystem} ecosystem')
! def check_components(context, num=0, components='', ecosystem=''):
    """Check that the specified number of components can be found."""
    components = split_comma_separated_list(components)

    json_data = context.response.json()

!   search_results = json_data['result']
    assert len(search_results) == num
    for search_result in search_results:
        assert search_result['ecosystem'] == ecosystem
        assert search_result['name'] in components

--- 117,139 ----
        if status_code == 200:
            break
        elif status_code != 404:
            raise Exception('Bad HTTP status code {c}'.format(c=status_code))

```

```

        time.sleep(sleep_amount)
    else:
        raise Exception('Timeout waiting for the component analysis results')

    @then('I should see {num:d} components ({components}), all from {ecosystem} ecosystem')
    ! def check_components(context, num, components='', ecosystem=''):
        """Check that the specified number of components can be found."""
        components = split_comma_separated_list(components)

        json_data = context.response.json()
+       assert json_data is not None

    !       search_results = json_data['analysis']
        assert len(search_results) == num
        for search_result in search_results:
            assert search_result['ecosystem'] == ecosystem
            assert search_result['name'] in components

```

## 5.2 Unifikovaný formát

Pravděpodobně nepoužívanějším formátem rozdílových souborů, který podporují všechny moderní varianty **diffu**, je takzvaný unifikovaný formát. Tento formát se povoluje přepínačem **-u**, takže volání **diffu** může vypadat následovně:

```
diff -u old.py new.py
```

Výsledek vypadá odlišně od předchozího formátu. Především můžeme vidět, že se informace o místu, v němž ke změně došlo, zapisují na jediný řádek uvozený dvojicí znaků @@. Před čísly řádků je znak + či - podle toho, jakého souboru se číslo týká (zda souboru prvního či druhého), číslo za čárkou značí počet řádků v hunku (včetně kontextových řádků před a za změnou). Vložené řádky začínají znakem + v prvním sloupci, řádky smazané znakem -. I pokud na řádku došlo jen k nepatrné změně, je tato změna zaznamenána dvěma operacemi: vymazáním původního řádku a vložením řádku s novým obsahem. Díky tomu, že se kromě vlastních změněných řádků zobrazí i kontext (zde konkrétně tři řádky před a za změnou), může utilita **patch** aplikovat změny i ve chvíli, kdy byl soubor mezitím změněn:

```

--- old.py      2018-01-23 17:19:26.424398470 +0100
+++ new.py      2018-01-23 17:19:40.856322400 +0100
@@ -103,6 +103,10 @@

    use_token = parse_token_clause(token)

+   # three new lines
+   # three new lines
+   # three new lines
+
    url = component_analysis_url(context, ecosystem, component, version)

    for _ in range(timeout // sleep_amount):
@@ -115,22 +119,19 @@
        elif status_code != 404:
            raise Exception('Bad HTTP status code {c}'.format(c=status_code))
            time.sleep(sleep_amount)
-       # TODO: delete these three lines
-       # TODO: delete these three lines
-       # TODO: delete these three lines
        else:
            raise Exception('Timeout waiting for the component analysis results')

-@then('I should see 0 components')
@then('I should see {num:d} components ({components}), all from {ecosystem} ecosystem')
-def check_components(context, num=0, components='', ecosystem=''):
+def check_components(context, num, components='', ecosystem=''):
    """Check that the specified number of components can be found."""
    components = split_comma_separated_list(components)

    json_data = context.response.json()
+   assert json_data is not None

-   search_results = json_data['result']
+   search_results = json_data['analysis']
    assert len(search_results) == num

```

### 5.3 Zobrazení rozdílů ve dvou sloupcích

Podívejme se na výsledek, který však byl zkrácený o shodné řádky. Povšimněte si, jak se pomocí znaků > a < označují ty řádky, které byly přidány či naopak vymazány. Změněné řádky jsou označeny znakem |:

## 6. Vylepšujeme výstup z diffu

## 7. Filtr idiff aneb barevné zobrazení rozdílů

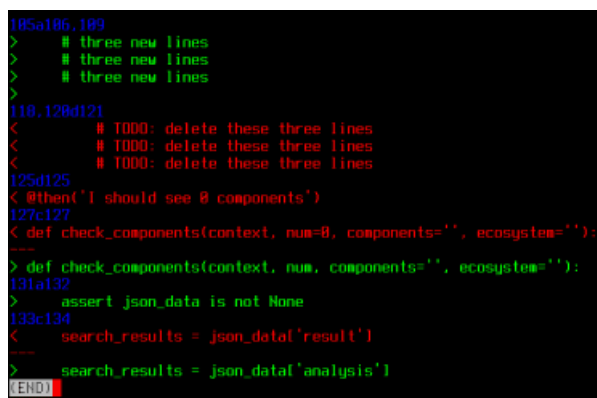
25. 1. 2018. 19:56



/scripts/iddiff). Překlad není zapotřebí provést, protože **iddiff** je napsán v shellu a je založen na nástrojích **less**, **sed** a **tput**, které by již měly být nainstalovány. Tento skript pracuje jako běžný filtr, tj. lze ho použít například takto (pokud se **iddiff** nachází v pracovním adresáři a má nastaveno právo „x“ pro uživatele):

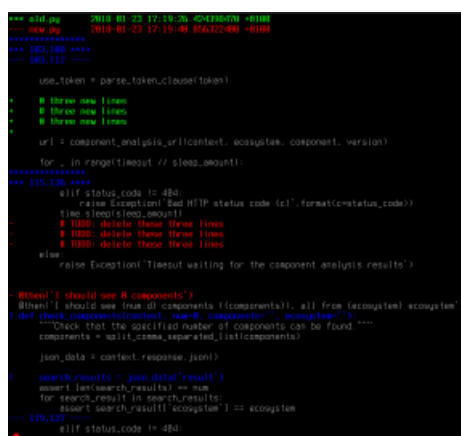
```
diff -u old.py new.py | ./iddiff
```

Možnosti skriptu jsou vidět na následující trojici screenshotů:



```
105a106,109
> # three new lines
> # three new lines
> # three new lines
>
110,120d121
< # T000: delete these three lines
< # T000: delete these three lines
< # T000: delete these three lines
125d125
< @then('I should see 0 components')
127c127
< def check_components(context, num=0, components='', ecosystem=''):
----
> def check_components(context, num, components='', ecosystem=''):
131a132
> assert json_data is not None
133c134
< search_results = json_data['result']
----
> search_results = json_data['analysis']
(END)
```

Obrázek 1: Obarvený výstup při použití standardního formátu **diffu**.



```
*** old.py      2018-01-23 17:19:56.403000000 +0100
--- new.py      2018-01-23 17:19:56.403000000 +0100
-----
*** 101,100 ****
--- 101,112 ----
     use_token = parse_token_clause(token)

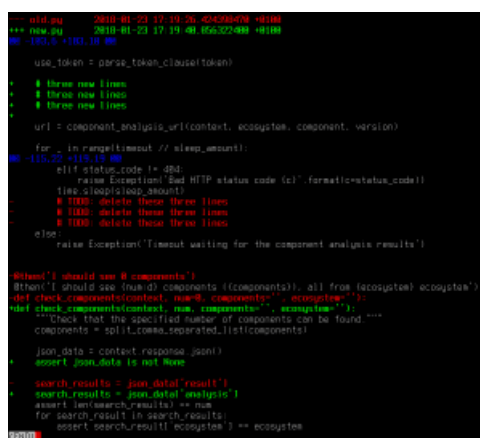
> # three new lines
> # three new lines
> # three new lines
>

url = component_analysis_url(context, ecosystem, component, version)
for .. in range(timeout // sleep_amount):
*** 110,110 ****
--- 110,122 ----
         elif status_code != 404:
             raise Exception('Bad HTTP status code (%d)' % status_code)
             time.sleep(sleep_amount)
             > T000: delete these three lines
             > T000: delete these three lines
             > T000: delete these three lines
         else:
             raise Exception('Timeout waiting for the component analysis results')

@then('I should see 0 components')
@then('I should see found components (%components), all from (%ecosystem) ecosystem')
def check_components(context, num=0, components='', ecosystem=''):
    """Check that the specified number of components can be found"""
    components = split_comma_separated_list(components)

    json_data = context.response.json()
    search_results = json_data['result']
    assert len(search_results) == num
    for search_result in search_results:
        assert search_result['ecosystem'] == ecosystem
*** 110,122 ****
--- 110,122 ----
         elif status_code != 404:
```

Obrázek 2: Obarvený výstup při použití formátu s kontextem (**-c**).



```
*** old.py      2018-01-23 17:19:56.403000000 +0100
--- new.py      2018-01-23 17:19:56.403000000 +0100
@@ -101,6 +101,12 @@
     use_token = parse_token_clause(token)

> # three new lines
> # three new lines
> # three new lines
>

url = component_analysis_url(context, ecosystem, component, version)
for .. in range(timeout // sleep_amount):
@@ -110,22 +110,12 @@
         elif status_code != 404:
             raise Exception('Bad HTTP status code (%d)' % status_code)
             time.sleep(sleep_amount)
             > T000: delete these three lines
             > T000: delete these three lines
             > T000: delete these three lines
         else:
             raise Exception('Timeout waiting for the component analysis results')

@then('I should see 0 components')
@then('I should see found components (%components), all from (%ecosystem) ecosystem')
def check_components(context, num=0, components='', ecosystem=''):
    """Check that the specified number of components can be found"""
    components = split_comma_separated_list(components)

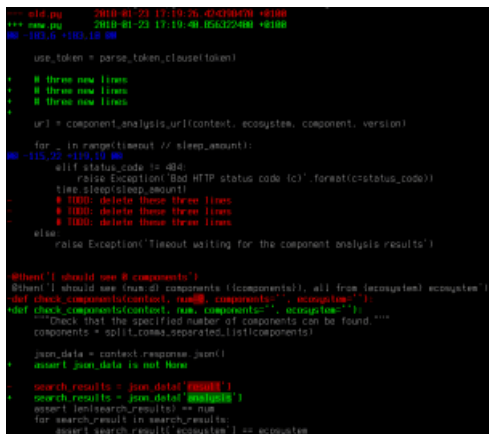
    json_data = context.response.json()
    assert json_data is not None
    search_results = json_data['result']
    assert len(search_results) == num
    for search_result in search_results:
        assert search_result['ecosystem'] == ecosystem
```

Obrázek 3: Obarvený výstup při použití unifikovaného formátu (**-u**). Pro zajímavost si porovnejte tento screenshot se screenshotem číslo 4.

Skript **iddiff** si samozřejmě můžete zkopírovat do adresáře, který se nachází na **PATH**. Potom se k němu nemusí uvádět cesta.

## 8. Filtr diff-so-fancy

Druhým filtrem, tentokrát již mnohem zajímavějším, je skript nazvaný **diff-so-fancy**, který je naprogramovaný v Perlu a můžete ho získat z Git repositáře <https://github.com/so-fancy/diff-so-fancy> (<https://github.com/so-fancy/diff-so-fancy>) (opět není zapotřebí provést žádný překlad, pouze umístění skriptu do adresáře v PATH). Jméno tohoto skriptu je příhodné, protože **diff-so-fancy** nejenom že zvýrazní rozdíly provedené v rámci jednoho řádku (ty je jinak mnohdy složité najít), ale dokáže upravit a především zpřehlednit i výstup z příkazu **git diff** atd.



Obrázek 4: Povšimněte si, jak **diff-so-fancy** dokáže zvýraznit rozdíly provedené na jediném řádku (spodní polovina screenshotu).

Pokud potřebujeme pouze zpřehlednit výstup z klasického **diffu**, můžeme spojit možnosti obou filtrů, tj. provést tento příkaz:

```
diff -u old.py new.py | idiff | diff-so-fancy
```

Jednodušší je si napsat příslušnou funkci představující nový příkaz shellu do **.bashrc** do **.bash\_profile**:

```
function bestdiff() {
    diff -u $1 $2 | idiff | diff-so-fancy
}
```



Obrázek 5: Ještě lépe jsou schopnosti skriptu v zobrazení změn provedených v rámci jednoho řádku patrné u zobrazení rozdílů v běžném textu (úvodní odstavec článku na Wikipedii o Linuxu).

## 9. Použití filtru diff-so-fancy s GItem

Filtr **diff-so-fancy** je určen pro spolupráci s GItem a dokonce GIT používá i ve chvíli, kdy pouze mění výstup z běžného **diffu** (čte informace o použitých barvách terminálu). Aby tento filtr pracoval správně, je nutné do souboru **~/.gitconfig** vložit nové řádky s konfigurací barev. Výchozí hodnoty, které můžete připojit na konec GITovského konfiguračního souboru, vypadají následovně:

```
[color]
    ui = true

[color "diff-highlight"]
    oldNormal = red bold
    oldHighlight = red bold 52
    newNormal = green bold
    newHighlight = green bold 22

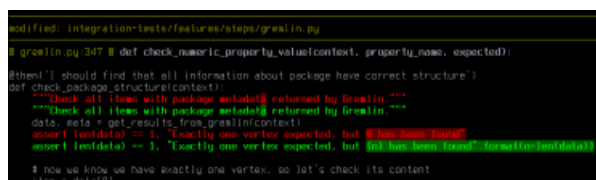
[color "diff"]
    meta = yellow
    frag = magenta bold
    commit = yellow bold
    old = red bold
    new = green bold
    whitespace = red reverse
```

Poté již můžeme použít například příkaz:

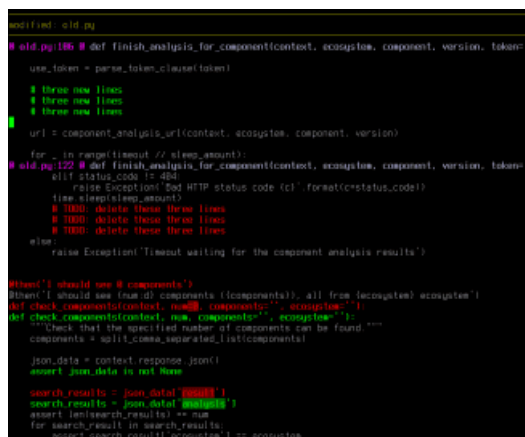
```
git diff --color | diff-so-fancy
```

Popř. změnit konfiguraci GITu tak, aby se **diff-so-fancy** používal vždy (změny se opět zapíší do souboru `~/.gitconfig`):

```
git config --global core.pager "diff-so-fancy | less --tabs=4 -RFX"
```



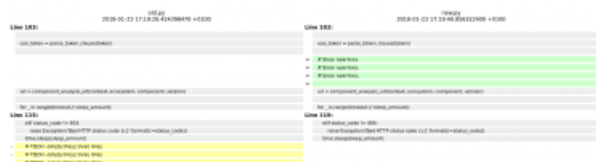
Obrázek 7: Takto vypadá výstup příkazu **git diff** při použití skriptu **diff-so-fancy**.



Obrázek 8: Po vložení testovacího souboru do GIT repozitáře se můžeme na změny podívat i přes **git diff** (porovnejte se screenshotem číslo 4).

## 10. Filtr diffh určený pro vygenerování HTML stránky se zobrazením rozdílů

Zatímco předchozí dva filtry upravovaly výstup **diffu** takovým způsobem, aby se vylepšilo zobrazení na běžném textovém terminálu, pracuje filtr nazvaný **diffh** odlišným způsobem, protože jeho výstupem je HTML stránka se zvýrazněním rozdílů:





Obrázek 9: Výstup generovaný nástrojem **diffh**.

Tento filtr je kupodivu naprogramovaný v jazyku C, takže pokud **diffh** chybí v repositáři vaší distribuce, musíte si provést překlad sami. Ve skutečnosti je to velmi jednoduché, protože má tento nástroj jen minimální závislosti:

1. Stáhněte zdrojové kódy z adresy <https://sourceforge.net/projects/diffh/>
2. Rozbalte tarball: **tar xvzf diffh-0.3.2.tar.gz**
3. Ve vytvořeném adresáři spusťte **./configure**
4. Následuje klasické **make**
5. Výsledný binární soubor objevíte v adresáři **src**

## 11. Další utility pro porovnání souborů s textovým rozhraním

Krátce se zmíním ještě o dalších utilitách, které slouží pro porovnání souborů a které taktéž používají textové rozhraní, ať již „pouze“ výstup na konzoli či plnohodnotné TUI.

## 12. Nástroj **sdiff**

Nástroj nazvaný **sdiff** (který pravděpodobně máte nainstalovaný) získal svoje jméno proto, že dokáže porovnávat dvě verze souborů „side-by-side“. Způsob zobrazení odpovídá příkazu **diff -y**, ovšem důvod, proč se o tomto nástroji dnes vůbec zmiňuji, souvisí s tím, že **sdiff** podporuje i interaktivní režim, v němž může uživatel postupně procházet jednotlivými změnami a ty aplikovat (schvaluje je) či přeskočit. Výstup produkovaný tímto nástrojem může vypadat následovně:

```
@then('I should see 0 components')
@then('I should see {num:d} components ({components}), all fr
def check_components(context, num=0, components='', ecosystem | def check_components(context, num, c
    """Check that the specified number of components can be f    """Check that the specified numb
    components = split_comma_separated_list(components)    components = split_comma_separated_l

    json_data = context.response.json()    json_data = context.response.json()
    >    assert json_data is not None

    search_results = json_data['result']    search_results = json_data['analysis']
    assert len(search_results) == num    assert len(search_results) == num
    for search_result in search_results:    for search_result in search_results:
        assert search_result['ecosystem'] == ecosystem        assert search_result['ecosystem']
        assert search_result['name'] in components        assert search_result['name'] in comp
```

## 13. Nástroj **wdiff**

Jméno dalšího nástroje **wdiff** vzniklo ze slovního spojení „word diff“. A skutečně – **wdiff** zobrazuje rozdíly na úrovni jednotlivých slov a nikoli celých řádků. To nemá (alespoň podle mého názoru) větší význam pro zdrojové kódy, protože se informace o změnách ztratí ve změti dalších řádků; ostatně to můžete sami posoudit:

```
@then('I should see [-0 components')
@then('I should see-] {num:d} components ({components}), all from {ecosystem} ecosystem')
def check_components(context, [-num=0,-] {+num,+} components='', ecosystem=''):
    """Check that the specified number of components can be found."""
    components = split_comma_separated_list(components)

    json_data = context.response.json()
```

```
{+assert json_data is not None+}

search_results = [-json_data['result']-] {+json_data['analysis']+}
assert len(search_results) == num
for search_result in search_results:
    assert search_result['ecosystem'] == ecosystem
    assert search_result['name'] in components
```

Ovšem pokud potřebujete porovnat například dvě verze souboru README, může být **wdiff** velmi užitečný. Podívejme se na ukázkový příklad s prvními dvěma odstavci o Linuxu, které jsem získal z Wikipedie a provedl v nich malé změny. Rozdíly mezi dvěma verzemi souboru se zobrazí formou značek [- ... -] a {+ ... +}, které jsem navíc zvýraznil tučně (větší změny by však již nebyly zobrazeny takto přehledně):

```
Linux is a name that broadly denotes a family of free and [-closed-source-] {+open-source+} software
operating systems (OS) built around the Linux kernel. Typically, Linux is
packaged in a form known as {+a+} Linux distribution (or distro for short) for both
desktop and server use. The defining component of a Linux distribution is the
Linux kernel,[11] an operating system kernel first released on September 17,
1991, by Linus Torvalds.[12][13][14] Many Linux distributions use the word
"Linux" in their name. The Free Software Foundation uses the name GNU/Linux to
refer to the operating system family, as well as specific distributions, to
emphasize that most Linux distributions are not just the Linux kernel, and that
they have in common not only {+the+} kernel, but also numerous utilities and
libraries, a large proportion of which are from the GNU project. This has led
to some controversy.
```

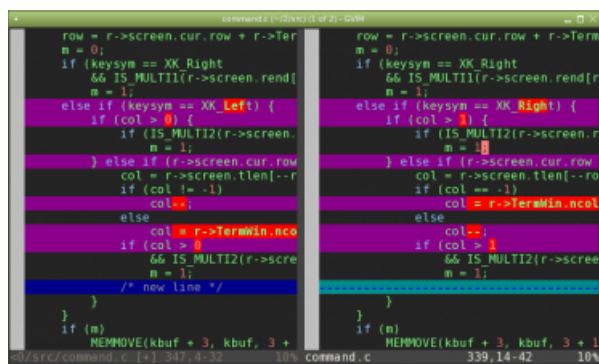
```
Linux was originally developed for personal computers based on the Intel x86
architecture, but has since been ported to more platforms than any other
operating system.[17] Because of the dominance of the Linux kernel-based
Android OS on smartphones, Linux has the largest installed base of all
general-purpose operating systems.[18] Linux is also the leading operating
system on servers and other big iron systems such as mainframe computers, and
the only OS used on TOP500 supercomputers (since November 2017, having before
gradually eliminated all competitors).[19][20] It is used by around 2.3% of
desktop computers.[21][22] The Chromebook, which runs the Linux kernel-based
Chrome OS, dominates the US K-12 education market and represents nearly [-12%-] {+20%+} of
the sub-$300 notebook sales in the US.[23] [-GNU/Linux-] {+Linux+} also runs on embedded
systems—devices whose operating system is typically built into the firmware and
is highly tailored to the system. This includes TiVo and similar {+DVR+} devices,
network routers, facility automation controls, televisions,[24][25] video game
consoles and smartwatches.[26] Many smartphones and tablet computers run
Android and other Linux derivatives.[27]
```

## 14. Diff režim Vimu

Pro zobrazení rozdílů mezi dvěma verzemi souboru i pro jejich synchronizaci je možné využít i interní nástroj zabudovaný do Vimu. Pokud se tento textový editor spustí s parametrem **-d**, očekávají se za tímto parametrem jména dvou souborů pro porovnání, popř. jméno souboru a jméno adresáře, v němž se nachází soubor stejného jména:

```
vim -d old.py new.py
vim -d old.py ../test-sources/
```

V obou případech textový editor Vim oba soubory skutečně načte do dvojice bufferů, ovšem režim zobrazení se změní takovým způsobem, že se pomocí různých barev zvýrazní ty části souborů, které jsou odlišné. Navíc je možné delší části, jež jsou v obou souborech stejné, „zabalit“ (klávesová zkratka **zc**).

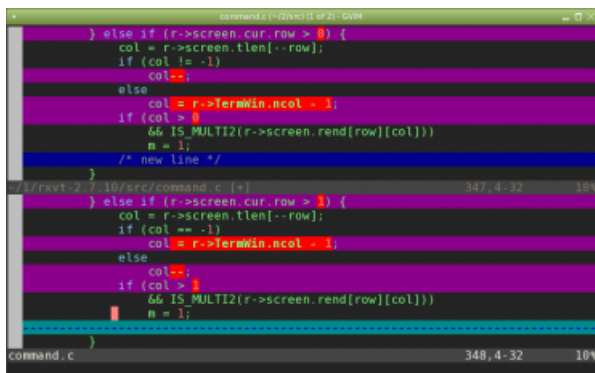


Obrázek 10: Porovnávání dvou souborů při vertikálním rozdělení oken. Toto rozdělení je vhodné použít v případě, že se jedná o zdrojové soubory s krátkými řádky, popř. pokud máte širokoúhlý monitor.

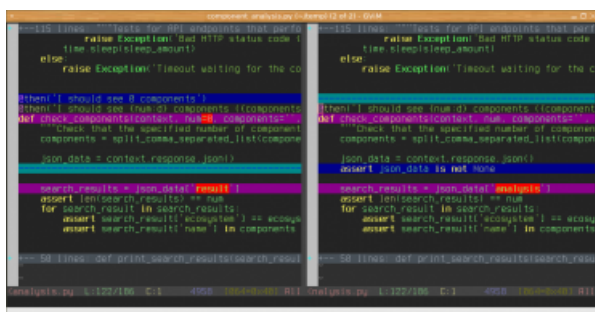
Barevné zvýraznění změn je sice pěkná a užitečná vlastnost, ovšem to není zdaleka vše, co Vim v „režimu diff“ programátorům nabízí. V tomto režimu jsou totiž k dispozici některé nové příkazy určené pro řízení synchronizaci mezi soubory. Mezi tyto příkazy patří především:

Příkaz	Význam
[c	skok na začátek předchozího bloku se změnami
]c	skok na začátek následujícího bloku se změnami
dp	pokud se kurzor nachází na řádcích, které se v obou souborech odlišují, je změna z aktuálního souboru přenesena do souboru druhého
do	opak předchozího příkazu – získání změny z druhého souboru a přenesení této změny do souboru aktuálního (tj. do souboru, v němž se nachází textový kurzor)
:diffupdate	tento příkaz provede nové vyhodnocení rozdílů mezi oběma soubory, vhodné v případech, kdy se oba buffery rozsynchronizují (to se stává poměrně často)

Příkazy **dp** a **do** lze použít i ve chvíli, kdy je vybraný blok textu. V tomto případě se změna aplikuje pouze na vybrané řádky, nikoli na celý blok změn.



Obrázek 11: Porovnávání dvou souborů při horizontálním rozdělení oken, které je vhodné využít v případě, že porovnávané zdrojové kódy (nebo jiné texty) obsahují dlouhé řádky.

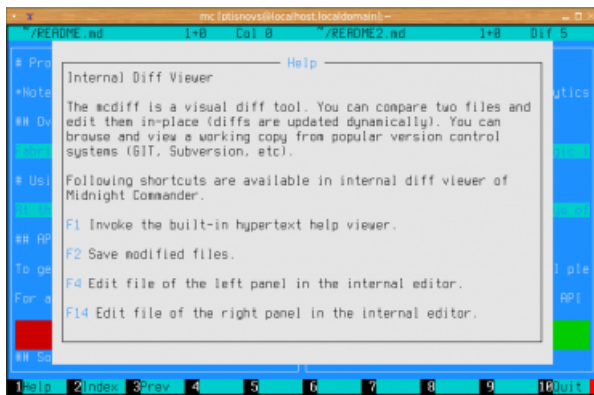


Obrázek 12: Zobrazení našeho testovacího souboru ve Vimě v režimu rozpoznávání změn.

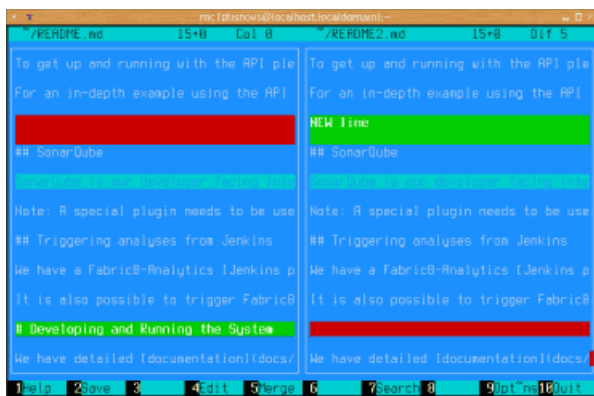
## 15. Nástroj mcdiff

Pokud preferujete použití nástroje pro zobrazení změn, který je založen na textovém uživatelském rozhraní, může být jednou z dobrých voleb utilitka nazvaná **mcdiff**, která je, podobně jako editor **mcedit** nebo **hexa** prohlížeče (<https://www.root.cz/clanky/hexadecimalni-prohlizece-a-editor-s-textovym-uzivatelskym-rozhranim/#k08>), součástí správce souborů *Midnight Commander*. Použití nástroje **mcdiff** je extrémně jednoduché – pouze ho zavolejte a předejte mu dvojici souborů (resp. přesněji řečeno verzí téhož souboru). Zobrazí se dva panely s obsahy obou verzí a uživateli jsou dány k dispozici nástroje pro sloučení změn,

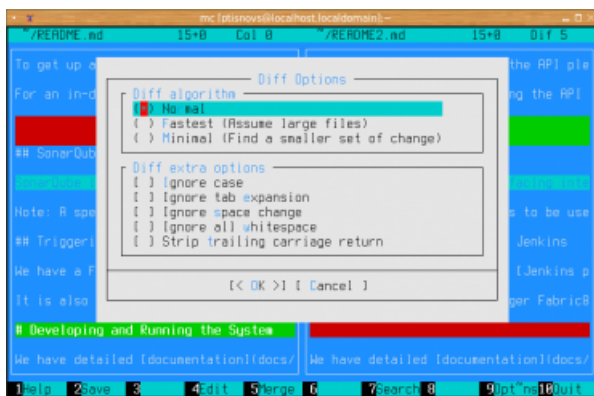
vyhledávání v obou verzích souboru atd. Klávesa pro sloučení změn se vždy vztahuje k prvnímu viditelnému rozdílu.



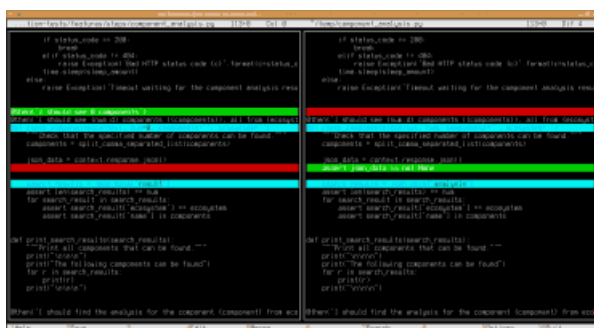
Obrázek 13: Integrovaná nápověda k nástroji **mcdiff**.



Obrázek 14: Zobrazení rozdílů v textových souborech.



Obrázek 15: Nastavení vlastností porovnávače **mcdiff**.



Obrázek 16: Změna barvové palety a zobrazení našeho testovacího souboru.

## 16. Utility s plnohodnotným GUI



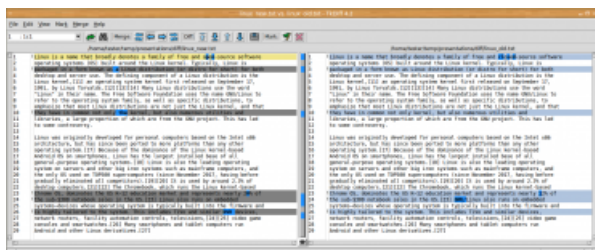
Následuje popis některých GUI aplikací pro zobrazení rozdílů mezi soubory a popř. i pro slučování těchto souborů. V této oblasti se pravděpodobně nejvíce používá *Meld* a *KDiff3*, ovšem i další utility mohou být zajímavé. To se týká jak postaršího *TkDiffu* ([#k17](#)), tak i méně známé utility *xxdiff* ([#k18](#)). O dalších nástrojích ([#k19](#)) se zmíním ve druhé části článku.

## 17. TkDiff

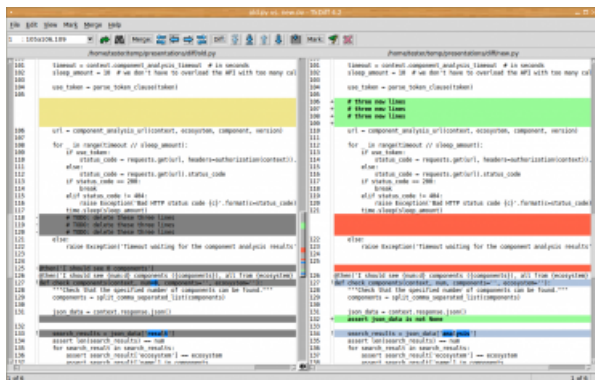
Další nástroj, s nímž se v dnešním článku alespoň ve stručnosti seznámíme, se jmenuje *TkDiff*. Jedná se o nástroj s grafickým uživatelským rozhraním a jak již jeho název napovídá, používá pro svoje GUI knihovnu *Tk*. Samotný *TkDiff* je naprogramován v jazyku TCL (<https://www.root.cz/serialy/programovaci-jazyk-tcl/>), který již sice dnes v žádném případě nestojí na předních místech v žebříčku popularity jazyků (<https://www.tiobe.com/tiobe-index/>), nicméně *TkDiff* je stále udržován a je ho možné provozovat prakticky na jakémkoli systému, v němž je dvojice Tcl/Tk nainstalována (to v Linuxu není problém, pro Windows existuje *ActiveTcl* (<https://www.activestate.com/activetcl/>) atd. atd.). Pokud samotný *TkDiff* nenaleznete v repositáři své distribuce, můžete si ho stáhnout z adresy <https://sourceforge.net/projects/tkdiff/files/latest/download> (<https://sourceforge.net/projects/tkdiff/files/latest/download>), rozbalit a přímo použít (žádná kompilace není nutná, protože TCL je čistý interpret).

Nástroj *TkDiff* zobrazuje dvě verze souboru, dokáže přeskakovat mezi jednotlivými částmi, které byly změněny (hunky), označovat tyto části a poté provést operaci „merge“. Pokud začnete tento nástroj používat, je nutné vědět, že se mezi změnami částmi musíte přesouvat pomocí ikon se šipkami a poté každou změnu odsouhlasit příslušným příkazem (ikona se zelenou vlajkou). Jednotlivé schválené změny jsou reprezentovány novými tlačítky na nástrojovém pruhu, takže se k nim můžete kdykoli vrátit. Po označení všech změn, které chcete schválit, je možné provést operaci „merge“. Dále je možné otevřít soubor obsahující informace o konfliktech (je generován například Gitem, pokud dojde k problémům při pokusu o automatické sloučení změn), ovšem jednu funkcionalitu současná verze postrádá – třícestný merge. Zajímavé je, že předchozí verze *TkDiffu* tuto funkci měly implementovanou. Pokud tedy potřebujete použít třícestný merge, který je skutečně potřebný, je lepší se poohlédnout po jiném nástroji, například po utilitě *Kdiff3*, kterou si popíšeme příště.

Podívejme se, jak vypadá uživatelské rozhraní tohoto nástroje při otevření dvou verzí jednoho souboru:



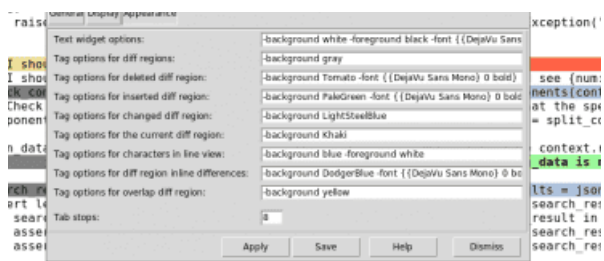
Obrázek 17: Porovnání textových souborů (plaintext).



Obrázek 18: Porovnání našeho testovacího souboru (povšimněte si scrollbaru s náhledem změn).







```
nt search_results(search_results):
    Print all components that can be found.***
    print("\n\n")

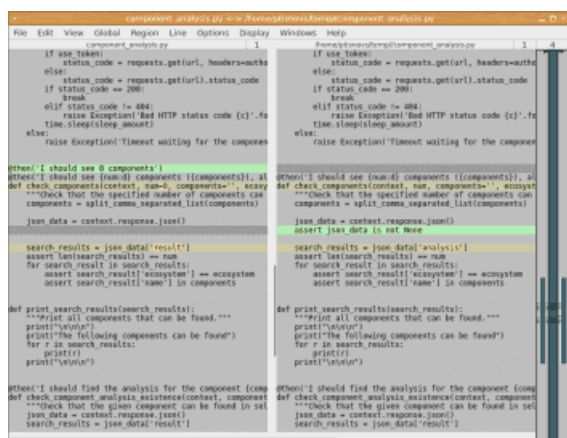
136
137 def print_search_results:
138     """Print all componer
139     print("\n\n")
```

Obrázek 19: Nastavení vlastností TkDiffu.

## 18. xxdiff

Poslední nástroj s grafickým uživatelským rozhraním, který si dnes představíme a který pravděpodobně naleznete i v repozitářích své Linuxové distribuce, se jmenuje *xxdiff*. Tento relativně malý a na systémové prostředky nenáročný nástroj sice není tak známý jako *Meld*, ovšem nabízí podobné funkce a navíc je v něm možné velmi snadno provádět třicestný merge, tj. porovnání a aplikování změn mezi třemi verzemi souborů (typicky se jedná o společného předka, soubor změněný lokálně a soubor mezitím změněný někým jiným). Utilitu *xxdiff* lze pro základní operace volat takto:

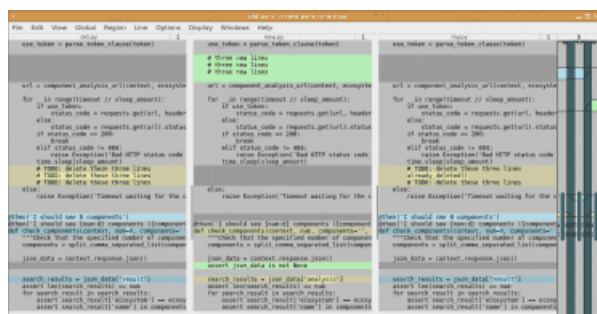
```
xxdiff old.py new.py
```



Obrázek 20: Porovnání dvou verzí souboru (klasický dvoucestný merge).

Popř. pokud budeme potřebovat provést třicestný merge, spustíme utilitu *xdiff* následujícím způsobem:

```
xxdiff puvodni.py muj.py vzdalene_zmeneny.py
```



Obrázek 21: Zobrazení tří verzí souboru při provádění třicestného merge.

K této utilitě se ještě na chvíli vrátíme příště při popisu nástrojů, které podporují provádění třicestného merge.

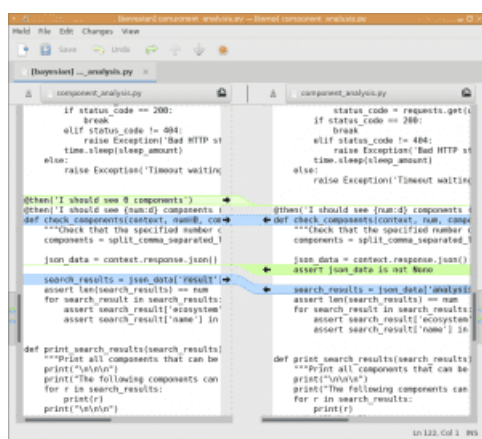




Obrázek 22: Integrovaná nápověda k nástroji xxdiff.

## 19. Další užitečné nástroje popsané příště

Ve druhé části tohoto článku si popíšeme několik dalších nástrojů s plnohodnotným grafickým uživatelským rozhraním. Bude se jednat především o *Meld* (pravděpodobně nejznámější nástroj z této kategorie), dále pak o aplikaci *Diffuse* a samozřejmě nezapomeneme ani na *KDiff3*.



Obrázek 23: Dvoucestný merge našeho testovacího zdrojového kódu v nástroji Meld.

## 20. Odkazy na Internetu

1. Comparing and Merging Files  
<https://www.gnu.org/software/diffutils/manual/diffutils.html>
2. Three-way merge  
[https://en.wikipedia.org/wiki/Merge\\_\(version\\_control\)#Three-way\\_merge](https://en.wikipedia.org/wiki/Merge_(version_control)#Three-way_merge)
3. diff (1) – Linux Man Pages  
<https://www.systutorials.com/docs/linux/man/1-diff/>
4. diff utility (Wikipedia)  
[https://en.wikipedia.org/wiki/Diff\\_utility](https://en.wikipedia.org/wiki/Diff_utility)
5. GNU Wdiff  
<https://www.gnu.org/software/wdiff/>
6. GNU wdiff Manual  
<https://www.gnu.org/software/wdiff/manual/>
7. wdiff (1) – Linux Man Pages  
<https://www.systutorials.com/docs/linux/man/1-wdiff/>
8. diff3 (1) – Linux Man Pages  
<https://www.systutorials.com/docs/linux/man/1-diff3/>
9. sdiff (1) – Linux Man Pages  
<https://www.systutorials.com/docs/linux/man/1-sdiff/>

10. Stránky nástroje Meld  
<http://meldmerge.org/>
11. Meld na stránkách GNOME  
<https://wiki.gnome.org/Apps/Meld>
12. Stránky nástroje TkDiff  
<https://sourceforge.net/projects/tkdiff/>
13. Zdrojové kódy TkDiffu  
<https://sourceforge.net/projects/tkdiff/files/tkdiff/4.2/>
14. Poslední verze nástroje TkDiff  
<https://sourceforge.net/projects/tkdiff/files/latest/download>
15. Manuálová stránka k nástroji TkDiff  
<http://linux.math.tifr.res.in/manuals/man/tkdiff.html>
16. diffh: Make your diff easier to see  
<https://inconsolation.wordpress.com/2013/10/07/diffh-make-your-diff-easier-to-see/>
17. Stránky projektu diffh  
<https://sourceforge.net/projects/diffh/>
18. Pretty Diff (implementovaný v JavaScriptu)  
<http://prettydiff.com/>
19. Nástroje pro diff textů  
<https://en.wikipedia.org/wiki/Diff-Text>
20. Pretty Diff (implementovaný v JavaScriptu)  
[https://en.wikipedia.org/wiki/Pretty\\_Diff](https://en.wikipedia.org/wiki/Pretty_Diff)
21. Stránky projektu colordiff  
<https://www.colordiff.org/>
22. Skript idiff  
<http://www.pixelbeat.org/scripts/idiff>
23. Three way git merging with Meld  
<https://lukas.zapletalovi.com/2012/09/three-way-git-merging-with-meld.html>
24. xxdiff na serveru SourceForge  
<https://sourceforge.net/projects/xxdiff/>
25. Stránka nástroje KDiff3  
<http://kdiff3.sourceforge.net/>
26. Seriál o programovacím jazyku TCL a GUI knihovně Tk  
<https://www.root.cz/serialy/programovaci-jazyk-tcl/>
27. ActiveTcl  
<https://www.activestate.com/activetcl>
28. Tiobe: žebříček popularity programovacích jazyků  
<https://www.tiobe.com/tiobe-index/>

---

**Root.cz** ([www.root.cz](http://www.root.cz))

Informace nejen ze světa Linuxu. ISSN 1212-8309

Copyright © 1998 – 2018 [Internet Info, s.r.o.](http://www.root.cz) Všechna práva vyhrazena. Powered by [Linux](http://www.root.cz).