https://www.tutorialspoint.com/git/index.htm

A Git egy elosztott verzió kezelő és forráskód-kezelő rendszer, amely a hangsúlyt a sebességre helyezi. A Git-et eredetileg Linus Torvalds tervezte és fejlesztette a Linux kernel fejlesztésére. A Git egy ingyenes szoftver, amelyet a GNU General Public License 2. verziójának feltételei szerint terjesztenek

Git - Basic Concepts

Verziókezelő rendszer

A Version Control System (VCS) olyan szoftver, amely segíti a szoftverfejlesztőket az együttműködésben és a munkájuk teljes előzményeinek megőrzésében.

Az alábbiakban felsoroljuk a VCS funkcióit -

- Lehetővé teszi a fejlesztők számára, hogy egyszerre dolgozzanak.
- Nem teszi lehetővé egymás változásainak felülírását.
- Minden verzió előzményeit megőrzi.

Az alábbiakban bemutatjuk a VCS típusait -

- Központosított verziókezelő rendszer (CVCS).
- Elosztott / decentralizált verziókezelő rendszer (DVCS).

Ebben a fejezetben csak az elosztott verzióvezérlő rendszerre és különösen a Gitre fogunk koncentrálni. A Git az elosztott verziókezelő rendszer alá tartozik.

Elosztott verziókezelő rendszer

A központosított verziókezelő rendszer (CVCS) egy központi szervert használ az összes fájl tárolására, és lehetővé teszi a csapat együttműködését. De a CVCS fő hátránya az egyetlen hibapont, azaz a központi szerver meghibásodása. Sajnos, ha a központi szerver leáll egy órára, akkor ez alatt az óra alatt senki sem tud együttműködni. És még a legrosszabb esetben is, ha a központi szerver lemeze megsérül és a megfelelő biztonsági másolat nem készült, akkor elveszíti a projekt teljes történetét. Itt az elosztott verziókezelő rendszer (DVCS) kerül képbe.

A DVCS kliensek nem csak a könyvtár legfrissebb pillanatképét nézik meg, hanem azt is teljes mértékben tükrözik. Ha a szerver leáll, akkor a kliens adattára visszaállítható a szerverre. A Git nem támaszkodik a központi szerverre, és ezért sok műveletet hajthat végre offline állapotban. Offline állapotban változtatásokat hajthat végre, ágakat hozhat létre, naplókat tekinthet meg és egyéb műveleteket hajthat végre. Csak a hálózati kapcsolatra van szüksége a módosítások közzétételéhez és a legújabb módosítások végrehajtásához.

A Git előnyei

Ingyenes és nyílt forráskódú

A Git a GPL nyílt forráskódú licencével jelenik meg. Szabadon elérhető az interneten keresztül. A Git segítségével egyetlen projekt fizetése nélkül is kezelheti projektjeit. Mivel nyílt forráskódról van szó, letöltheti a forráskódot, és az igényeinek megfelelően is végezhet módosításokat.

Gyors és kicsi

Mivel a műveletek nagy részét helyben hajtják végre, ez a sebesség szempontjából óriási előnyt jelent. A Git nem támaszkodik a központi szerverre; ezért nem szükséges minden műveletnél kölcsönhatásba lépni a távoli szerverrel. A Git központi része C-ben van írva, amely elkerüli a más magas szintű nyelvekhez kapcsolódó futásidejű általános költségeket. Bár a Git a teljes adattárat tükrözi, a kliens oldalon az adatok mérete kicsi. Ez szemlélteti a Git hatékonyságát az adatok tömörítésében és tárolásában a kliens oldalon.

Implicit mentés

Az adatok elvesztésének esélye nagyon ritka, ha több példány van belőle. Bármely kliens oldalon található adatok tükrözik az adattárat, ezért összeomlás vagy lemezsérülés esetén használhatók fel.

Biztonság

A Git egy közös titkosítási hash funkciót, az úgynevezett biztonságos hash függvényt (secure hash function) (SHA1) használ az objektumok megnevezésére és azonosítására az adatbázisában. Minden fájlt és commit check-sum-olva van. Ez azt jelenti, hogy lehetetlen megváltoztatni a fájlt, a dátumot, az üzenetet és az egyéb adatokat a Git adatbázisból megváltoztatni a Git ismerete nélkül.

Nincs szükség erős hardverre

CVCS esetén a központi szervernek elég erősnek kell lennie ahhoz, hogy az egész csapat kéréseit kiszolgálja. Kisebb csapatok számára ez nem kérdés, de a csapat méretének növekedésével a kiszolgáló hardveres korlátai teljesítményszűkületet jelenthetnek. DVCS esetén a fejlesztők csak akkor lépnek kapcsolatba a szerverrel, ha változtatásokat kell végrehajtaniuk.

Könnyebb branching

A CVCS olcsó másolási mechanizmust használ. Ha új branch-et hozunk létre, az összes kódot átmásolja az új bracnhbe, így időigényes és nem hatékony. Ezenkívül a branch-ek törlése és összevonása bonyolult és időigényes. De a branch kezelése a Gittel nagyon egyszerű. Az ágak (branch) létrehozása, törlése és egyesítése csak néhány másodpercet vesz igénybe.

DVCS terminológiák

Helyi adattár

Minden VCS eszköz egy magán munkahelyet biztosít munkapéldányként. A fejlesztők változtatásokat hajtanak végre a magán munkahelyükön, és kommitjuk után ezek a változtatások a tárház részévé válnak. A Git egy lépéssel tovább lép azáltal, hogy a teljes tárház privát példányát megadja nekik. A felhasználók számos műveletet végezhetnek ezzel az adattárral, például fájl hozzáadása, fájl eltávolítása, fájl átnevezése, fájl áthelyezése, változtatások végrehajtása és még sok más.

Munkahelyi könyvtár és átmeneti terület vagy index

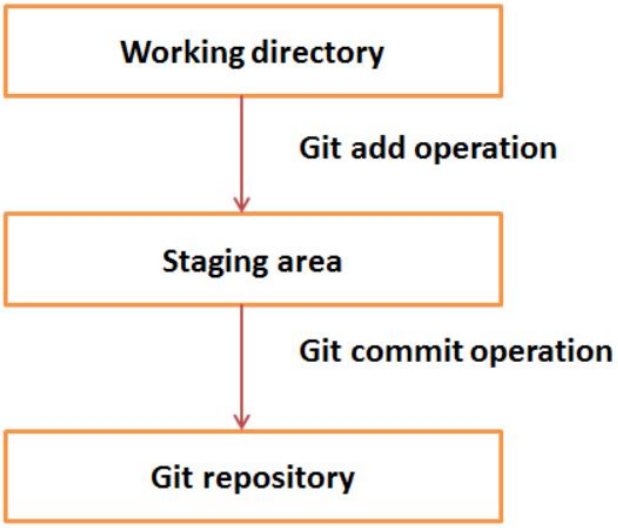
A munkakönyvtár az a hely, ahol a fájlokat ellenőrzik. Más CVCS-ben a fejlesztők általában módosításokat hajtanak végre, és a változtatásokat közvetlenül az adattárban végzik el. De Git más stratégiát alkalmaz. A Git nem követi az egyes módosított fájlokat. Valahányszor műveletet hajt végre, Git megkeresi az átmeneti területen lévő fájlokat. Csak az átmeneti területen található fájlokat vesszük figyelembe kommit-kor, és nem az összes módosított fájlt.

Nézzük meg a Git alapvető munkafolyamatát.

1. lépés - Módosít egy fájlt a munkakönyvtárból.

2. lépés - Ezeket a fájlokat hozzáadja az átmeneti területhez. (staging area)

3. lépés - Végrehajtási műveletet hajt végre, amely áthelyezi a fájlokat az átmeneti területről (staging area). Push művelet után a változásokat véglegesen tárolja a Git adattárban.



Tegyük fel, hogy két fájlt módosított, nevezetesen a „sort.c” és a „search.c” fájlokat, és minden művelethez két különböző commit-ot szeretne. Felvehet egy fájlt az átmeneti területre (staging area), és elvégezheti a commit-ot. Az első kommit után ismételje meg ugyanezt az eljárást egy másik fájl esetében.

```
# First commit
[bash]$ git add sort.c

# adds file to the staging area
[bash]$ git commit –m “Added sort operation”

# Second commit
[bash]$ git add search.c

# adds file to the staging area
[bash]$ git commit –m “Added search operation”
```

Blobs

A Blob a bináris nagy objektumot jelenti. A fájl minden verzióját blob ábrázolja. A blob tárolja a fájl adatait, de nem tartalmaz a fájl metaadatait. Ez egy bináris fájl, és a Git adatbázisban a fájl SHA1 kivonatának nevezik.

Trees

A Tree egy objektum, amely egy könyvtárat képvisel. Blobokat és más alkönyvtárakat is tartalmaz. A fa egy bináris fájl, amely blobokra és fákra hivatkozásokat tárol, amelyeket a fa objektum SHA1 kivonatának is neveznek.

Commit

A Commit megőrzi a tárház aktuális állapotát. Az commit az SHA1 hash is kap. A commit objektumot a láncolt listacsomópontjának tekintheti. Minden commit objektumnak van mutatója a szülő commit objektumra. Egy adott commit-ből visszaléphet, ha megnézi a szülőmutatót a commit előzményeinek megtekintéséhez. Ha egy commit-nak több szülői commit-ja van, akkor az adott commit két ág egyesítésével jött létre.

Branches

Az ágakat (branches) egy újabb fejlesztési vonal létrehozására használják. Alapértelmezés szerint a Git rendelkezik egy főággal, amely megegyezik a Subversion törzsével. Általában egy ágat hoznak létre egy új funkció kezelésére. Miután a szolgáltatás elkészült, merg-elik a master ágba, és töröljük az ágat. Minden ágra a HEAD hivatkozik, amely a fiók legutóbbi kommit-jára mutat. Amikor kommitol, a HEAD frissül a legújabb kommittal.

Tag

A Tag egy értelmes nevet rendel hozzá egy adott verzióhoz a reposity-ban. A tag-el nagyon hasonlítanak a branch-ekhez, de a különbség az, hogy megváltoztathatatlanok. Ez azt jelenti, hogy a tag egy branch, amelyet senki sem szándékozik módosítani. Miután létrehozott egy tag-et egy adott commit.hoz, még akkor sem, ha új commit-ot hoz létre, az nem frissül. A fejlesztők általában tag-eket hoznak létre a termékkiadásokhoz.

Clone

A clone művelet létrehozza az repository példányát. A clone-ozás nem csak a munkamásolatot ellenőrzi, hanem tükrözi a teljes repository-t is. A felhasználók sok műveletet hajthatnak végre ezzel a helyi repository-val. A hálózati átvitel csak akkor történik meg, amikor a repositorgy példányait szinkronizálják.

Pull

A pull művelet másolja a változásokat egy távoli repository példányból egy lokálisra. A pull műveletet két repository-példány közötti szinkronizáláshoz használják. Ez megegyezik a Subversion frissítési műveletével.

Push

A push művelet másolja a változásokat egy helyi repository példányból egy távoli példányba. Ez a változások végleges tárolására szolgál a Git repository-ban. Ez megegyezik a Subversion commit műveletével.

HEAD

A HEAD pointer, amely mindig az ág legutóbb commit-jára mutat. Amikor commit-ol, a HEAD frissül a legújabb commit-tal. Az branch-ek head-jei a .git / refs / heads / könyvtárban vannak tárolva.

```
[CentOS]$ ls -1 .git/refs/heads/
master

[CentOS]$ cat .git/refs/heads/master
570837e7d58fa4bccd86cb575d884502188b0c49
```

Revision

A Revision a forráskód verzióját jelenti. A Gitben a revision-t a commit jelenti. Ezeket a végrehajtásokat az SHA1 biztonságos hashek azonosítják.

URL

Az URL a Git adattár helyét jelöli. A Git URL a konfigurációs fájlban van tárolva.

```
[tom@CentOS tom_repo]$ pwd
/home/tom/tom_repo

[tom@CentOS tom_repo]$ cat .git/config
[core]
repositoryformatversion = 0
filemode = true
bare = false
logallrefupdates = true
[remote "origin"]
url = gituser@git.server.com:project.git
fetch = +refs/heads/*:refs/remotes/origin/*
```

Git - Environment Setup

A Git Environment testreszabása

A Git biztosítja a git config eszközt, amely lehetővé teszi konfigurációs változók beállítását. A Git az összes globális konfigurációt a home könyvtárban található .gitconfig fájlban tárolja. Ha ezeket a konfigurációs értékeket globálisra szeretné állítani, adja hozzá a --global beállítást, és ha kihagyja a --global beállítást, akkor a konfigurációi az aktuális Git-repository-ra vonatkoznak.

Beállíthat rendszerszintű konfigurációt is. A Git ezeket az értékeket az / etc / gitconfig fájlban tárolja, amely tartalmazza a rendszer minden felhasználójának és repository-nak konfigurációját. Ezen értékek beállításához rendelkeznie kell a root jogosultságokkal, és a --system kapcsolót kell használnia.

A fenti kód összeállításakor és végrehajtásakor a következő eredményt kapja -

Felhasználónév beállítása

Ezt az információt Git használja minden egyes commit-hoz.

```
[jerry@CentOS project]$ git config --global user.name "Jerry Mouse"
```

E-mail azonosító beállítása

Ezt az információt Git használja minden egyes commit-hoz.

```
[jerry@CentOS project]$ git config --global user.email "jerry@tutorialspoint.com"
```

Avoid merge commits for pulling

A legfrissebb módosításokat egy távoli repository-ból szerzi be, és ha ezek a változások eltérnek, akkor a Git alapértelmezés szerint merge commits-t hoz létre. Ezt a következő beállításokkal tudjuk elkerülni.

```
jerry@CentOS project]$ git config --global branch.autosetuprebase always
```

Színkiemelés

A következő parancsok lehetővé teszik a színkiemelést a Git számára a konzolon.

```
[jerry@CentOS project]$ git config --global color.ui true
[jerry@CentOS project]$ git config --global color.status auto
[jerry@CentOS project]$ git config --global color.branch auto
```

Alapértelmezett szerkesztő beállítása

Alapértelmezés szerint a Git a rendszer alapértelmezett szerkesztőjét használja, amely a VISUAL vagy az EDITOR környezeti változóból származik. Konfigurálhatunk egy mást a git config használatával.

```
[jerry@CentOS project]$ git config --global core.editor vim
```

Alapértelmezett merge eszköz beállítása

A Git nem nyújt alapértelmezett merge eszközt az ütköző változások integrálásához a working tree-be. Az alapértelmezett merge eszközt a következő beállítások engedélyezésével állíthatjuk be.

```
[jerry@CentOS project]$ git config --global merge.tool vimdiff
```

A Git beállításainak felsorolása

A helyi adattár Git-beállításainak ellenőrzéséhez használja a git config –list parancsot az alábbiak szerint.

```
[jerry@CentOS ~]$ git config --list
```

Eredmény:

```
user.name=Jerry Mouse
user.email=jerry@tutorialspoint.com
push.default=nothing
branch.autosetuprebase=always
color.ui=true
color.status=auto
color.branch=auto
core.editor=vim
merge.tool=vimdiff
```

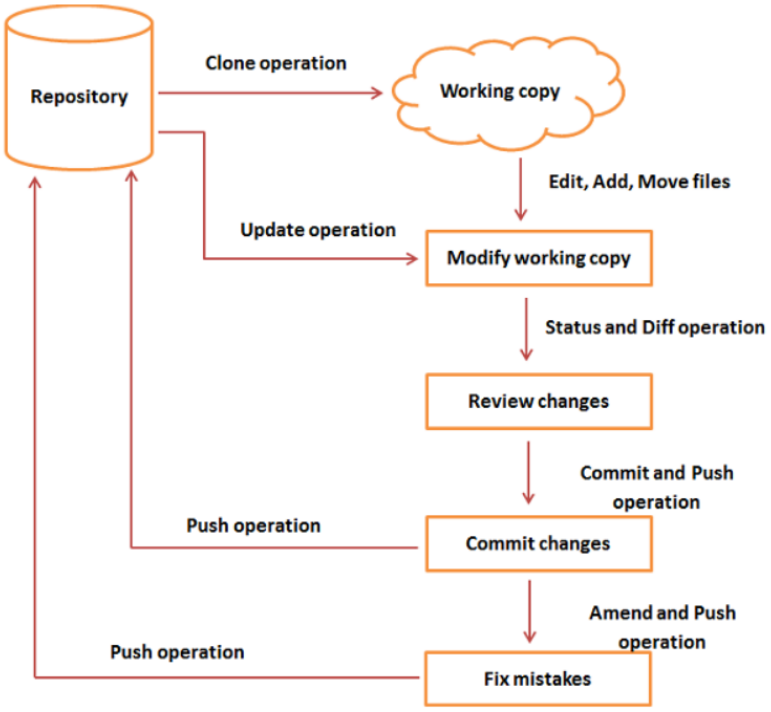
Git - Life Cycle

Ebben a fejezetben a Git életciklusát tárgyaljuk. A későbbi fejezetekben az egyes műveletek Git parancsait ismertetjük.

Az általános munkafolyamat a következő -

- Munkamásolatként (working copy) klónozza a Git repository-t.
- A munkamásolatot (working copy) fájlok hozzáadásával / szerkesztésével módosíthatja.
- Szükség esetén frissíti a munkapéldányt (woring copy) más fejlesztői változtatások végrehajtásával.
- Átnézi a változásokat, mielőtt commit-olna .
- Commitol-ja a változásokat. Ha minden rendben van, akkor a módosításokat push-olja a repository-ba.
- A commit után, ha észreveszi, hogy valami nincs rendben, akkor kijavítja az utolsó commit-ot, és push-olhatja a módosításokat a repository-ba.

Az alábbiakban látható a munkafolyamat képi ábrázolása.



Git - Create Operation

Ebben a fejezetben megtudjuk, hogyan lehet távoli Git-repository-t létrehozni; ezentúl Git Server néven fogjuk emlegetni. Szükségünk van egy Git szerverre a csapat együttműködésének lehetővé tételéhez.

Új felhasználó létrehozása

```
# add new group
[root@CentOS ~]# groupadd dev

# add new user
[root@CentOS ~]# useradd -G devs -d /home/gituser -m -s /bin/bash gituser

# change password
[root@CentOS ~]# passwd gituser
```

A fenti parancs a következő eredményt adja.

```
Changing password for user gituser.
New password:
Retype new password:
passwd: all authentication token updated successfully.
```

Hozzon létre egy üres repository-t

Inicializáljunk egy új repository-t az init paranccsal, majd a --bare opcióval. Munkakönyvtár nélkül inicializálja az adattárat. Megállapodás szerint az üres tárházat .git néven kell megnevezni.

```
[gituser@CentOS ~]$ pwd
/home/gituser

[gituser@CentOS ~]$ mkdir project.git

[gituser@CentOS ~]$ cd project.git/

[gituser@CentOS project.git]$ ls

[gituser@CentOS project.git]$ git --bare init
Initialized empty Git repository in /home/gituser-m/project.git/

[gituser@CentOS project.git]$ ls
branches config description HEAD hooks info objects refs
```

Generate Public/Private RSA Key Pair

Járjuk végig a Git szerver konfigurálásának folyamatát, az ssh-keygen segédprogram előállítja a nyilvános / privát RSA kulcspárokat, amelyeket felhasználunk a felhasználói hitelesítéshez.

Nyisson meg egy terminált, írja be a következő parancsot, és csak nyomja meg az Enter billentyűt minden bemenetnél. A sikeres befejezés után .ssh könyvtárat hoz létre a saját könyvtárban.

```
tom@CentOS ~]$ pwd
/home/tom

[tom@CentOS ~]$ ssh-keygen
```

A fenti parancs a következő eredményt adja.

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/tom/.ssh/id_rsa): Press Enter Only
Created directory '/home/tom/.ssh'.
Enter passphrase (empty for no passphrase): ----------------> Press Enter Only
Enter same passphrase again: -----------------------------> Press Enter Only
Your identification has been saved in /home/tom/.ssh/id_rsa.
Your public key has been saved in /home/tom/.ssh/id_rsa.pub.
The key fingerprint is:
df:93:8c:a1:b8:b7:67:69:3a:1f:65:e8:0e:e9:25:a1 tom@CentOS
The key's randomart image is:
+--[ RSA 2048]----+
| |
| |
| |
|
.
|
| Soo |
| o*B. |
| E = *.= |
| oo==. . |
| ..+Oo
|
+-----------------+
```

Az ssh-keygen két kulcsot generált, az egyik privát (azaz id_rsa), a második nyilvános (azaz id_rsa.pub).

Megjegyzés: Soha ne ossza meg másokkal a PRIVÁT KULCSOT.

Kulcsok hozzáadása az authorized_keys-hez

Tegyük fel, hogy két fejlesztő dolgozik egy projekten, nevezetesen Tom és Jerry. Mindkét felhasználó létrehozott nyilvános kulcsokat. Nézzük meg, hogyan használhatjuk ezeket a kulcsokat a hitelesítéshez.

Tom hozzáadta nyilvános kulcsát a szerverhez az alábbiakban megadott ssh-copy-id paranccsal –

```
[tom@CentOS ~]$ pwd
/home/tom

[tom@CentOS ~]$ ssh-copy-id -i ~/.ssh/id_rsa.pub gituser@git.server.com
```

Az alábbi eredményt kapjuk:

```
gituser@git.server.com's password:
Now try logging into the machine, with "ssh 'gituser@git.server.com'", and check in:
.ssh/authorized_keys
to make sure we haven't added extra keys that you weren't expecting.
```

Hasonlóan, jerry is hozzáadta nyilvános kulcsát a szerverhez az ssh-copy-id paranccsal.

```
[jerry@CentOS ~]$ pwd
/home/jerry

[jerry@CentOS ~]$ ssh-copy-id -i ~/.ssh/id_rsa gituser@git.server.com
```

A fenti parancs a következő eredményt adja.

```
gituser@git.server.com's password:
Now try logging into the machine, with "ssh 'gituser@git.server.com'", and check in:
.ssh/authorized_keys
to make sure we haven't added extra keys that you weren't expecting.
```

Push Changes to the Repository

Létrehoztunk egy üres repository-t a szerveren, és két felhasználó számára engedélyeztük a hozzáférést. Mostantól Tom és Jerry a repository-ba push-olhatja a változásokat.

A Git init parancs létrehoz egy .git könyvtárat, amely tárolja a repository metaadatait, valahányszor olvassa a konfigurációt a .git / config fájlból.

Tom létrehoz egy új könyvtárat, felveszi a README fájlt, és változtatását kezdeti kommittal hajtja végre. Commit után a git log parancs futtatásával ellenőrzi a commit üzenetet.

```
[tom@CentOS ~]$ pwd
/home/tom

[tom@CentOS ~]$ mkdir tom_repo

[tom@CentOS ~]$ cd tom_repo/

[tom@CentOS tom_repo]$ git init
Initialized empty Git repository in /home/tom/tom_repo/.git/

[tom@CentOS tom_repo]$ echo 'TODO: Add contents for README' > README

[tom@CentOS tom_repo]$ git status -s
?? README

[tom@CentOS tom_repo]$ git add .

[tom@CentOS tom_repo]$ git status -s
A README

[tom@CentOS tom_repo]$ git commit -m 'Initial commit'
```

A fenti parancs a következő eredményt adja.

```
[master (root-commit) 19ae206] Initial commit
1 files changed, 1 insertions(+), 0 deletions(-)
create mode 100644 README
```

Tom a git log parancs végrehajtásával ellenőrzi a log üzenetet.

```
[tom@CentOS tom_repo]$ git log
```

A fenti parancs a következő eredményt adja.

```
commit 19ae20683fc460db7d127cf201a1429523b0e319
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 07:32:56 2013 +0530

Initial commit
```

Tom commit-olta változtatásait a helyi repository-n. Itt az ideje, hogy áttegye a módosításokat a távoli repository-ba. De előtte hozzá kell adnunk a repository-t remote-ként, ez egy egyszeri művelet. Ezek után biztonságosan átküldheti a változásokat a távoli repository-ba.

Megjegyzés - Alapértelmezés szerint a Git csak az illeszkedő branch-ekre push-ol: A helyi oldalon található minden branch esetében a távoli oldal frissül, ha már létezik azonos nevű branch. Oktatóanyagainkban minden alkalommal, amikor változtatásokat hajtunk végre az origin master branch-en, használjon megfelelő branch nevet az Ön igényeinek megfelelően.

```
[tom@CentOS tom_repo]$ git remote add origin gituser@git.server.com:project.git

[tom@CentOS tom_repo]$ git push origin master
```

A fenti parancs a következő eredményt adja.

```
Counting objects: 3, done.
Writing objects: 100% (3/3), 242 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To gituser@git.server.com:project.git
* [new branch]
master -> master
```

Most a változtatások sikeresen végrehajtásra kerültek a távoli repository-ban.

Git - Clone Operation

Üres repository-nk van a Git szerveren, és Tom az első verzióját is push-olta. Jerry most megtekintheti a változásokat. A Clone művelet létrehozza a távoli repository egy példányát.

Jerry létrehoz egy új könyvtárat a saját könyvtárában, és végrehajtja a clone műveletet.

```
[jerry@CentOS ~]$ mkdir jerry_repo

[jerry@CentOS ~]$ cd jerry_repo/

[jerry@CentOS jerry_repo]$ git clone gituser@git.server.com:project.git
```

A fenti parancs a következő eredményt adja.

```
Initialized empty Git repository in /home/jerry/jerry_repo/project/.git/
remote: Counting objects: 3, done.
Receiving objects: 100% (3/3), 241 bytes, done.
remote: Total 3 (delta 0), reused 0 (delta 0)
```

Git - Perform Changes

Jerry klónozza a tárat, és úgy dönt, hogy végrehajtja az alapvető karakterlánc-műveleteket. Így létrehozza a string.c fájlt. A tartalom hozzáadása után a string.c a következőképpen fog kinézni –

```c
#include <stdio.h>

int my_strlen(char *s)
{
   char *p = s;

   while (*p)
      ++p;

   return (p - s);
}

int main(void)
{
   int i;
   char *s[] =
   {
      "Git tutorials",
      "Tutorials Point"
   };

   for (i = 0; i < 2; ++i)

   printf("string lenght of %s = %d\n", s[i], my_strlen(s[i]));

   return 0;
}
```

Tesztelte a kódját, és minden rendben működik. Most biztonságosan hozzáadhatja ezeket a változásokat a repository-ba.

A Git add művelet hozzáadja a fájlt az átmeneti területhez (staging area).

```
[jerry@CentOS project]$ git status -s
?? string
?? string.c

[jerry@CentOS project]$ git add string.c
```

Git kérdőjelet mutat a fájlnevek előtt. Nyilvánvaló, hogy ezek a fájlok nem részei a Git-nek, és ezért a Git nem tudja, mit kezdjen ezekkel a fájlokkal. Ezért Git kérdőjelet mutat a fájlnevek előtt.

Jerry hozzáadta a fájlt az átmeneti területhez, a git status parancs megmutatja az átmeneti területen (staging area) lévő fájlokat.

```
[jerry@CentOS project]$ git status -s
A string.c
?? string
```

A változtatások végrehajtásához használta a git commit parancsot, amelyet az –m opció követett. Ha kihagyjuk az –m opciót. A Git megnyit egy szövegszerkesztőt, ahol többsoros commit üzenetet írhatunk.

```
[jerry@CentOS project]$ git commit -m 'Implemented my_strlen function'
```

A fenti parancs a következő eredményt adja –

```
[master cbe1249] Implemented my_strlen function
1 files changed, 24 insertions(+), 0 deletions(-)
create mode 100644 string.c
```

Miután commitolt, a napló részleteinek megtekintésére, futtassa a git log parancsot. Megjeleníti az összes commit adatait a commit ID-vel, a commit szerzőjével, a commit dátumával és az SHA-1 commit hash-el.

```
[jerry@CentOS project]$ git log
```

A fenti parancs a következő eredményt adja –

```
commit cbe1249b140dad24b2c35b15cc7e26a6f02d2277
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Implemented my_strlen function


commit 19ae20683fc460db7d127cf201a1429523b0e319
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 07:32:56 2013 +0530

Initial commit
```

Git - Review Changes

A commit részleteinek megtekintése után Jerry rájön, hogy a karakterlánc hossza nem lehet negatív, ezért úgy dönt, hogy megváltoztatja a my_strlen függvény visszatérési típusát.

Jerry a git log paranccsal megtekintheti a napló részleteit.

```
[jerry@CentOS project]$ git log
```

A fenti parancs a következő eredményt adja.

```
commit cbe1249b140dad24b2c35b15cc7e26a6f02d2277
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Implemented my_strlen function
```

Jerry a git show paranccsal megtekintheti a commit részleteit. A git show parancs az SHA-1 commit azonosítóját veszi paraméterként.

```
[jerry@CentOS project]$ git show cbe1249b140dad24b2c35b15cc7e26a6f02d2277
```

A fenti parancs a következő eredményt adja –

```
commit cbe1249b140dad24b2c35b15cc7e26a6f02d2277
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Implemented my_strlen function


diff --git a/string.c b/string.c
new file mode 100644
index 0000000..187afb9
--- /dev/null
```

```
+++ b/string.c
@@ -0,0 +1,24 @@
+#include <stdio.h>
+
+int my_strlen(char *s)
+{
   +
   char *p = s;
   +
   +
   while (*p)
   + ++p;
   + return (p -s );
   +
}
+
```

Megváltoztatja a függvény visszatérési típusát int-ről size_t-re. A kód tesztelése után a git diff parancs futtatásával áttekinti változásait.

```
[jerry@CentOS project]$ git diff
```

A fenti parancs a következő eredményt adja –

```
diff --git a/string.c b/string.c
index 187afb9..7da2992 100644
--- a/string.c
+++ b/string.c
@@ -1,6 +1,6 @@
#include <stdio.h>

-int my_strlen(char *s)
+size_t my_strlen(char *s)
{
   char *p = s;
   @@ -18,7 +18,7 @@ int main(void)
};
for (i = 0; i < 2; ++i)
{
   - printf("string lenght of %s = %d\n", s[i], my_strlen(s[i]));
   + printf("string lenght of %s = %lu\n", s[i], my_strlen(s[i]));
   return 0;
}
```

A Git diff a + jelet mutatja a sorok előtt, amelyek újonnan lettek hozzáadva, és a - a törölt sorokhoz.

Git - Commit Changes

Jerry már végrehajtotta a változásokat, és ki akarja javítani utolsó commit-ot. Ebben az esetben a git módosítási művelet segít. A módosító művelet megváltoztatja az utolsó commit-ot, beleértve az Ön commit üzenetét is; új commit-azonosítót hoz létre.

A művelet módosítása előtt ellenőrzi a commit log-ot.

```
[jerry@CentOS project]$ git log
```

A fenti parancs a következő eredményt adja.

```
commit cbe1249b140dad24b2c35b15cc7e26a6f02d2277
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Implemented my_strlen function


commit 19ae20683fc460db7d127cf201a1429523b0e319
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 07:32:56 2013 +0530

Initial commit
```

Jerry a commit művelettel hajtja végre az új módosításokat, és megtekinti a commit log-ot.

```
[jerry@CentOS project]$ git status -s
M string.c
?? string

[jerry@CentOS project]$ git add string.c

[jerry@CentOS project]$ git status -s
M string.c
?? string

[jerry@CentOS project]$ git commit --amend -m 'Changed return type of my_strlen to size_t'
[master d1e19d3] Changed return type of my_strlen to size_t
1 files changed, 24 insertions(+), 0 deletions(-)
create mode 100644 string.c
```

Most a git log új commit üzenetet jelenít meg új commit azonosítóval –

```
[jerry@CentOS project]$ git log
```

A fenti parancs a következő eredményt adja.

```
commit d1e19d316224cddc437e3ed34ec3c931ad803958
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Changed return type of my_strlen to size_t


commit 19ae20683fc460db7d127cf201a1429523b0e319
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 07:32:56 2013 +0530

Initial commit
```

Git - Push Operation

Jerry módosította utolsó commit-ot, és készen áll a változtatások végrehajtására. A Push művelet véglegesen push-olja az adatokat a Git repository-ba. A sikeres push művelet után más fejlesztők láthatják Jerry változásait.

A végrehajtás részleteinek megtekintéséhez a git log parancs:

```
[jerry@CentOS project]$ git log
```

A fenti parancs a következő eredményt adja:

```
commit d1e19d316224cddc437e3ed34ec3c931ad803958
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Changed return type of my_strlen to size_t
```

A push művelet előtt át akarja tekinteni a változásait, ezért a git show paranccsal áttekinti változását.

```
[jerry@CentOS project]$ git show d1e19d316224cddc437e3ed34ec3c931ad803958
```

A fenti parancs a következő eredményt adja:

```
commit d1e19d316224cddc437e3ed34ec3c931ad803958
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Changed return type of my_strlen to size_t

diff --git a/string.c b/string.c
new file mode 100644
index 0000000..7da2992
--- /dev/null
+++ b/string.c
@@ -0,0 +1,24 @@
+#include <stdio.h>
+
+size_t my_strlen(char *s)
+
{
   +
   char *p = s;
   +
   +
   while (*p)
   + ++p;
   + return (p -s );
   +
}
+
+int main(void)
+
{
   + int i;
   + char *s[] =
   {
      + "Git tutorials",
      + "Tutorials Point"
      +
   };
   +
   +
   +
   for (i = 0; i < 2; ++i)
   printf("string lenght of %s = %lu\n", s[i], my_strlen(s[i]));
   +
   +
   return 0;
   +
}
```

Jerry örül a változásainak, és készen áll a változások push-olására.

```
[jerry@CentOS project]$ git push origin master
```

Eredmény:

```
Counting objects: 4, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 517 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To gituser@git.server.com:project.git
19ae206..d1e19d3 master -> master
```

Git - Update Operation

Meglévő funkció módosítása

Tom elvégzi a klónozási műveletet, és új fájlt talál a string.c fájlban. Azt akarja tudni, hogy ki és milyen célból adta hozzá ezt a fájlt a repository-hoz, ezért végrehajtja a git log parancsot.

```
[tom@CentOS ~]$ git clone gituser@git.server.com:project.git
```

A fenti parancs a következő eredményt adja –

```
Initialized empty Git repository in /home/tom/project/.git/
remote: Counting objects: 6, done.
remote: Compressing objects: 100% (4/4), done.
Receiving objects: 100% (6/6), 726 bytes, done.
remote: Total 6 (delta 0), reused 0 (delta 0)
```

A klónozás új könyvtárat hoz létre a jelenlegi munkakönyvtárban. Megváltoztatja a könyvtárat újonnan létrehozott könyvtárra, és végrehajtja a git log parancsot.

```
[tom@CentOS ~]$ cd project/

[tom@CentOS project]$ git log
```

A fenti parancs a következő eredményt adja –

```
commit d1e19d316224cddc437e3ed34ec3c931ad803958
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Changed return type of my_strlen to size_t


commit 19ae20683fc460db7d127cf201a1429523b0e319
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 07:32:56 2013 +0530

Initial commit
```

A napló megfigyelése után rájön, hogy a string.c fájlt Jerry adta hozzá az alapvető karakterlánc-műveletek végrehajtásához. Kíváncsi Jerry kódjára. Tehát megnyitja a string.c-t a szövegszerkesztőben, és azonnal hibát talál. A my_strlen függvényben Jerry nem használ állandó mutatót. Tehát úgy dönt, hogy módosítja Jerry kódját. Módosítás után a kód a következőképpen néz ki –

```
[tom@CentOS project]$ git diff
```

A fenti parancs a következő eredményt adja -

```
diff --git a/string.c b/string.c
index 7da2992..32489eb 100644
--- a/string.c
+++ b/string.c
@@ -1,8 +1,8 @@
#include <stdio.h>
-size_t my_strlen(char *s)
+size_t my_strlen(const char *s)
{
   - char *p = s;
   + const char *p = s;
   while (*p)
   ++p;
}
```

Tesztelés után commit-olja változását.

```
[tom@CentOS project]$ git status -s
M string.c
?? string

[tom@CentOS project]$ git add string.c

[tom@CentOS project]$ git commit -m 'Changed char pointer to const char pointer'
[master cea2c00] Changed char pointer to const char pointer
1 files changed, 2 insertions(+), 2 deletions(-)

[tom@CentOS project]$ git log
```

A fenti parancs a következő eredményt adja –

```
commit cea2c000f53ba99508c5959e3e12fff493b
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 08:32:07 2013 +0530

Changed char pointer to const char pointer


commit d1e19d316224cddc437e3ed34ec3c931ad803958
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Changed return type of my_strlen to size_t


commit 19ae20683fc460db7d127cf201a1429523b0e319
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 07:32:56 2013 +0530
Initial commit
```

Tom a git push parancsot használja a változásainak elérésére.

```
[tom@CentOS project]$ git push origin master
```

A fenti parancs a következő eredményt adja –

```
Counting objects: 5, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 336 bytes, done.
Total 3 (delta 1), reused 0 (delta 0)
To gituser@git.server.com:project.git
d1e19d3..cea2c00 master -> master
```

Új függvény hozzáadása

Eközben Jerry úgy dönt, hogy végrehajtja a karakterlánc-összehasonlító funkciókat. Tehát módosítja a stringet.c. Módosítás után a fájl a következőképpen néz ki –

```
[jerry@CentOS project]$ git diff
```

A fenti parancs a következő eredményt adja –

```
index 7da2992..bc864ed 100644
--- a/string.c
+++ b/string.c
30Git Tutorials
@@ -9,9 +9,20 @@ size_t my_strlen(char *s)
return (p -s );
}
+char *my_strcpy(char *t, char *s)
+
{
   +
   char *p = t;
   +
   + while (*t++ = *s++)
   + ;
   +
   +
   return p;
   +
}
+
int main(void)
{
   int i;
   +
   char p1[32];
   char *s[] =
   {
      "Git tutorials",
      "Tutorials Point"
      @@ -20,5 +31,7 @@ int main(void)
      for (i = 0; i < 2; ++i)
      printf("string lenght of %s = %lu\n", s[i], my_strlen(s[i]));
      +
      printf("%s\n", my_strcpy(p1, "Hello, World !!!"));
      +
      return 0;
   }
}
```

A tesztelés után készen áll a változásra.

```
[jerry@CentOS project]$ git status -s
M string.c
?? string

[jerry@CentOS project]$ git add string.c

[jerry@CentOS project]$ git commit -m "Added my_strcpy function"
[master e944e5a] Added my_strcpy function
1 files changed, 13 insertions(+), 0 deletions(-)
```

A push művelet előtt a log message-t megnézi.

```
[jerry@CentOS project]$ git log
```

A fenti parancs a következő eredményt adja –

```
commit e944e5aab74b26e7447d3281b225309e4e59efcd
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:41:42 2013 +0530

Added my_strcpy function


commit d1e19d316224cddc437e3ed34ec3c931ad803958
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Changed return type of my_strlen to size_t


commit 19ae20683fc460db7d127cf201a1429523b0e319
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 07:32:56 2013 +0530

Initial commit
```

Jerry örül a változásoknak, és push-olni szeretné azt.

```
[jerry@CentOS project]$ git push origin master
```

A fenti parancs a következő eredményt adja –

```
To gituser@git.server.com:project.git
! [rejected]
master -> master (non-fast-forward)
error: failed to push some refs to 'gituser@git.server.com:project.git'
To prevent you from losing history, non-fast-forward updates were rejected
Merge the remote changes before pushing again. See the 'Note about
fast-forwards' section of 'git push --help' for details.
```

De Git nem engedi Jerrynek, hogy push-olja változásait. Mivel Git azonosította, hogy a távoli repository és Jerry helyi repository-ja nincsenek szinkronban. Emiatt elveszítheti a project history-ját. Hogy elkerülje ezt a rendetlenséget, Git nem hajtja verge így a műveletet. Most Jerry-nek először frissítenie kell a helyi repository-t, és csak ezután push-olhatja a saját változtatásaira.

A legfrissebb változtatások letöltése

Jerry végrehajtja a git pull parancsot, hogy szinkronizálja a helyi repository-t a távoli repository-val.

```
[jerry@CentOS project]$ git pull
```

A fenti parancs a következő eredményt adja –

```
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From git.server.com:project
d1e19d3..cea2c00 master -> origin/master
First, rewinding head to replay your work on top of it...
Applying: Added my_strcpy function
```

A pull művelet után Jerry ellenőrzi a log üzeneteket, és megtalálja Tom commit-jának részleteit az alábbi ID-vel: cea2c000f53ba99508c5959e3e12fff493ba6f69

```
[jerry@CentOS project]$ git log
```

A fenti parancs a következő eredményt adja –

```
commit e86f0621c2a3f68190bba633a9fe6c57c94f8e4f
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:41:42 2013 +0530

Added my_strcpy function


commit cea2c000f53ba99508c5959e3e12fff493ba6f69
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 08:32:07 2013 +0530

Changed char pointer to const char pointer


commit d1e19d316224cddc437e3ed34ec3c931ad803958
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Changed return type of my_strlen to size_t


commit 19ae20683fc460db7d127cf201a1429523b0e319
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 07:32:56 2013 +0530
Initial commit
```

Most Jerry helyi repository-ja teljesen szinkronban van a távoli repository-val. Így nyugodtan push-olhatja a változásait.

```
[jerry@CentOS project]$ git push origin master
```

A fenti parancs a következő eredményt adja –

```
Counting objects: 5, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 455 bytes, done.
Total 3 (delta 1), reused 0 (delta 0)
To gituser@git.server.com:project.git
cea2c00..e86f062 master -> master
```

Git - Stash Operation

Tegyük fel, hogy új funkciót vezet be a termékéhez. A kódod folyamatban van, és hirtelen jön az ügyfelek eszkalációja. Emiatt néhány óráig félre kell tennie új funkcióját. Nem adhatja le a részleges kódot, és nem dobhatja el a módosításokat sem. Szüksége van tehát egy ideiglenes helyre, ahol tárolhatja részleges változtatásait, majd később commit-olhatjaazokat.

A Gitben a stash művelet végrehajtja a módosított nyomon követett fájlokat, szakaszokat állít be és ment el , amelyeket bármikor újra alkalmazhat.

```
[jerry@CentOS project]$ git status -s
M string.c
?? string
```

Most branch-et akar váltani az ügyfelek eszkalációja érdekében, de nem akarja commit-olni azt, amin még dolgozik; így el fogja rejteni a változásokat. Ha új stash-t szeretne push-olni a stack-be, amihez futtassa a git stash parancsot.

```
[jerry@CentOS project]$ git stash
Saved working directory and index state WIP on master: e86f062 Added my_strcpy function
HEAD is now at e86f062 Added my_strcpy function
```

Most a munkakönyvtárad tiszta, és az összes változtatást egy stack-be menti. Ellenőrizzük a git status paranccsal.

```
[jerry@CentOS project]$ git status -s
?? string
```

Most már biztonságosan kapcsolhatja a branch-et, és másutt dolgozhat. Az elrejtett (stashed) változtatások listáját a git stash list paranccsal tekinthetjük meg.

```
[jerry@CentOS project]$ git stash list
stash@{0}: WIP on master: e86f062 Added my_strcpy function
```

Tegyük fel, hogy megoldotta az ügyfelek eszkalációját, és visszatér az új funkcióhoz, és a félkész kódot keresi, csak hajtsa végre a git stash pop parancsot, hogy eltávolítsa a változásokat a stack-ből, és elhelyezze azokat az aktuális munkakönyvtárban.

```
[jerry@CentOS project]$ git status -s
?? string

[jerry@CentOS project]$ git stash pop
```

A fenti parancs a következő eredményt adja:

```
# On branch master
# Changed but not updated:
# (use "git add ..." to update what will be committed)
# (use "git checkout -- ..." to discard changes in working directory)
#
#
modified: string.c
#
# Untracked files:
# (use "git add ..." to include in what will be committed)
#
#
string
no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (36f79dfedae4ac20e2e8558830154bd6315e72d4)

[jerry@CentOS project]$ git status -s
M string.c
?? string
```

Git - Move Operation

Ahogy a neve is sugallja, a move művelet könyvtárat vagy fájlt helyez át egyik helyről a másikra. Tom úgy dönt, hogy áthelyezi a forráskódot az src könyvtárba. A módosított könyvtárszerkezet a következőképpen jelenik meg:

```
[tom@CentOS project]$ pwd
/home/tom/project

[tom@CentOS project]$ ls
README string string.c

[tom@CentOS project]$ mkdir src

[tom@CentOS project]$ git mv string.c src/

[tom@CentOS project]$ git status -s
R string.c -> src/string.c
?? string
```

Ahhoz, hogy ezek a változások véglegessé váljanak, a módosított könyvtárstruktúrát a távoli repository-ba kell push-olni, hogy ezt más fejlesztők is láthassák.

```
[tom@CentOS project]$ git commit -m "Modified directory structure"

[master 7d9ea97] Modified directory structure
1 files changed, 0 insertions(+), 0 deletions(-)
rename string.c => src/string.c (100%)

[tom@CentOS project]$ git push origin master
Counting objects: 4, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 320 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To gituser@git.server.com:project.git
e86f062..7d9ea97 master -> master
```

Jerry helyi repository-jában a pull művelet előtt megmutatja a régi könyvtárstruktúrát.

```
[jerry@CentOS project]$ pwd
/home/jerry/jerry_repo/project

[jerry@CentOS project]$ ls
README string string.c
```

De a pull művelet után a könyvtárstruktúra frissül. Most Jerry láthatja az src könyvtárat és a benne lévő fájlt.

```
[jerry@CentOS project]$ git pull
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From git.server.com:project
e86f062..7d9ea97 master -> origin/master
First, rewinding head to replay your work on top of it...
Fast-forwarded master to 7d9ea97683da90bcdb87c28ec9b4f64160673c8a.

[jerry@CentOS project]$ ls
```

```
README src string

[jerry@CentOS project]$ ls src/
string.c
```

Git - Rename Operation

Eddig mind Tom, mind Jerry manuális parancsokat használt a projekt összeállításához. Most Jerry úgy dönt, hogy létrehozza a Makefile-t a projektjükhöz, és megfelelő nevet ad a „string.c” fájlnak is.

```
[jerry@CentOS project]$ pwd
/home/jerry/jerry_repo/project

[jerry@CentOS project]$ ls
README src

[jerry@CentOS project]$ cd src/

[jerry@CentOS src]$ git add Makefile

[jerry@CentOS src]$ git mv string.c string_operations.c

[jerry@CentOS src]$ git status -s
A Makefile
R string.c -> string_operations.c
```

A Git R-t mutat a fájlnév előtt, jelezve, hogy a fájlt átnevezték.

A commit műveletéhez Jerry egy-flaget használt, amely a git commit automatikusan felismeri a módosított fájlokat.

```
[jerry@CentOS src]$ git commit -a -m 'Added Makefile and renamed strings.c to
string_operations.c '

[master 94f7b26] Added Makefile and renamed strings.c to string_operations.c
1 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 src/Makefile
rename src/{string.c => string_operations.c} (100%)
```

Commit után push-olja a változásokat a repository-ba.

```
[jerry@CentOS src]$ git push origin master
```

A fenti parancs a következő eredményt adja –

```
Counting objects: 6, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 396 bytes, done.
Total 4 (delta 0), reused 0 (delta 0)
To gituser@git.server.com:project.git
7d9ea97..94f7b26 master -> master
```

Git - Delete Operation

Tom frissíti a helyi repository-t, és megtalálja a lefordított bináris fájlt az src könyvtárban. A commit üzenet megtekintése után rájön, hogy az összeállított bináris dokumentumot Jerry adta hozzá.

```
[tom@CentOS src]$ pwd
/home/tom/project/src

[tom@CentOS src]$ ls
Makefile string_operations string_operations.c

[tom@CentOS src]$ file string_operations
string_operations: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked
(uses
shared libs), for GNU/Linux 2.6.18, not stripped

[tom@CentOS src]$ git log
commit 29af9d45947dc044e33d69b9141d8d2dad37cc62
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 10:16:25 2013 +0530

Added compiled binary
```

A VCS csak a forráskód és nem a futtatható bináris fájlok tárolására szolgál. Tehát Tom úgy dönt, hogy eltávolítja ezt a fájlt a tárból. A további műveletekhez a git rm parancsot használja.

```
[tom@CentOS src]$ ls
Makefile string_operations string_operations.c

[tom@CentOS src]$ git rm string_operations
rm 'src/string_operations'

[tom@CentOS src]$ git commit -a -m "Removed executable binary"

[master 5776472] Removed executable binary
1 files changed, 0 insertions(+), 0 deletions(-)
delete mode 100755 src/string_operations
```

A commit után push-olja a változásokat a repository-ba.

```
[tom@CentOS src]$ git push origin master
```

A fenti parancs a következő eredményt adja.

```
Counting objects: 5, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 310 bytes, done.
Total 3 (delta 1), reused 0 (delta 0)
To gituser@git.server.com:project.git
29af9d4..5776472 master -> master
```

Git - Fix Mistakes

Tévedni emberi. Tehát minden VCS olyan funkcióval rendelkezik, amely egy bizonyos pontig kijavítja a hibákat. A Git olyan funkcióval rendelkezik, amelyet felhasználva visszavonhatjuk a helyi adattárban végrehajtott módosításokat.

Tegyük fel, hogy a felhasználó véletlenül végrehajt néhány módosítást a helyi adattárban, majd vissza akarja vonni ezeket a módosításokat. Ilyen esetekben a visszavonási művelet fontos szerepet játszik.

Nem commit-tolt változtatások visszaállítása

Tegyük fel, hogy Jerry véletlenül módosít egy fájlt a helyi repository-ból. De vissza akarja vonni a módosítását. Ennek kezelésére használhatjuk a git checkout parancsot. Ezzel a paranccsal visszaállíthatjuk a fájl tartalmát.

```
[jerry@CentOS src]$ pwd
/home/jerry/jerry_repo/project/src

[jerry@CentOS src]$ git status -s
M string_operations.c

[jerry@CentOS src]$ git checkout string_operations.c

[jerry@CentOS src]$ git status –s
```

Ezenkívül használhatjuk a git checkout parancsot egy törölt fájl megszerzéséhez a helyi repository-ból. Tegyük fel, hogy Tom töröl egy fájlt a helyi repository-ból, és ezt a fájlt szeretnénk visszaadni. Ezt ugyanazzal a paranccsal érhetjük el.

```
[tom@CentOS src]$ pwd
/home/tom/top_repo/project/src

[tom@CentOS src]$ ls -1
Makefile
string_operations.c

[tom@CentOS src]$ rm string_operations.c

[tom@CentOS src]$ ls -1
Makefile

[tom@CentOS src]$ git status -s
D string_operations.c
```

Git a D betűt mutatja a fájlnév előtt. Ez azt jelzi, hogy a fájlt törölték a helyi repository-ból.

```
[tom@CentOS src]$ git checkout string_operations.c

[tom@CentOS src]$ ls -1
Makefile
string_operations.c

[tom@CentOS src]$ git status -s
```

Távolítsa el a változásokat az átmeneti területről (Staging Area)

Láttuk, hogy amikor hozzáadási műveletet hajtunk végre, a fájlok a helyi repository-ból a átmeneti területre (Staging Area) kerülnek. Ha egy felhasználó véletlenül módosít egy fájlt, és hozzáadja az átmeneti területhez (Staging Area), akkor a git checkout paranccsal visszaállíthatja a változtatásokat.

A Gitben van egy HEAD mutató, amely mindig a legutóbbi commit-ra mutat. Ha visszavonni szeretne egy módosítást az átmeneti területről (Staging Area), használhatja a git checkout parancsot, de a checkout paranccsal meg kell adnia egy további paramétert, vagyis a HEAD mutatót. A további commit mutató paraméter arra utasítja a git checkout parancsot, hogy állítsa alaphelyzetbe a működő fát, és távolítsa el a staged módosításokat is.

Tegyük fel, hogy Tom módosít egy fájlt a helyi adattárából. Ha megnézzük ennek a fájlnak az állapotát, akkor az azt mutatja, hogy a fájlt módosították, de nem adták hozzá az átmeneti területhez (Staging Area).

```
tom@CentOS src]$ pwd
/home/tom/top_repo/project/src
# Unmodified file

[tom@CentOS src]$ git status -s

# Modify file and view it's status.
[tom@CentOS src]$ git status -s
M string_operations.c

[tom@CentOS src]$ git add string_operations.c
```

A Git status azt mutatja, hogy a fájl az átmeneti területen (staging area) található, most állítsa vissza a git checkout paranccsal, és tekintse meg a visszaállított fájl állapotát.

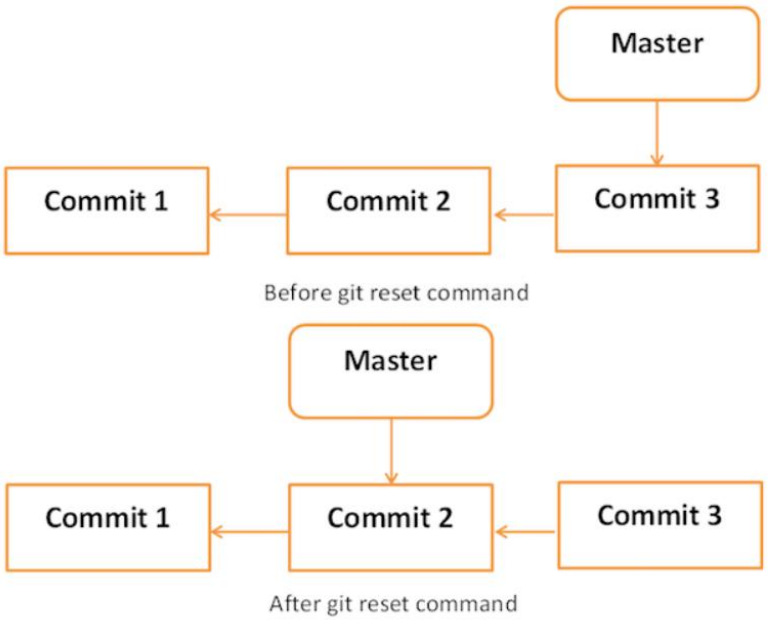
```
[tom@CentOS src]$ git checkout HEAD -- string_operations.c

[tom@CentOS src]$ git status -s
```

Mozgassa a HEAD mutatót a Git Reset használatával

Néhány módosítás elvégzése után dönthet úgy, hogy eltávolítja ezeket a módosításokat. A Git reset parancs a változtatások visszaállítására vagy törlésére szolgál. Három különböző típusú visszaállítási műveletet hajthatunk végre.

Az alábbi ábra a Git reset parancs képi ábrázolását mutatja.



Soft

Minden branch-nek van egy HEAD mutatója, amely a legutóbbi commit-ra mutat. Ha a Git reset parancsot használjuk a --soft opcióval, majd a commit azonosítóval, akkor csak a HEAD mutatót állítja alaphelyzetbe anélkül, hogy bármit is törölne.

Az .git / refs / heads / master fájl a HEAD mutató commit azonosítóját tárolja. A git log -1 paranccsal ellenőrizhetjük.

```
[jerry@CentOS project]$ cat .git/refs/heads/master
577647211ed44fe2ae479427a0668a4f12ed71a1
```

Most nézze meg a legújabb commit azonosítót, amely megegyezik a fenti commit azonosítóval.

```
[jerry@CentOS project]$ git log -2
```

A fenti parancs a következő eredményt adja.

```
commit 577647211ed44fe2ae479427a0668a4f12ed71a1
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 10:21:20 2013 +0530

Removed executable binary


commit 29af9d45947dc044e33d69b9141d8d2dad37cc62
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 10:16:25 2013 +0530

Added compiled binary
```

Állítsuk vissza a HEAD mutatót.

```
[jerry@CentOS project]$ git reset --soft HEAD~
```

Most csak egy pozícióval visszaállítottuk a HEAD mutatót. Ellenőrizzük a .git / refs / heads / master fájl tartalmát.

```
[jerry@CentOS project]$ cat .git/refs/heads/master
29af9d45947dc044e33d69b9141d8d2dad37cc62
```

A fájl commit azonosítója megváltozott, most ellenőrizze a commit üzenetek megtekintésével.

```
jerry@CentOS project]$ git log -2
```

A fenti parancs a következő eredményt adja.

```
commit 29af9d45947dc044e33d69b9141d8d2dad37cc62
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 10:16:25 2013 +0530

Added compiled binary


commit 94f7b26005f856f1a1b733ad438e97a0cd509c1a
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 10:08:01 2013 +0530
```

```
Added Makefile and renamed strings.c to string_operations.c
```

mixed

A Git reset --mixed opcióval visszaállítja azokat a változtatásokat az átmeneti területről (staging area), amelyeket még nem hajtottak végre. Csak a staging area-ről tér vissza a változásokra. A fájl munkapéldányának tényleges módosításait ez nem érinti. Az alapértelmezett Git reset egyenértékű a git reset-mixed paranccsal.

hard

Ha a --hard opciót használja a Git reset paranccsal, akkor az törli az átmeneti területet (staging area); visszaállítja a HEAD mutatót az adott commit azonosító legújabb commit-ját, és a helyi fájlváltozásokat is törli.

Ellenőrizzük a commit azonosítóját.

```
[jerry@CentOS src]$ pwd
/home/jerry/jerry_repo/project/src

[jerry@CentOS src]$ git log -1
```

A fenti parancs a következő eredményt adja.

```
commit 577647211ed44fe2ae479427a0668a4f12ed71a1
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 10:21:20 2013 +0530

Removed executable binary
```

Jerry módosított egy fájlt egy soros megjegyzés hozzáadásával a fájl elejére.

```
[jerry@CentOS src]$ head -2 string_operations.c
/* This line be removed by git reset operation */
#include <stdio.h>
```

A git status parancs segítségével ellenőrizte.

```
[jerry@CentOS src]$ git status -s
M string_operations.c
```

Jerry hozzáadja a módosított fájlt az átmeneti területhez (staging area), és a git status paranccsal ellenőrzi.

```
[jerry@CentOS src]$ git add string_operations.c
[jerry@CentOS src]$ git status
```

A fenti parancs a következő eredményt adja.

```
# On branch master
# Changes to be committed:
```

```
# (use "git reset HEAD <file>..." to unstage)
#
#
modified: string_operations.c
#
```

A Git status azt jelzi, hogy a fájl az átmeneti területen (staging area) található. Most állítsa vissza a HEAD -et -- hard opcióval.

```
[jerry@CentOS src]$ git reset --hard 577647211ed44fe2ae479427a0668a4f12ed71a1

HEAD is now at 5776472 Removed executable binary
```

A Git reset parancs sikeres volt, amely visszaállítja a fájlt az átmeneti területről (staging area), valamint eltávolítja a fájlban végrehajtott esetleges helyi változásokat.

```
[jerry@CentOS src]$ git status -s
```

A Git status azt mutatja, hogy a fájlt visszaállították az átmeneti területről (staging area).

```
[jerry@CentOS src]$ head -2 string_operations.c
#include <stdio.h>
```

A head parancs azt is megmutatja, hogy a reset művelet eltávolította a helyi változásokat is.

Git - Tag Operation

A címkeművelet lehetővé teszi, hogy értelmes neveket adjon a repository egy adott verziójához. Tegyük fel, hogy Tom és Jerry úgy döntenek, hogy felcímkézik a projektkódjukat, hogy később könnyen hozzáférhessenek hozzá.

Címkék (tag) létrehozása

Címkézzük az aktuális HEAD-et a git tag paranccsal. Tom megadja a tag nevét -a opcióval, a tag üzenetet pedig az –m opcióval.

```
tom@CentOS project]$ pwd
/home/tom/top_repo/project

[tom@CentOS project]$ git tag -a 'Release_1_0' -m 'Tagged basic string operation code' HEAD
```

Ha meg akar címkézni (tag) egy adott commit-ot, akkor használja a megfelelő COMMIT azonosítót a HEAD mutató helyett. Tom a következő paranccsal push-olja a címkét (tag) a távoli repository-ba.

```
[tom@CentOS project]$ git push origin tag Release_1_0
```

A fenti parancs a következő eredményt adja –

```
Counting objects: 1, done.
```

```
Writing objects: 100% (1/1), 183 bytes, done.
Total 1 (delta 0), reused 0 (delta 0)
To gituser@git.server.com:project.git
* [new tag]
Release_1_0 -> Release_1_0
```

Címkék megtekintése

Tom létrehozott címkéket. Most Jerry az összes elérhető címkét megtekintheti a Git tag parancs használatával az –l opcióval.

```
[jerry@CentOS src]$ pwd
/home/jerry/jerry_repo/project/src

[jerry@CentOS src]$ git pull
remote: Counting objects: 1, done.
remote: Total 1 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (1/1), done.
From git.server.com:project
* [new tag]
Release_1_0 -> Release_1_0
Current branch master is up to date.

[jerry@CentOS src]$ git tag -l
Release_1_0
```

Jerry a Git show parancsot és a címke nevét használja a címkével kapcsolatos további információk megtekintéséhez.

```
[jerry@CentOS src]$ git show Release_1_0
```

A fenti parancs a következő eredményt adja –

```
tag Release_1_0
Tagger: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 13:45:54 2013 +0530

Tagged basic string operation code


commit 577647211ed44fe2ae479427a0668a4f12ed71a1
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 10:21:20 2013 +0530

Removed executable binary

diff --git a/src/string_operations b/src/string_operations
deleted file mode 100755
index 654004b..0000000
Binary files a/src/string_operations and /dev/null differ
```

Címkék törlése

Tom a következő paranccsal törli a címkéket a helyi, valamint a távoli repository-ból.

```
[tom@CentOS project]$ git tag
Release_1_0

[tom@CentOS project]$ git tag -d Release_1_0
Deleted tag 'Release_1_0' (was 0f81ff4)
# Remove tag from remote repository.

[tom@CentOS project]$ git push origin :Release_1_0
```

```
To gituser@git.server.com:project.git
- [deleted]
Release_1_0
```

Git - Patch Operation

A Patch egy szöveges fájl, amelynek tartalma hasonló a Git diff-hez, de a kóddal együtt tartalmaz metaadatokat is a commet-ekről; pl. comment azonosító, dátum,comment üzenet stb. Készíthetünk patch-t comment-ekből, és mások alkalmazhatják őket a repository-nkban.

Jerry megvalósítja projektjéhez az strcat függvényt. Jerry létrehozhatja a kódja elérési útját, és elküldheti Tomnak. Ezután alkalmazhatja a kapott patch-t a kódjára.

Jerry a Git format-patch paranccsal hozza létre a legfrissebb commit patch-jét. Ha patch-t szeretne létrehozni egy adott commit-hoz, akkor használja a COMMIT_ID elemet a format-patch paranccsal.

```
[jerry@CentOS project]$ pwd
/home/jerry/jerry_repo/project/src

[jerry@CentOS src]$ git status -s
M string_operations.c
?? string_operations

[jerry@CentOS src]$ git add string_operations.c

[jerry@CentOS src]$ git commit -m "Added my_strcat function"

[master b4c7f09] Added my_strcat function
1 files changed, 13 insertions(+), 0 deletions(-)

[jerry@CentOS src]$ git format-patch -1
0001-Added-my_strcat-function.patch
```

A fenti parancs .patch fájlokat hoz létre az aktuális munkakönyvtárban. Tom ezzel a patch-el módosíthatja a fájljait. A Git két parancsot ad a git amand git apply –t a patch alkalmazásához. A Git apply módosítja a helyi fájlokat, anélkül, hogy létrehozná a dedikálást, míg a git amand módosítja a fájlt, és létrehoz egy dedikálást is.

A patch alkalmazásához és a végrehajtás létrehozásához használja a következő parancsot –

```
[tom@CentOS src]$ pwd
/home/tom/top_repo/project/src

[tom@CentOS src]$ git diff

[tom@CentOS src]$ git status –s

[tom@CentOS src]$ git apply 0001-Added-my_strcat-function.patch

[tom@CentOS src]$ git status -s
M string_operations.c
?? 0001-Added-my_strcat-function.patch
```

A patch sikeresen alkalmazható, most a git diff paranccsal tekinthetjük meg a módosításokat.

```
[tom@CentOS src]$ git diff
```

A fenti parancs a következő eredményt adja –

```
diff --git a/src/string_operations.c b/src/string_operations.c
index 8ab7f42..f282fcf 100644
--- a/src/string_operations.c
+++ b/src/string_operations.c
@@ -1,5 +1,16 @@
#include <stdio.h>
+char *my_strcat(char *t, char *s)
diff --git a/src/string_operations.c b/src/string_operations.c
index 8ab7f42..f282fcf 100644
--- a/src/string_operations.c
+++ b/src/string_operations.c
@@ -1,5 +1,16 @@
#include <stdio.h>
+char *my_strcat(char *t, char *s)
+
{
   +
   char *p = t;
   +
   +
   +
   while (*p)
   ++p;
   +
   while (*p++ = *s++)
   + ;
   + return t;
   +
}
+
size_t my_strlen(const char *s)
{
   const char *p = s;
   @@ -23,6 +34,7 @@ int main(void)
   {
```

Git - Managing Branches

A branch működése lehetővé teszi egy újabb fejlesztési vonal létrehozását. Használhatjuk ezt a műveletet a fejlesztési folyamat két különböző irányba történő elágazására. Például kiadtunk egy terméket a 6.0 verzióhoz, és érdemes létrehozni egy ágat, hogy a 7.0 funkciók fejlesztése külön maradjon a 6.0 hibajavításoktól.

Branch létrehozása

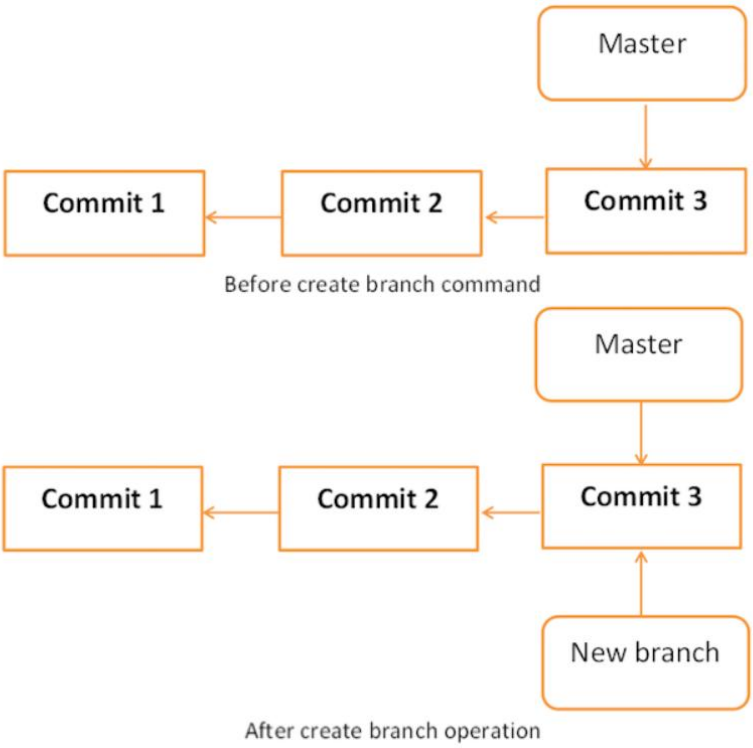
Tom új ágat (branch) hoz létre a git branch <branch neve> paranccsal. Létrehozhatunk egy új ágat. Kiindulásként használhatunk egy adott commit-ot vagy címkét. Ha nincs megadva konkrét végrehajtási azonosító, akkor az ág létrejön, a HEAD kiindulópontként.

```
[jerry@CentOS src]$ git branch new_branch

[jerry@CentOS src]$ git branch
* master
new_branch
```

Új branch jön létre; Tom a git branch paranccsal listázta az elérhető ágakat. A Git csillagjelet mutat az aktuálisan kijelölt branch előtt.

A branch létrehozásának műveleti képi ábrázolása az alábbiakban látható –



Váltás az ágak között

Jerry a git checkout paranccsal vált az ágak között.

```
[jerry@CentOS src]$ git checkout new_branch
Switched to branch 'new_branch'
[jerry@CentOS src]$ git branch
master
* new_branch
```

Parancs az ág létrehozására és váltására

A fenti példában két parancsot használtunk branch-ek létrehozására és váltására. A Git biztosítja a –b opciót a checkout paranccsal; ez a művelet új ágat hoz létre, és azonnal átáll az új ágra.

```
[jerry@CentOS src]$ git checkout -b test_branch
Switched to a new branch 'test_branch'

[jerry@CentOS src]$ git branch
master
new_branch
* test_branch
```

Branch törlése

Az ág törölhető a –D opció megadásával a git branch paranccsal. De a meglévő ág törlése előtt váltson a másik ágra.

Jerry jelenleg a test_branch oldalon van, és el akarja távolítani ezt az ágat. Tehát ágat vált és törli az ágat az alábbiak szerint.

```
[jerry@CentOS src]$ git branch
master
```

```
new_branch
* test_branch

[jerry@CentOS src]$ git checkout master
Switched to branch 'master'

[jerry@CentOS src]$ git branch -D test_branch
Deleted branch test_branch (was 5776472).
```

Most a Git csak két ágat mutat.

```
[jerry@CentOS src]$ git branch
* master
new_branch
```

Átnevezhet egy branch-t

Már létrehozott egy új branch-t, de a branch neve nem megfelelő. Tehát megváltoztatja az ág nevét az –m opcióval.

```
[jerry@CentOS src]$ git branch
* master
new_branch

[jerry@CentOS src]$ git branch -m new_branch wchar_support
```

Most a git branch parancs megmutatja az új ág nevét.

```
[jerry@CentOS src]$ git branch
* master
wchar_support
```

Két ág egyesítése

Jerry új kódot készít. Új a kód a következőképpen jelenik meg –

```
[jerry@CentOS src]$ git branch
master
* wchar_support

[jerry@CentOS src]$ pwd
/home/jerry/jerry_repo/project/src

[jerry@CentOS src]$ git diff
```

A fenti parancs a következő eredményt adja –

```
t a/src/string_operations.c b/src/string_operations.c
index 8ab7f42..8fb4b00 100644
--- a/src/string_operations.c
+++ b/src/string_operations.c
@@ -1,4 +1,14 @@
#include <stdio.h>
+#include <wchar.h>
+
+size_t w_strlen(const wchar_t *s)
+
{
   +
   const wchar_t *p = s;
   +
```

```
   +
   while (*p)
   + ++p;
   + return (p - s);
   +
}
```

Tesztelés után commit-olja és push-olja változtatásait az új ágra.

```
[jerry@CentOS src]$ git status -s
M string_operations.c
?? string_operations

[jerry@CentOS src]$ git add string_operations.c

[jerry@CentOS src]$ git commit -m 'Added w_strlen function to return string lenght of wchar_t
string'

[wchar_support 64192f9] Added w_strlen function to return string lenght of wchar_t string
1 files changed, 10 insertions(+), 0 deletions(-)
```

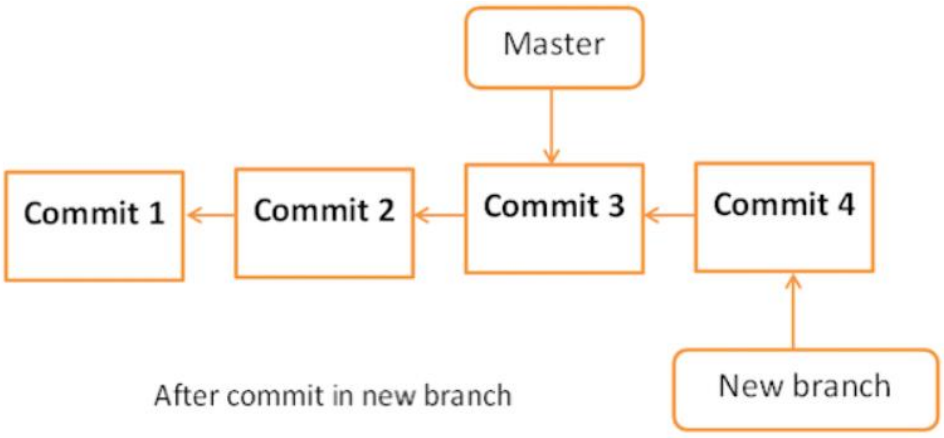
Ne feledje, hogy Jerry ezeket a változtatásokat push-olja az új ágra, ezért használta a wchar_support ág nevet a master ág helyett.

```
[jerry@CentOS src]$ git push origin wchar_support    <--- Observer branch_name
```

A fenti parancs a következő eredményt adja.

```
Counting objects: 7, done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 507 bytes, done.
Total 4 (delta 1), reused 0 (delta 0)
To gituser@git.server.com:project.git
* [new branch]
wchar_support -> wchar_support
```

A változtatások commit-olása után az új branch a következőképpen jelenik meg:



Tom kíváncsi arra, hogy Jerry mit csinál a magán branch-ben, és ellenőrzi a naplót a wchar_support branch-nél.

```
[tom@CentOS src]$ pwd
/home/tom/top_repo/project/src
```

```
[tom@CentOS src]$ git log origin/wchar_support -2
```

A fenti parancs a következő eredményt adja.

```
commit 64192f91d7cc2bcdf3bf946dd33ece63b74184a3
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 16:10:06 2013 +0530

Added w_strlen function to return string lenght of wchar_t string


commit 577647211ed44fe2ae479427a0668a4f12ed71a1
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 10:21:20 2013 +0530

Removed executable binary
```

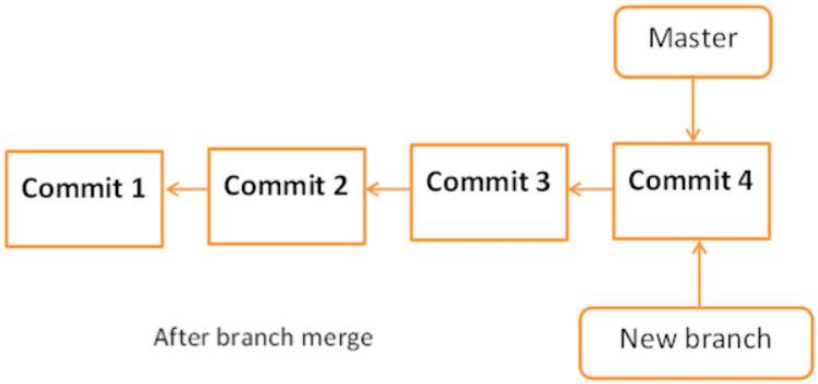
A kommit üzenetek megtekintésével Tom rájön, hogy Jerry a strlen függvényt számára valósította meg, és ugyanezt a funkcionalitást akarja a master ágban is. Az újbóli megvalósítás helyett úgy dönt, hogy felveszi Jerry kódját azáltal, hogy egyesíti a branch-et a fő ággal.

```
[tom@CentOS project]$ git branch
* master

[tom@CentOS project]$ pwd
/home/tom/top_repo/project

[tom@CentOS project]$ git merge origin/wchar_support
Updating 5776472..64192f9
Fast-forward
src/string_operations.c | 10 ++++++++++
1 files changed, 10 insertions(+), 0 deletions(-)
```

A merge művelet után a master branch a következőképpen jelenik meg:



Most a wchar_support ágat egyesítették a fő ággal. Ellenőrizhetjük a véglegesítési üzenet vagy a string_operation.c fájlban végrehajtott módosítások megtekintésével.

```
[tom@CentOS project]$ cd src/

[tom@CentOS src]$ git log -1

commit 64192f91d7cc2bcdf3bf946dd33ece63b74184a3
Author: Jerry Mouse
```

```
Date: Wed Sep 11 16:10:06 2013 +0530

Added w_strlen function to return string lenght of wchar_t string

[tom@CentOS src]$ head -12 string_operations.c
```

A fenti parancs a következő eredményt adja.

```
#include <stdio.h>
#include <wchar.h>
size_t w_strlen(const wchar_t *s)
{
   const wchar_t *p = s;

   while (*p)
      ++p;

   return (p - s);
}
```

Tesztelés után a kódváltozásokat a master ágra tolja.

```
[tom@CentOS src]$ git push origin master
Total 0 (delta 0), reused 0 (delta 0)
To gituser@git.server.com:project.git
5776472..64192f9 master -> master
```

Rebase Branches

A Git rebase parancs elágazási egyesítési parancs, de a különbség az, hogy módosítja a végrehajtások sorrendjét.

A Git merge parancs megpróbálja más ágakból származó végrehajtásokat az aktuális helyi fiók HEAD tetejére tenni. Például a helyi branch commit-jai A–> B–> C–> D, az merge ág pedig A–> B–> X–> Y, majd a git merge átalakítja az aktuális helyi ágat A–> B–> C–> D–> X–> Y

A Git rebase parancs megpróbálja kideríteni a közös ősöket az aktuális helyi ág és az egyesítési ág között. Ezután az aktuális helyi repository-ban végrehajtott változtatások sorrendjének módosításával push-oljs a helyi repository commit-ját. Például, ha a helyi branch commit-jai A–> B–> C–> D, a merge ág pedig A–> B–> X–> Y, akkor a Git rebase átalakítja az aktuális helyi ágat valami hasonlóra, mint: A– > B–> X–> Y–> C–> D.

Ha több fejlesztő dolgozik egyetlen távoli repository-n, akkor nem módosíthatja a távoli repository-ban végrehajtott sorrendeket. Ebben a helyzetben a rebase művelettel a helyi commit-okat a távoli repository tetejére helyezhetjük, mikor a változásokat push-olunk be.

Git - Handling Conflicts

Végezze el a változtatásokat a wchar_support branch-en

Jerry a wchar_support branch-en dolgozik. Megváltoztatja a függvények nevét, és tesztelés után commit-olja a változtatásokat.

```
[jerry@CentOS src]$ git branch
 master
* wchar_support
[jerry@CentOS src]$ git diff
```

A fenti parancs a következő eredményt adja –

```
diff --git a/src/string_operations.c b/src/string_operations.c
index 8fb4b00..01ff4e0 100644
--- a/src/string_operations.c
+++ b/src/string_operations.c
@@ -1,7 +1,7 @@
#include <stdio.h>
#include <wchar.h>
-size_t w_strlen(const wchar_t *s)
+size_t my_wstrlen(const wchar_t *s)
{
   const wchar_t *p = s;
```

A kód ellenőrzése után végrehajtja a változtatásokat.

```
[jerry@CentOS src]$ git status -s
M string_operations.c

[jerry@CentOS src]$ git add string_operations.c

[jerry@CentOS src]$ git commit -m 'Changed function name'
[wchar_support 3789fe8] Changed function name
1 files changed, 1 insertions(+), 1 deletions(-)

[jerry@CentOS src]$ git push origin wchar_support
```

A fenti parancs a következő eredményt adja –

```
Counting objects: 7, done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 409 bytes, done.
Total 4 (delta 1), reused 0 (delta 0)
To gituser@git.server.com:project.git
64192f9..3789fe8 wchar_support -> wchar_support
```

Végezze el a változásokat a főágban

Eközben a fő ágban Tom is megváltoztatja ugyanannak a funkciónak a nevét, és a változtatásokat a fő ágba push-olja.

```
[tom@CentOS src]$ git branch
* master
[tom@CentOS src]$ git diff
```

A fenti parancs a következő eredményt adja –

```
diff --git a/src/string_operations.c b/src/string_operations.c
index 8fb4b00..52bec84 100644
--- a/src/string_operations.c
+++ b/src/string_operations.c
@@ -1,7 +1,8 @@
```

```
#include <stdio.h>
#include <wchar.h>
-size_t w_strlen(const wchar_t *s)
+/* wide character strlen fucntion */
+size_t my_wc_strlen(const wchar_t *s)
{
   const wchar_t *p = s;
```

A különbség ellenőrzése után végrehajtja a változtatásokat.

```
[tom@CentOS src]$ git status -s
M string_operations.c

[tom@CentOS src]$ git add string_operations.c

[tom@CentOS src]$ git commit -m 'Changed function name from w_strlen to my_wc_strlen'
[master ad4b530] Changed function name from w_strlen to my_wc_strlen
1 files changed, 2 insertions(+), 1 deletions(-)

[tom@CentOS src]$ git push origin master
```

A fenti parancs a következő eredményt adja –

```
Counting objects: 7, done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 470 bytes, done.
Total 4 (delta 1), reused 0 (delta 0)
To gituser@git.server.com:project.git
64192f9..ad4b530 master -> master
```

A wchar_support ágon Jerry végrehajtja az strchr függvényt. Tesztelés után commit-olja és push-olja a változtatásokat a wchar_support ágba.

```
[jerry@CentOS src]$ git branch
master
* wchar_support
[jerry@CentOS src]$ git diff
```

A fenti parancs a következő eredményt adja –

```
diff --git a/src/string_operations.c b/src/string_operations.c
index 01ff4e0..163a779 100644
--- a/src/string_operations.c
+++ b/src/string_operations.c
@@ -1,6 +1,16 @@
#include <stdio.h>
#include <wchar.h>
+wchar_t *my_wstrchr(wchar_t *ws, wchar_t wc)
+
{
   +
   while (*ws)
   {
      +
      if (*ws == wc)
      +
      return ws;
      +
      ++ws;
      +
   }
   + return NULL;
   +
}
+
```

```
size_t my_wstrlen(const wchar_t *s)
{
   const wchar_t *p = s;
```

Ellenőrzés után commit-olja a változtatásokat.

```
[jerry@CentOS src]$ git status -s
M string_operations.c

[jerry@CentOS src]$ git add string_operations.c

[jerry@CentOS src]$ git commit -m 'Addded strchr function for wide character string'
[wchar_support 9d201a9] Addded strchr function for wide character string
1 files changed, 10 insertions(+), 0 deletions(-)

[jerry@CentOS src]$ git push origin wchar_support
```

A fenti parancs a következő eredményt adja –

```
Counting objects: 7, done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 516 bytes, done.
Total 4 (delta 1), reused 0 (delta 0)
To gituser@git.server.com:project.git
3789fe8..9d201a9 wchar_support -> wchar_support
```

Konfliktusok kezelése

Tom meg akarja nézni, mit csinál Jerry a saját fiókjában, ezért megpróbálja pull-ozni a legújabb módosításokat a wchar_support fiókból, de Git a következő hibaüzenettel megszakítja a műveletet.

```
[tom@CentOS src]$ git pull origin wchar_support
```

A fenti parancs a következő eredményt adja –

```
remote: Counting objects: 11, done.
63Git Tutorials
remote: Compressing objects: 100% (8/8), done.
remote: Total 8 (delta 2), reused 0 (delta 0)
Unpacking objects: 100% (8/8), done.
From git.server.com:project
* branch
wchar_support -> FETCH_HEAD
Auto-merging src/string_operations.c
CONFLICT (content): Merge conflict in src/string_operations.c
Automatic merge failed; fix conflicts and then commit the result.
```

Konfliktusok megoldása

A hibaüzenetből kiderül, hogy ütközés van az src / string_operations.c fájlban. A további részletek megtekintéséhez futtatja a git diff parancsot.

```
[tom@CentOS src]$ git diff
```

A fenti parancs a következő eredményt adja –

```
diff --cc src/string_operations.c
index 52bec84,163a779..0000000
```

```
--- a/src/string_operations.c
+++ b/src/string_operations.c
@@ -1,8 -1,17 +1,22 @@@
#include <stdio.h>
#include <wchar.h>
++<<<<<<< HEAD
+/* wide character strlen fucntion */
+size_t my_wc_strlen(const wchar_t *s)
++=======
+ wchar_t *my_wstrchr(wchar_t *ws, wchar_t wc)
+
{
   +
   +
   while (*ws)
   {
      if (*ws == wc)
      +
      return ws;
      +
      ++ws;
      +
   }
   + return NULL;
   +
}
+
+ size_t my_wstrlen(const wchar_t *s)
++>>>>>>> 9d201a9c61bc4713f4095175f8954b642dae8f86
{
   const wchar_t *p = s;
```

Mivel Tom és Jerry is megváltoztatta ugyanazon funkció nevét, Git zavart állapotban van, és kéri a felhasználót, hogy oldja meg manuálisan a konfliktust.

Tom úgy dönt, hogy megtartja a Jerry által javasolt függvénynevet, de az általa hozzáírt megjegyzést megtartja. A konfliktusjelzők eltávolítása után a git diff így fog kinézni.

```
[tom@CentOS src]$ git diff
```

A fenti parancs a következő eredményt adja.

```
diff --cc src/string_operations.c
diff --cc src/string_operations.c
index 52bec84,163a779..0000000
--- a/src/string_operations.c
+++ b/src/string_operations.c
@@@ -1,8 -1,17 +1,18 @@@
#include <stdio.h>
#include <wchar.h>
+ wchar_t *my_wstrchr(wchar_t *ws, wchar_t wc)
+
{
   +
   while (*ws)
   {
      +
      if (*ws == wc)
      +
      return ws;
      +
      ++ws;
      +
   }
   + return NULL;
   +
}
+
+/* wide character strlen fucntion */
```

```
- size_t my_wc_strlen(const wchar_t *s)
+ size_t my_wstrlen(const wchar_t *s)
{
   const wchar_t *p = s;
```

Mivel Tom módosította a fájlokat, először ezeket a változtatásokat kell végrehajtania, majd ezt követően pull-ozhatja a módosításokat.

```
[tom@CentOS src]$ git commit -a -m 'Resolved conflict'
[master 6b1ac36] Resolved conflict

[tom@CentOS src]$ git pull origin wchar_support.
```

Tom megoldotta a konfliktust, most a pull művelet sikeres lesz.

Git - Different Platforms

A GNU / Linux és a Mac OS a line-feed (LF), vagy az új sort használja végződő karakterként, míg a Windows a line-feed and carriage-return (LFCR) kombinációt használja a sor végződő karakter képviseletére.

Ezen sorvégi különbségek miatti szükségtelen elkövetések elkerülése érdekében be kell állítanunk a Git klienst, hogy ugyanazt a sorvéget írja a Git adattárba.

Windows rendszer esetén beállíthatjuk a Git klienst, hogy a sorvégeket CRLF formátumba konvertálja, miközben kijelentkezik, és a végrehajtási művelet során visszaállítja őket LF formátumba. A következő beállítások szükségesek.

```
[tom@CentOS project]$ git config --global core.autocrlf true
```

GNU / Linux vagy Mac OS esetén konfigurálhatjuk a Git klienst a sorvégződések CRLF-ről LF-re konvertálására.

```
[tom@CentOS project]$ git config --global core.autocrlf input
```

Git - Online Repositories

A GitHub egy webalapú tárhelyszolgáltatás szoftverfejlesztési projektekhez, amely a Git verziókezelő rendszert használja. Ezenkívül a szokásos GUI alkalmazásuk is letölthető (Windows, Mac, GNU / Linux), közvetlenül a szolgáltatás webhelyéről. De ebben a részben csak a CLI részt fogjuk látni.

Hozzon létre GitHub-repository-t

Nyissa meg a github.com oldalt. Ha már rendelkezik GitHub-fiókkal, akkor jelentkezzen be azzal a fiókkal, vagy hozzon létre egy újat. Kövesse a github.com webhelyének lépéseit egy új repository létrehozásához.

Push művelet

Tom úgy dönt, hogy a GitHub szervert használja. Új projekt elindításához létrehoz egy új könyvtárat és egy fájlt azon belül.

```
[tom@CentOS]$ mkdir github_repo

[tom@CentOS]$ cd github_repo/

[tom@CentOS]$ vi hello.c

[tom@CentOS]$ make hello
cc hello.c -o hello

[tom@CentOS]$ ./hello
```

A fenti parancs a következő eredményt adja:

```
Hello, World !!!
```

Kódjának ellenőrzése után inicializálja a könyvtárat a git init paranccsal, és lokálisan commit-olja a változtatásokat.

```
[tom@CentOS]$ git init
Initialized empty Git repository in /home/tom/github_repo/.git/

[tom@CentOS]$ git status -s
?? hello
?? hello.c

[tom@CentOS]$ git add hello.c

[tom@CentOS]$ git status -s
A hello.c
?? hello

[tom@CentOS]$ git commit -m 'Initial commit'
```

Ezt követően hozzáadja a GitHub repository URL-jét remote origin-ként, és a módosításokat a távoli repository-ba push-olja.

```
[tom@CentOS]$ git remote add origin https://github.com/kangralkar/testing_repo.git

[tom@CentOS]$ git push -u origin master
```

A push művelet meg fogja kérni a GitHub felhasználói nevet és jelszót. Sikeres hitelesítés után a művelet sikeres lesz.

A fenti parancs a következő eredményt adja:

```
Username for 'https://github.com': kangralkar
Password for 'https://kangralkar@github.com':
```

```
Counting objects: 3, done.
Writing objects: 100% (3/3), 214 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/kangralkar/test_repo.git
 * [new branch]      master -> master
 Branch master set up to track remote branch master from origin.
```

Mostantól kezdve Tom bármilyen módosítást push-olhat a GitHub repóba. Használhatja az ebben a fejezetben tárgyalt összes parancsot a GitHub adattárral.

Pull művelet

Tom minden változtatását sikeresen push-olta a GitHub adattárba. Most más fejlesztők clone művelet végrehajtásával vagy a helyi repó frissítésével tekinthetik meg ezeket a változásokat.

Jerry létrehoz egy új könyvtárat a saját könyvtárában, és a git clone paranccsal klónozza a GitHub repót.

```
[jerry@CentOS]$ pwd
/home/jerry

[jerry@CentOS]$ mkdir jerry_repo

[jerry@CentOS]$ git clone https://github.com/kangralkar/test_repo.git
```

A fenti parancs a következő eredményt adja:

```
Cloning into 'test_repo'...
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 3 (delta 0)
Unpacking objects: 100% (3/3), done.
```

A könyvtár tartalmát az ls parancs végrehajtásával ellenőrzi.

```
[jerry@CentOS]$ ls
test_repo

[jerry@CentOS]$ ls test_repo/
hello.c
```