git - kapitola 1. repositáře a větve

Git je rychlý distribuovaný systém pro správu verzí. Tato uživatelská príručka je napsána tak aby byla čitelná kýmkoliv se základními znalostmi UNIXové příkazové řádky, a to i bez předchozích zkušeností s gitem.

Obsah:

jak získat git repositář

Bude užitečné mít git repositář se kterým budete moci experimentovat během čtení tohoto manuálu.

Nejlepší způsob jak takový repositář získat je použít příkaz git-clone(1) pro zkopírování existujícího repositáře. Pokud vás žádný vhodný projekt nenapadá, tady je několik příkladů:

git samotný (přibližně 10 MB ke stažení)

1. $ git clone git: //git.kernel.org/pub/scm/git/git.git

Linuxové jádro (přibližně 150MB ke stažení)

1. $ git clone git: //git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux-2.6.git

Počáteční klonování může pro velký projekt chvíli trvat, ale klonovat budete muset jenom jednou.

Příkaz clone vytvoří nový adresář pojmenovaný po projektu ("git" nebo "linux-2.6" pro příklady uvedené výše). Po vstupu do tohoto adresáře uvidíte že obsahuje kopii souborů projektu, nazývané "pracovní strom" (working tree), společně se speciálním top-level adresářem ".git" obsahujícím informace o historii projektu.

jak načíst jinou verzi projektu

O gitu se nejlépe přemýšlí jako o nástroji pro ukládání historie skupiny souborů. Historie je ukládána ve formě zkomprimované kolekce vzájemně souvisejících snapshotů obsahu projektu. V gitu je každá taková verze nazývána commit.

Tyto snapshoty nemusí být nutně uspořádány v jediné posloupnosti od nejstaršího po nejnovější; namísto toho může práce současně probíhat v několika paralelních vývojových liniích, nazývaných větve (branches), které se mohou spojovat i rozdělovat.

Jediný gitový repositář může udržovat vývoj v několika větvích současně. Dosahuje toho udržováním seznamu hrotů (heads) které odkazují na poslední commit v každé větvi; příkaz git-branch(1) vám vypíše seznam hrotů větví:

1. $ git branch
2. * master

Čerstvě naklonovaný repositář obsahuje obsahuje jediný vrchol, pojmenovaný standardně "master", s pracovním adresářem inicializovaným do stavu projektu ve kterém byl odkazovaný daným vrchol větve.

Většina projektů také používá tagy. Tagy, stejně jako vrcholy, jsou odkazy do historie projektu, a jejich seznam může být vypsán pomocí příkazu git-tag(1):

1. $ git tag -l
2. v2.6.11
3. v2.6.11-tree
4. v2.6.12
5. v2.6.12-rc2
6. v2.6.12-rc3
7. v2.6.12-rc4
8. v2.6.12-rc5
9. v2.6.12-rc6
10. v2.6.13

11. ...

U tagů se očekává že vždy odkazují na stejnou verzi projektu, zatímco u vrcholů se předpokládá že se mění s tím jak vývoj pokračuje.

Nový vrchol odkazující na jednu z těchto revizí vytvoříte a načtete příkazem git-checkout(1):

1. $ git checkout -b new v2.6.13

Pracovní adresář poté odráží obsah který projekt měl v okamžiku kdy byl otagován jako v2.6.13, a příkaz git-branch(1) vypisuje dvě větve, s hvězdičkou označující aktuálně vybranou větev:

1. $ git branch
2. master
3. * new

Pokud se rozhodnete že byste raději viděli verzi 2.6.17, můžete modifikovat aktuální větev tak aby ukazovala na v2.6.17, a to příkazem

1. $ git reset --hard v2.6.17

Uvědomte si že pokud byl vrchol aktuální větve byl váš jediný odkaz na konkrétní bod v historii projektu, potom po přepnutí této větve nemusíte mít žádný způsob jak se odkazovat na původně odkazované historické verze; tento příkaz tedy používejte opatrně.

porozumění historii: commity

Každá změna v historii projektu je reprezentována commitem. Příkaz git-show(1) vypisuje informace o posledním commitu v aktuální větvi:

```
1.
$ git show

2.
commit 17cf781661e6d38f737f15f53ab552f1e95960d7

3.
Author: Linus Torvalds < torvalds @ ppc970.osdl.org.(none) >Date:   Tue Apr 19 14:11:06 2005 -0700

4.


5.
    Remove duplicate getenv(DB_ENVIRONMENT) call

6.


7.
    Noted by Tony Luck.

8.


9.
diff --git a/init-db.c b/init-db.c

10.
index 65898fa..b002dc6 100644

11.
--- a/init-db.c

12.
+++ b/init-db.c

13.
@@ -7,7 +7,7 @@

14.


15.
 int main(int argc, char **argv)

16.
 {

17.
-       char *sha1_dir = getenv(DB_ENVIRONMENT), *path;

18.
+       char *sha1_dir, *path;

19.
```

```
        int len, i;

20.

21.
        if (mkdir(".git", 0755) < 0) {
```

Jak vidíte, commit vypisuje informace kdo provedl poslední změnu, jakou změnu provedl, a proč.

Každý commit má 40-znakové hexadecimální ID, někdy nazývané také "jméno objektu" (object name) nebo "SHA-1 id", zobrazované na prvním řádku výstupu příkazu "git show". Na commit se obvykle můžete odkazovat i kratším jménem, jako je například název větve, ale toto delší jméno může být také užitečné. A co je nejdůležitější, toto jméno je globálně unikátní jméno pro tento commit: takže pokud někomu jinému sdělíte objektové jméno (například v e-mailu), potom máte zaručeno že toto jméno bude v jejich repositáři odkazovat na stejný commit jako v tom vašem (za předpokladu že jejich repositář váš commit vůbec obsahuje). Protože objektové jméno je počítáno jako hash obsahu commitu, máte zaručeno že commit se nikdy nemůže změnit aniž by se změnilo také jeho jméno.

Ve skutečnosti, v Kapitole 7, Koncepty Gitu se podíváme že všechno uložené v historii gitu, včetně dat souborů a obsahu adresářů, je ukládáno v objektu se jménem které je hashem jeho obsahu.

porozumění historii: commity, předkové, a dosažitelnost

Každý commit (s výjimkou prvního commitu v projektu) má také rodičovský (parent) commit, zachycující co se událo před daným commitem. Sledováním řetězu rodičovských commitů vás nakonec dovede zpět na začátek projektu.

Avšak, commity nejsou uspořádány do jednoduchého seznamu; git umožňuje aby se linie vývoje rozdělilily a poté opět spojily, přičemž bod ve kterém se dvě linie vývoje spojují se nazývá "merge". Commit reprezentující merge tudíž může mít více než jednoho předka, přičemž každý předek reprezentuje poslední commit v jedné z linií vývoje vedoucí k tomuto bodu.

Nejlepší způsob jak ukázat jak toto funguje je pomocí příkazu gitk(1); spuštění gitk na git repositáři a vyhledání merge commitů vám pomůže pochopit jak git organizuje historii.

Dále říkáme že commit X je "dosažitelný" z commitu Y pokud je commit X předkem commitu Y. Ekvivalentně říkáme že Y je potomek X, nebo že existuje řetězec předků vedoucí z commitu Y do commitu X.

porozumění historii: diagramy historie

Git historii budeme někdy znázorňovat pomocí diagramů jako je ten následující. Commity jsou zobrazeny jako "o", a spojnice mezi nimi jsou vykresleny jako - / a \. Čas plyne zleva doprava:

```
1.
        o--o--o <-- Branch A

2.
        /

3.
    o--o--o <-- master

4.
        \

5.
```

```
            o--o--o <-- Branch B
```

Pokud potřebujeme mluvit o konkrétním commitu, potom znak "o" může být nahrazen jiným písmenem nebo číslicí.

porozumění historii: co je větev?

Pokud budeme potřebovat být přesní, budeme slovo "větev" (branch) používat pro označení linie vývoje, a "vrchol větve" (branch head) pro odkaz na poslední commit ve větvi. Ve výše uvedeném příkladu, vrchol větve "A" je ukazatel na jeden konkrétní commit, ale celou posloupnost tří commitů vedoucích k tomuto bodu budeme označovat jako "větev A".

Nicméně, pokud nebude hrozit žádné zmatení, budeme často používat pouze označení větev" jak pro označení větve tak i jejího vrcholu.

manipulace s větvemi

Vytváření, mazání a modifikace větví je rychlé a jednoduché: tady je souhrn příkazů:

- **git branch**
  vypíše všechny větve
- **git branch < branch >**
  vytvoří novou větev nazvanou < branch >, odkazující na stejný bod v historii jako současná větev
- **git branch < branch > < start-point >**
  vytvoří novou větev nazvanou < branch >, odkazující na < start-point >, který může být specifikován jakkoliv chcete, včetně jména větve nebo jména tagu
- **git branch -d < branch >**
  smaže větev < branch >; pokud větev kterou mažete odkazuje na commit který není dosažitelný z aktuální větve, tento příkaz selže s varováním.
- **git branch -D < branch >**
  dokonce i pokud větev ukazuje na commit nedosažitelný z aktuální větve, můžete vědět že daný commit je stále dosažitelný z nějaké jiné větve nebo tagu. V takovém případě je bezpečné použít tento příkaz pro smazání větve.
- **git checkout < branch >**
  udělá z větve < branch > aktuální větev, přičemž aktualizuje pracovní adresář aby obsahoval na verzi ve větvi < branch >
- **git checkout -b < new > < start-point >**
  vytvoří novou větev < new > odkazující < start-point >, a načte ji

Speciální symbol "HEAD" může být použit jako odkaz na aktuální větev. Ve skutečnosti git používá soubor nazývaný "HEAD" v adresáři .git aby si pamatoval která větev je aktuální:

1. $ cat .git/HEAD
2. ref: refs/heads/master

zkoumání staré verze bez vytváření nové větve

Příkaz *git checkout* normálně očekává vrchol větve, ale přijme také jakékoliv commit; například můžete načíst commit na který odkazuje tag:

1. $ git checkout v2.6.17
2. Note: moving to "v2.6.17" which isn't a local branch
3. If you want to create a new branch from this checkout, you may do so
4. (now or later) by using -b with the checkout command again. Example:
5. git checkout -b HEAD is now at 427abfa... Linux v2.6.17
6.

HEAD poté odkazuje na SHA-1 commitu namísto větve, a výstup příkazu git branch ukazuje že nadále nejste ve větvi:

1. $ cat .git/HEAD
2. 427abfa28afedffadfca9dd8b067eb6d36bac53f
3. $ git branch
4. * (no branch)
5. master

V tomto případě říkáme že HEAD je "odpojená".

Toto je jednoduchý způsob jak načíst určitou větev aniž byste si museli vymýšlet název pro novou větev. Vždy můžete vytvořit

novou větev (nebo tag) pokud se tak později rozhodnete.

zkoumání větví ve vzdáleném repositáři

Větev "master" která byla vytvořena v okamžiku klonování je kopií HEAD z repositáře který jste klonovali. Nicméně tento repositář také může obsahovat další větve, a váš lokální repositář udržuje větve které sledují každou z těchto vzdálených větví, a tyto větve si můžete zobrazit pomocí přepínače "-r" příkazu git-branch(1):

1. $ git branch -r
2. origin/HEAD
3. origin/html
4. origin/maint
5. origin/man
6. origin/master
7. origin/next
8. origin/pu
9. origin/todo

Tyto větve sloužící ke sledování větví ve vzdáleném repositáři nelze načíst (check out), ale můžete je zkoumat v libovolné větvi, stejně jako v případě tagu:

1. $ git checkout -b my-todo-copy origin/todo

Všimněte si že jméno "origin" je pouze jméno které git standardně používá pro odkazování na repositář ze kterého jste klonovali.

pojmenovávání větví, tagů, a dalších referencí

Větve, větve pro sledování vzdálených větví (remote-tracking branches), a tagy jsou všechno odkazy na commity. K pojmenování odkazů jsou použity cesty s lomítky jako oddělovači, začínající s "refs"; jména které jsme používali dosud jsou ve skutečnosti zkratky:

- Větev "test" je zkratka pro "refs/heads/test".

- Tag "v2.6.18" je zkratka pro "refs/tags/v2.6.18".

- "origin/master" je zkratka pro "refs/remotes/origin/master".

Plné jméno se občas hodí pokud, například, existují tag a větev se stejným jménem.

(Nově vytvořené reference jsou ve skutečnosti uloženy v adresáři .git/refs, v adresáři daným jejich jménem. Avšak pro úspornost mohou být také zabaleny společně do jediného souboru; viz. příkaz git-pack-refs(1)).

Další užitečná zkratka, na "HEAD" repositáře se můžete odkazovat používáním názvu samotného repositáře. Takže, například, "origin" je obvykle zkratka pro HEAD větev v repositáři "origin".

Pro úplný seznam cest ve kterých git vyhledává odkazy, a pořadí které používá pro rozhodování kdy který odkaz použít pokud existuje několik odkazů se stejnou zkratkou, podívejte se na oddíl "SPECIFYING REVISIONS" v git-rev-parse(1).

aktualizace repositáře pomocí git fetch

Můžete také sledovat větve z repositářů ze kterých jste neklonovali, a to pomocí příkazu git-remote(1):

1. $ git remote add linux-nfs git: //linux-nfs.org/pub/nfs-2.6.git
2. $ git fetch linux-nfs
3. * refs/remotes/linux-nfs/master: storing branch 'master' ...
4. commit: bf81b46

Nové remote-tracking větve budou uloženy pod zkratkou kterou jim dáte při volání příkazu "git remote add", v tomto případě linux-nfs:

1. $ git branch -r
2. linux-nfs/master
3. origin/master

Pokud spustíte "git fetch < remote >" později, tracking větve pro jmenovaný < remote > repositář budou aktualizovány.

Pokud se podíváte na soubor .git/config, uvidíte že git do něj přidal následující odstavec:

1. $ cat .git/config
2. ...
3. [remote "linux-nfs"]
4. url = git: //linux-nfs.org/pub/nfs-2.6.git
5. fetch = +refs/heads/*:refs/remotes/linux-nfs/*
6. ...

Právě toto způsobuje že git sleduje vzdálené větve; tyto konfigurace můžete upravit nebo smazat úpravou souboru .git/config pomocí textového editoru. (detaily najdete v odstavci "CONFIGURATION FILE" v kapitole o git-config(1)).

Zdroj: http://www.fuzzy.cz/

Na http://knihy.nic.cz/ je zdarma dostupná kniha "Pro Git" v češtině, pokrývající Git daleko lépe a podrobněji.