# A bit more advanced git

Sanghee Kim

# What can I learn from this?

After learning about Git 101, you might need to know some set of commands to make your life get easier. So, I will let you know the following commands.

- git revert
- git rebase -i
- git stash
- git log -G
- gitk

# git revert

The git revert undoes a committed snapshot. However, instead of removing the commit from the history, it makes a reversed commit. So, you can still keep track of all your commit history.

$ git revert <commit>

# git revert: demo

Now, you have two commits which added one line respectively and pushed them into remote.

```
(ml3) sanghee@sp:~/Dev/demo$ git log -p
commit 105f911ad7241ae56b606f08276e6ea69a7bbaba
Author: Sanghee Kim <sangheestyle@gmail.com>
Date:   Wed May 6 00:14:43 2015 -0600

    second commit

diff --git a/a.txt b/a.txt
index 69dd9b9..36f11b6 100644
--- a/a.txt
+++ b/a.txt
@@ -1 +1,2 @@
 aaaaaaaaaa
+bbbbbbbbbb

commit 9894eb970b435d60af846f4b171637acbbb4e037
Author: Sanghee Kim <sangheestyle@gmail.com>
Date:   Wed May 6 00:14:05 2015 -0600

    first commit

diff --git a/a.txt b/a.txt
new file mode 100644
index 0000000..69dd9b9
--- /dev/null
+++ b/a.txt
@@ -0,0 +1 @@
+aaaaaaaaaa
```

# git revert: demo

You think the 2nd commit should be removed and you can do revert it.

```
(ml3) sanghee@sp:~/Dev/demo$ git revert HEAD
[master 8bf1f9c] Revert "second commit"
 1 file changed, 1 deletion(-)
(ml3) sanghee@sp:~/Dev/demo$ git log -p -1
commit 8bf1f9c11da6e1c3384c9c0ea5acacbb126af9dd
Author: Sanghee Kim <sangheestyle@gmail.com>
Date:   Wed May 6 00:21:26 2015 -0600

    Revert "second commit"

    This reverts commit 105f911ad7241ae56b606f08276e6ea69a7bbaba.

diff --git a/a.txt b/a.txt
index 36f11b6..69dd9b9 100644
--- a/a.txt
+++ b/a.txt
@@ -1,2 +1 @@
 aaaaaaaaaa
-bbbbbbbbbb
```

You can see the reverted commit which includes reversed content of the 2nd commit. Now, you can push it into your remote.

# git revert: demo

It’s really simple way to remove a specific commit but safe because you still have entire commit history.

```
(ml3) sanghee@sp:~/Dev/demo$ git log
commit 8bf1f9c11da6e1c3384c9c0ea5acacbb126af9dd
Author: Sanghee Kim <sangheestyle@gmail.com>
Date:   Wed May 6 00:21:26 2015 -0600

    Revert "second commit"

    This reverts commit 105f911ad7241ae56b606f08276e6ea69a7bbaba.

commit 105f911ad7241ae56b606f08276e6ea69a7bbaba
Author: Sanghee Kim <sangheestyle@gmail.com>
Date:   Wed May 6 00:14:43 2015 -0600

    second commit

commit 9894eb970b435d60af846f4b171637acbbb4e037
Author: Sanghee Kim <sangheestyle@gmail.com>
Date:   Wed May 6 00:14:05 2015 -0600

    first commit
```

It means you can make revert of reverted commit if you want.

# git rebase -i

Sometimes you might want to change commit history on your local before push them into remote repo. git rebase -i is a really helpful to do. -i means interactive.

$ git rebase -i <base>

# git rebase -i: demo

Now, we have three commits on our local repo. However, you think the order of last two commits should be changed.

$ git rebase -i **9894eb**

```
(ml3) sanghee@sp:~/Dev/demo$ git log -p
commit 56c4443cd9643f205a3ef5750971d669303622ec
Author: Sanghee Kim <sangheestyle@gmail.com>
Date:   Wed May 6 00:59:16 2015 -0600

    third commit

diff --git a/b.txt b/b.txt
new file mode 100644
index 0000000..beb6147
--- /dev/null
+++ b/b.txt
@@ -0,0 +1 @@
+ccccccccc

commit 105f911ad7241ae56b606f08276e6ea69a7bbaba
Author: Sanghee Kim <sangheestyle@gmail.com>
Date:   Wed May 6 00:14:43 2015 -0600

    second commit

diff --git a/a.txt b/a.txt
index 69dd9b9..36f11b6 100644
--- a/a.txt
+++ b/a.txt
@@ -1 +1,2 @@
 aaaaaaaaaa
+bbbbbbbbbb

commit 9894eb970b435d60af846f4b171637acbbb4e037
Author: Sanghee Kim <sangheestyle@gmail.com>
Date:   Wed May 6 00:14:05 2015 -0600

    first commit

diff --git a/a.txt b/a.txt
new file mode 100644
index 0000000..69dd9b9
--- /dev/null
+++ b/a.txt
@@ -0,0 +1 @@
+aaaaaaaaaa
```

# git rebase -i: demo

Git shows you two commit so you can replace order of them and save the file.

```
1 pick 105f911 second commit
2 pick 56c4443 third commit
3
4 # Rebase 9894eb9..56c4443 onto 9894eb9 (2 command(s))
5 #
```

```
 1 pick 56c4443 third commit
 2 pick 105f911 second commit
 3
 4 # Rebase 9894eb9..56c4443 onto 9894eb9 (2 command(s))
 5 #
 6 # Commands:
 7 # p, pick = use commit
 8 # r, reword = use commit, but edit the commit message
 9 # e, edit = use commit, but stop for amending
10 # s, squash = use commit, but meld into previous commit
11 # f, fixup = like "squash", but discard this commit's log message
12 # x, exec = run command (the rest of the line) using shell
13 #
14 # These lines can be re-ordered; they are executed from top to bottom.
15 #
16 # If you remove a line here THAT COMMIT WILL BE LOST.
17 #
18 # However, if you remove everything, the rebase will be aborted.
19 #
20 # Note that empty commits are commented out
```

# git rebase -i: demo

You can see that the commit history has been changed. Now, it's time to push commits into remote repo.

```
(ml3) sanghee@sp:~/Dev/demo$ git rebase -i 9894eb
Successfully rebased and updated refs/heads/master.
(ml3) sanghee@sp:~/Dev/demo$ git log
commit 114fb29380fae40324da6afbe9f38262f6c423de
Author: Sanghee Kim <sangheestyle@gmail.com>
Date:   Wed May 6 00:14:43 2015 -0600

    second commit

commit 59cde634ca05cacc6c27eb0e78d12577971fe854
Author: Sanghee Kim <sangheestyle@gmail.com>
Date:   Wed May 6 00:59:16 2015 -0600

    third commit

commit 9894eb970b435d60af846f4b171637acbbb4e037
Author: Sanghee Kim <sangheestyle@gmail.com>
Date:   Wed May 6 00:14:05 2015 -0600

    first commit
```

# git stash

What if you need to update your repository or switch working branch but you have some changes in a working directory? You can use git stash for stashing the changes in a dirty working directory away.

$ git stash

$ git stash pop

# git stash: demo

You cloned remote repo on your local and modified files which are not committed yet. At the sametime, someone pushed a new commit into remote repo. Let's try to update your local repo.

```
(ml3) sanghee@sp:~/Dev/remote_demo$ git log
commit 3cae2e67e4c5dd745d511f9e8b756c0e6a0e6486
Author: Sanghee Kim <sangheestyle@gmail.com>
Date:   Wed May 6 01:12:28 2015 -0600

    fourth commit

commit 114fb29380fae40324da6afbe9f38262f6c423de
Author: Sanghee Kim <sangheestyle@gmail.com>
Date:   Wed May 6 00:14:43 2015 -0600

    second commit

commit 59cde634ca05cacc6c27eb0e78d12577971fe854
Author: Sanghee Kim <sangheestyle@gmail.com>
Date:   Wed May 6 00:59:16 2015 -0600

    third commit

commit 9894eb970b435d60af846f4b171637acbbb4e037
Author: Sanghee Kim <sangheestyle@gmail.com>
Date:   Wed May 6 00:14:05 2015 -0600

    first commit
```

# git stash: demo

Here is the setup for this demo. Left side is remote and right side is your local repo.

```
(ml3) sanghee@sp:~/Dev/remote_demo$ git log
commit 3cae2e67e4c5dd745d511f9e8b756c0e6a0e6486
Author: Sanghee Kim <sangheestyle@gmail.com>
Date:   Wed May 6 01:12:28 2015 -0600

    fourth commit

commit 114fb29380fae40324da6afbe9f38262f6c423de
Author: Sanghee Kim <sangheestyle@gmail.com>
Date:   Wed May 6 00:14:43 2015 -0600

    second commit

commit 59cde634ca05cacc6c27eb0e78d12577971fe854
Author: Sanghee Kim <sangheestyle@gmail.com>
Date:   Wed May 6 00:59:16 2015 -0600

    third commit

commit 9894eb970b435d60af846f4b171637acbbb4e037
Author: Sanghee Kim <sangheestyle@gmail.com>
Date:   Wed May 6 00:14:05 2015 -0600

    first commit
```

```
(ml3) sanghee@sp:~/Dev/demo$ vim b.txt
(ml3) sanghee@sp:~/Dev/demo$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   b.txt

no changes added to commit (use "git add" and/or "git commit -a")
(ml3) sanghee@sp:~/Dev/demo$ git log
commit 114fb29380fae40324da6afbe9f38262f6c423de
Author: Sanghee Kim <sangheestyle@gmail.com>
Date:   Wed May 6 00:14:43 2015 -0600

    second commit

commit 59cde634ca05cacc6c27eb0e78d12577971fe854
Author: Sanghee Kim <sangheestyle@gmail.com>
Date:   Wed May 6 00:59:16 2015 -0600

    third commit

commit 9894eb970b435d60af846f4b171637acbbb4e037
Author: Sanghee Kim <sangheestyle@gmail.com>
Date:   Wed May 6 00:14:05 2015 -0600

    first commit
```

# git stash: demo

After git pull, you can find that the merge was not able to be done.

```
(ml3) sanghee@sp:~/Dev/demo$ git pull
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From /home/sanghee/Dev/remote_demo
   114fb29..3cae2e6  master     -> origin/master
Updating 114fb29..3cae2e6
error: Your local changes to the following files would be overwritten by merge:
        b.txt
Please, commit your changes or stash them before you can merge.
Aborting
(ml3) sanghee@sp:~/Dev/demo$ git status
On branch master
Your branch is behind 'origin/master' by 1 commit, and can be fast-forwarded.
  (use "git pull" to update your local branch)
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   b.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

# git stash: demo

Let’s do git stash and git pull. There has no problem. So, it’s time to recover the changes.

```
(ml3) sanghee@sp:~/Dev/demo$ git stash
Saved working directory and index state WIP on master: 114fb29 second commit
HEAD is now at 114fb29 second commit
(ml3) sanghee@sp:~/Dev/demo$ git pull
Updating 114fb29..3cae2e6
Fast-forward
 b.txt | 1 +
 1 file changed, 1 insertion(+)
(ml3) sanghee@sp:~/Dev/demo$ git stash pop
Auto-merging b.txt
CONFLICT (content): Merge conflict in b.txt
```

However, there has merge conflict in b.txt because you modified same chunk of the new commit from the remote.

# git stash: demo

But, it’s not commit. So, you just open the conflicted file and modify it on your own.

```
1 ccccccccc
2 <<<<<<< Updated upstream
3 ddddddddd
4 ======
5 eeeeeeeeee
6 >>>>>>> Stashed changes
```

```
1 ccccccccc
2 ddddddddd
3 eeeeeeeee
```

# git stash: demo

Then save it and make a commit. That's it.

```
(ml3) sanghee@sp:~/Dev/demo$ git add b.txt
(ml3) sanghee@sp:~/Dev/demo$ git commit -m "fifth commit"
[master 0c0189c] fifth commit
 1 file changed, 1 insertion(+)
(ml3) sanghee@sp:~/Dev/demo$ git log
commit 0c0189c2b1a9c1c00abe3970bcd10d873059bac3
Author: Sanghee Kim <sangheestyle@gmail.com>
Date:   Wed May 6 01:30:39 2015 -0600

    fifth commit

commit 3cae2e67e4c5dd745d511f9e8b756c0e6a0e6486
Author: Sanghee Kim <sangheestyle@gmail.com>
Date:   Wed May 6 01:12:28 2015 -0600

    fourth commit

commit 114fb29380fae40324da6afbe9f38262f6c423de
Author: Sanghee Kim <sangheestyle@gmail.com>
Date:   Wed May 6 00:14:43 2015 -0600

    second commit

commit 59cde634ca05cacc6c27eb0e78d12577971fe854
Author: Sanghee Kim <sangheestyle@gmail.com>
Date:   Wed May 6 00:59:16 2015 -0600

    third commit

commit 9894eb970b435d60af846f4b171637acbbb4e037
Author: Sanghee Kim <sangheestyle@gmail.com>
Date:   Wed May 6 00:14:05 2015 -0600

    first commit
```

# git log -G

What if you need to find some differences whose patch text contains added/removed lines that match <regex>. git log -G might be helpful.

$ git log -G<regex>

# git log -G: demo

Look at the result of git log -G'aaa' -p. It shows only a commit which contains string 'aaa' in it's own patch text

```
(ml3) sanghee@sp:~/Dev/demo$ git log --grep 'first'
commit 9894eb970b435d60af846f4b171637acbbb4e037
Author: Sanghee Kim <sangheestyle@gmail.com>
Date:   Wed May 6 00:14:05 2015 -0600

    first commit
```

```
(ml3) sanghee@sp:~/Dev/dem
commit 9894eb970b435d60af8
Author: Sanghee Kim <sangh
Date:   Wed May 6 00:14:05

    first commit

diff --git a/a.txt b/a.txt
new file mode 100644
index 0000000..69dd9b9
--- /dev/null
+++ b/a.txt
@@ -0,0 +1 @@
+aaaaaaaaaa
(ml3) sanghee@sp:~/Dev/demo$ git log -G'first' -p
(ml3) sanghee@sp:~/Dev/demo$
```

One of interesting point is that it does not consider commit message at all.

# git log -G: demo

Instead, you can use git log --grep <regex> for searching commits by commit message.

```
(ml3) sanghee@sp:~/Dev/demo$ git log --grep 'first'
commit 9894eb970b435d60af846f4b171637acbbb4e037
Author: Sanghee Kim <sangheestyle@gmail.com>
Date:   Wed May 6 00:14:05 2015 -0600

    first commit
```

# gitk

gitk is the Git repository browser. It displays changes in a repository or a selected set of commits. It is maintained by git project and simple but strong enough to show you the status of your current repository.

$ gitk

# gitk: demo

To use gitk, you might need to install gitk.

$ sudo apt-get install gitk

```
(ml3) sanghee@sp:~/Dev/demo$ sudo apt-get install gitk
```

# gitk

Just run gitk and it shows you commit history graphically. It’s really simple and lightweight.