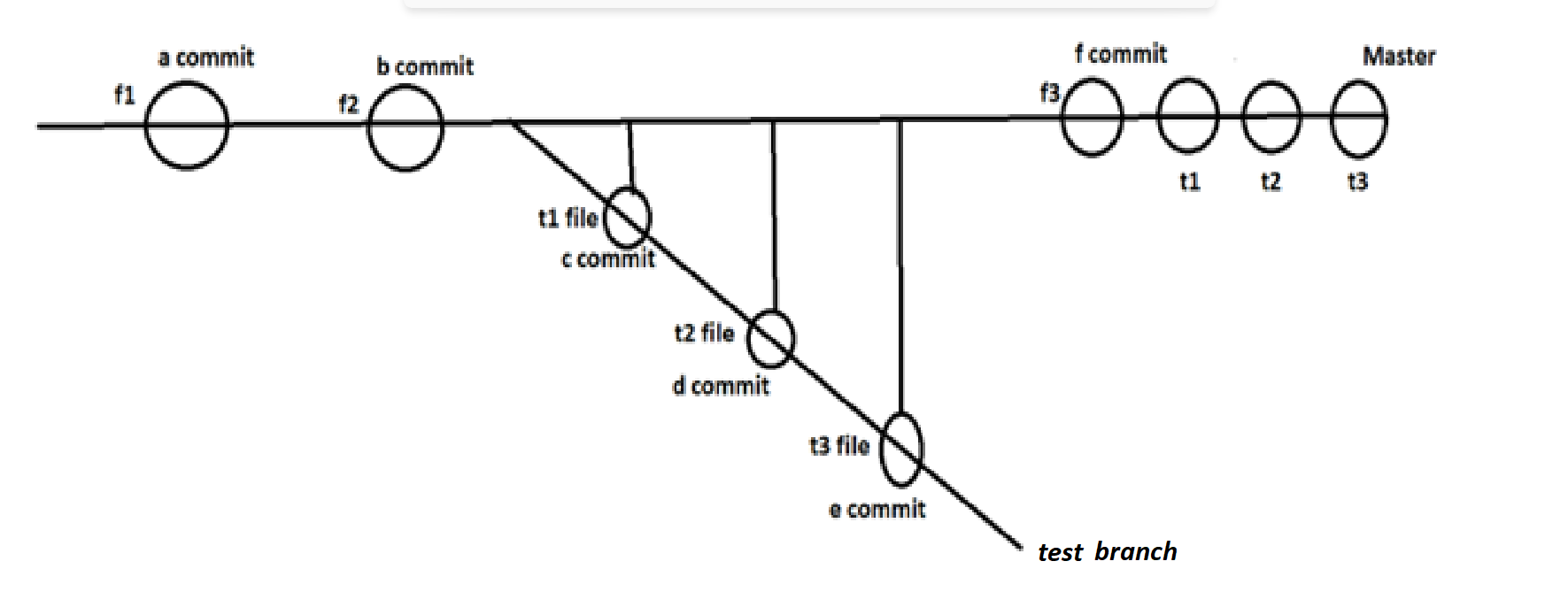
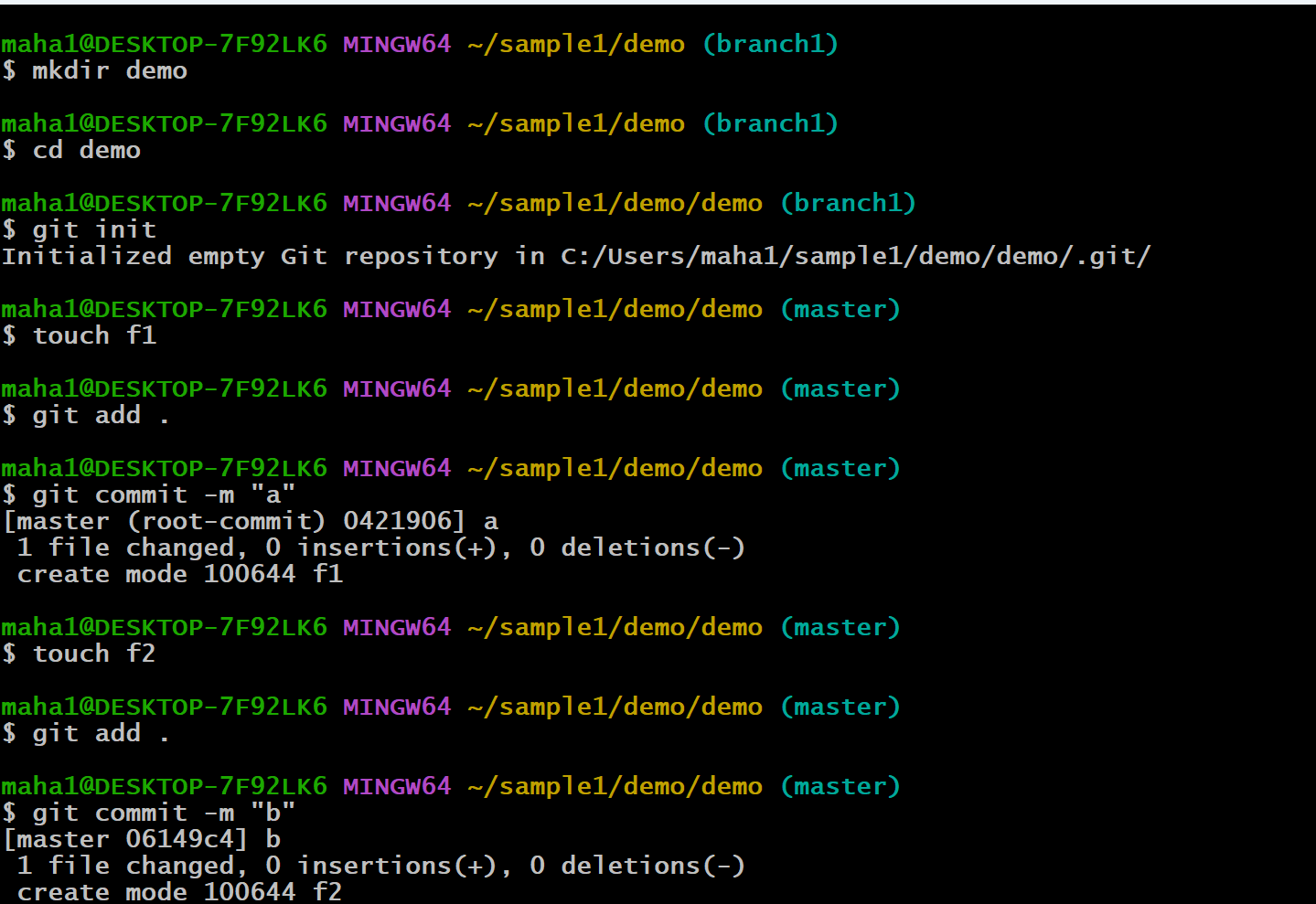
**git rebase**

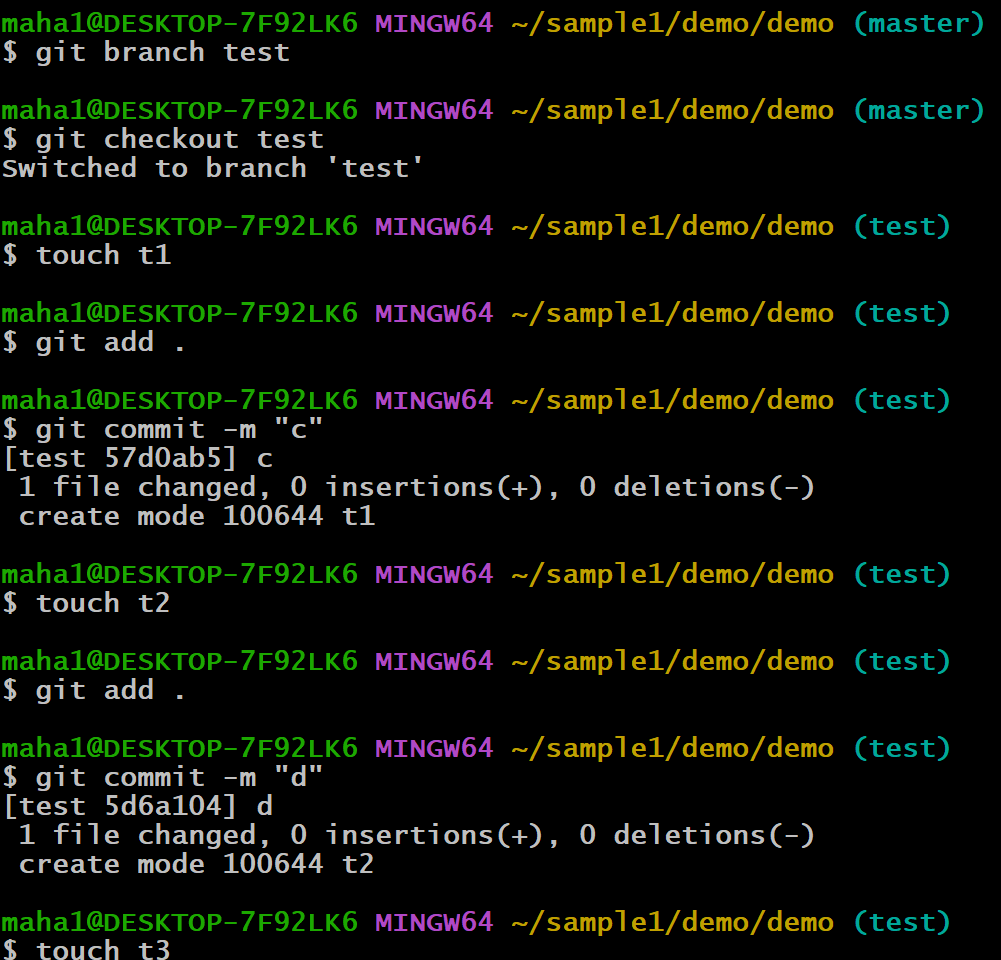
* This is called as fastforward merge.
* The commits from the child branch are added to the top of the master branch.
* This is helpful when we want code from a branch to be reflected as the latest working version on master.

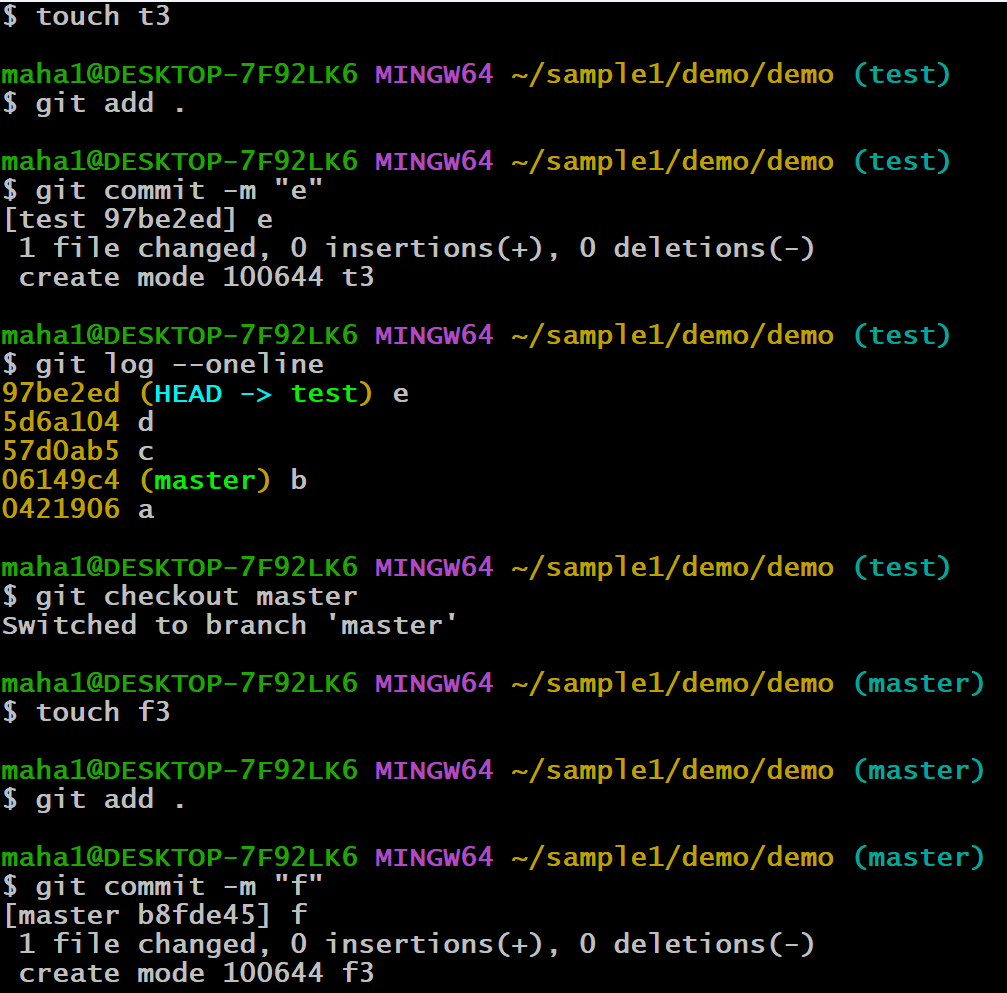
Let’s see the first functionality of rebase: fastforward merge.

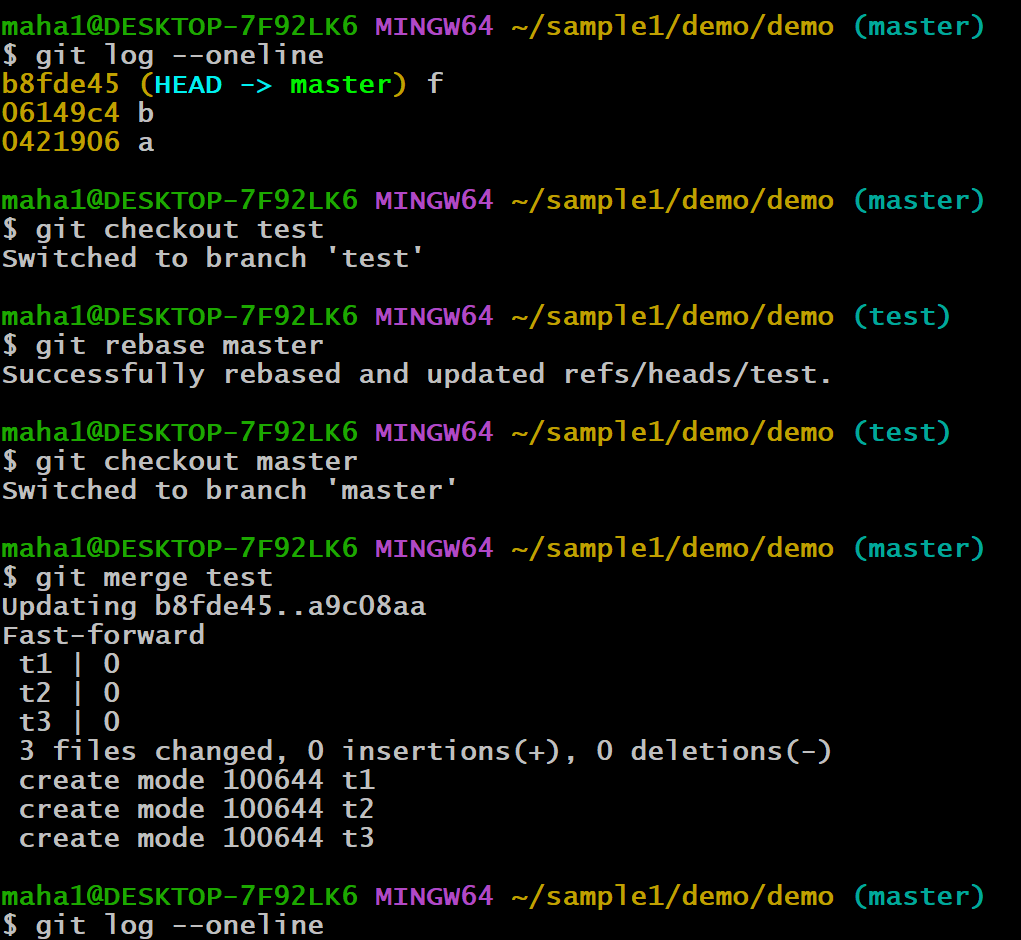


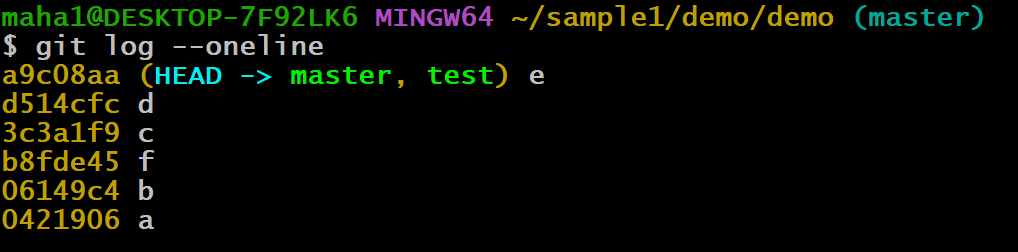
When we wants to rebase we have to come to child branch and then rebase.





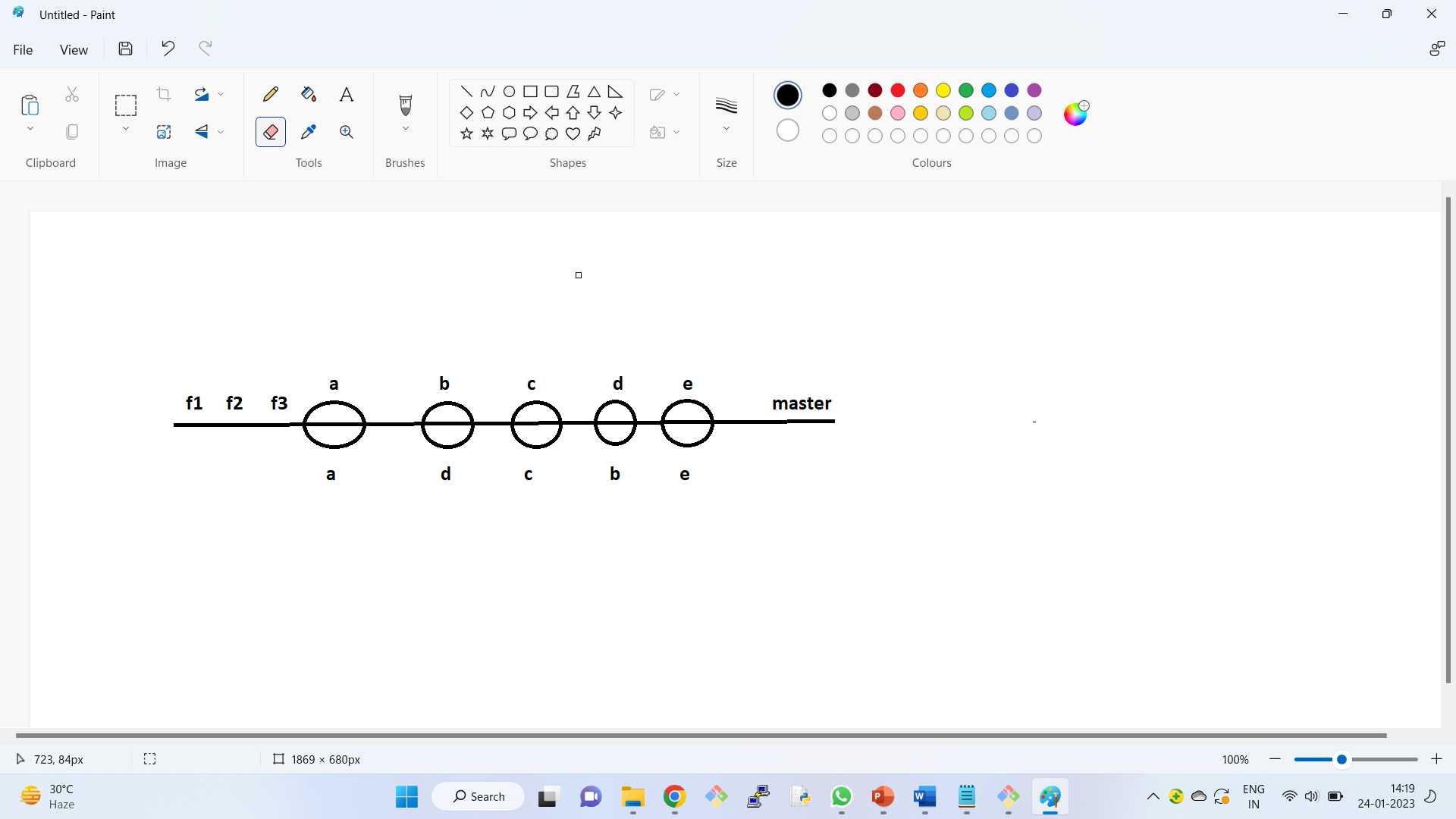


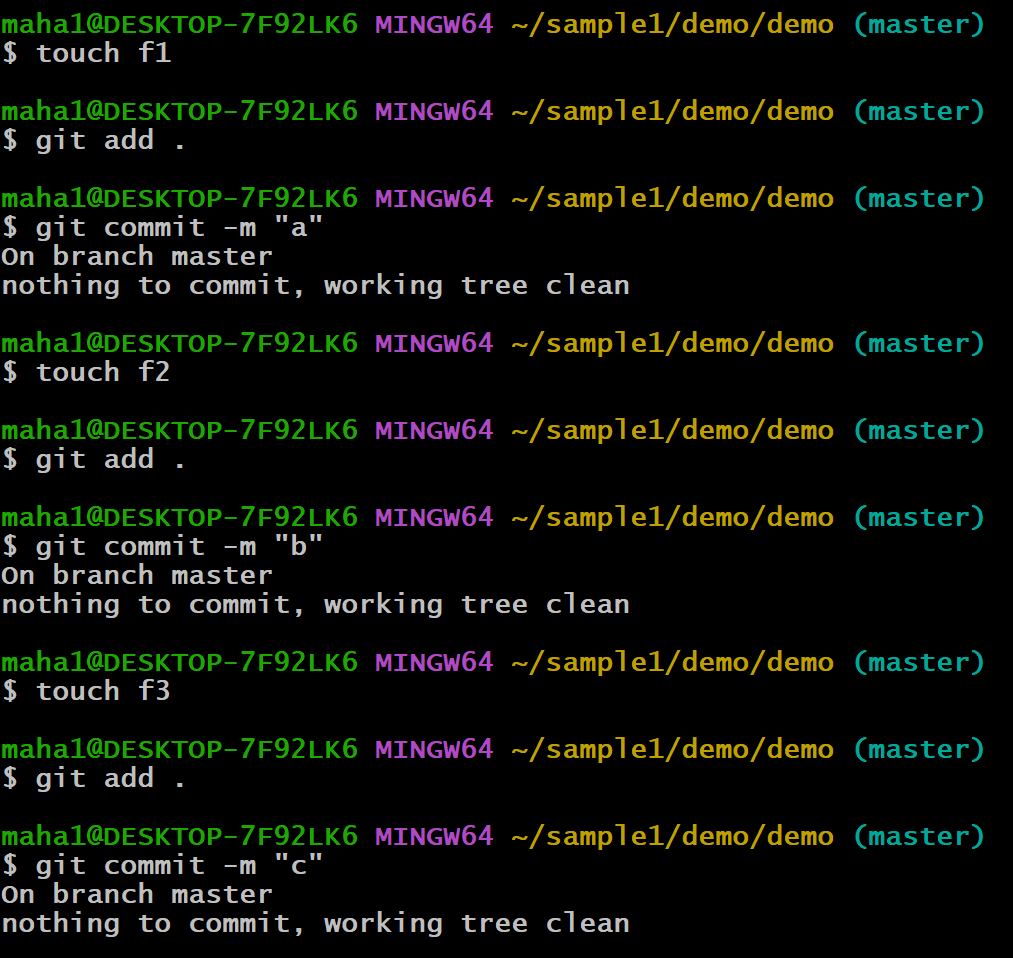




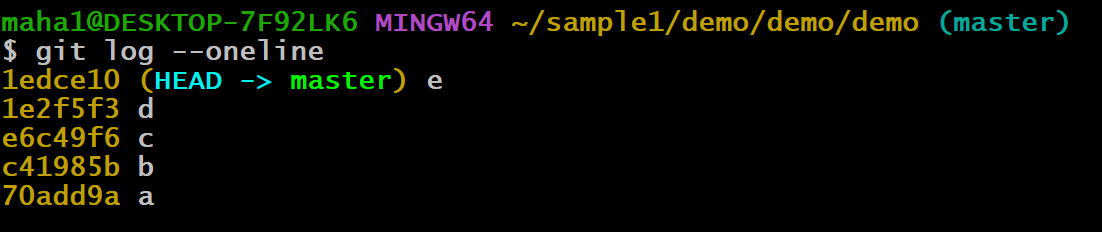
Let’s see the second functionality of rebase:

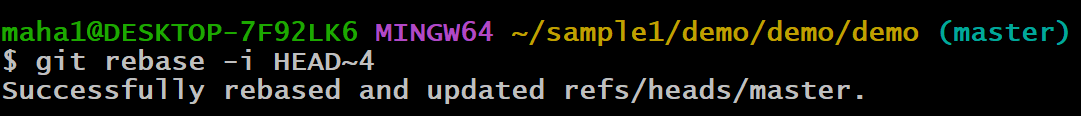
**Rearrange the commit history**

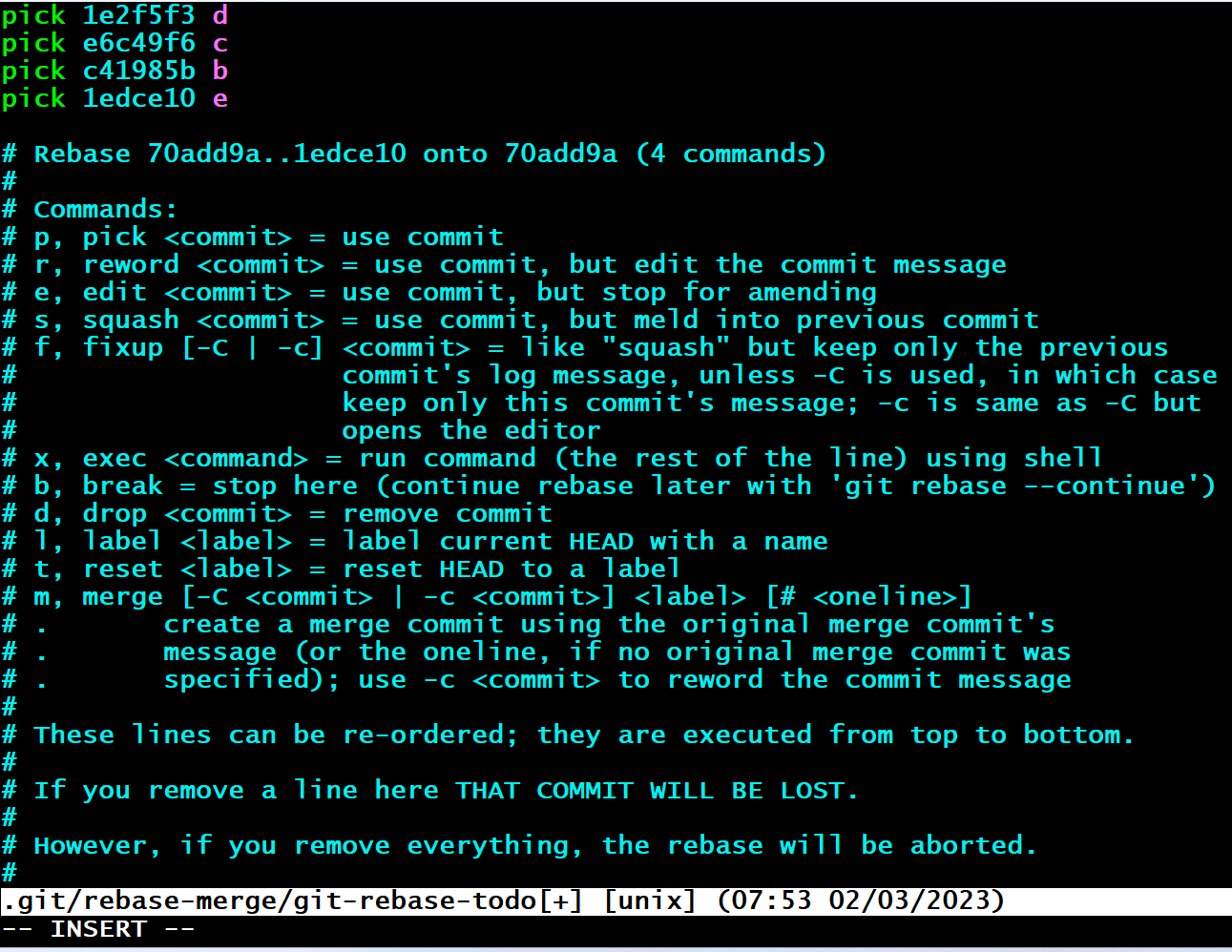


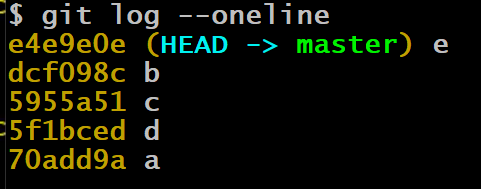












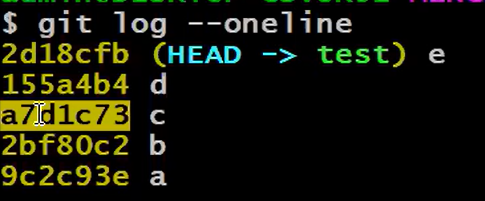
Git rebase -I HEAD~4 (it can change commit order and it can uncommit by using squash) The first commit ‘a’ cannot be removed, it is fixed.

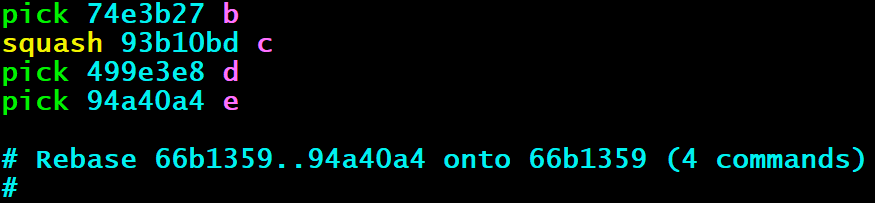
1. **Merging the commits using “squash”**

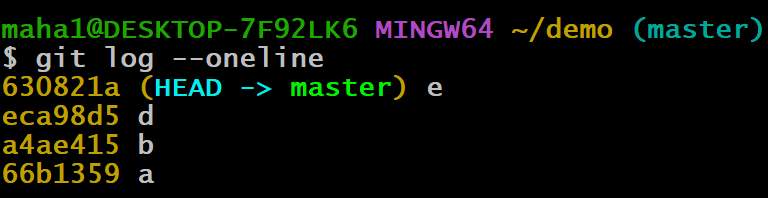
$git rebase -i HEAD~4

Remove pick word and replace “squash”.

Squash means stop something from happening

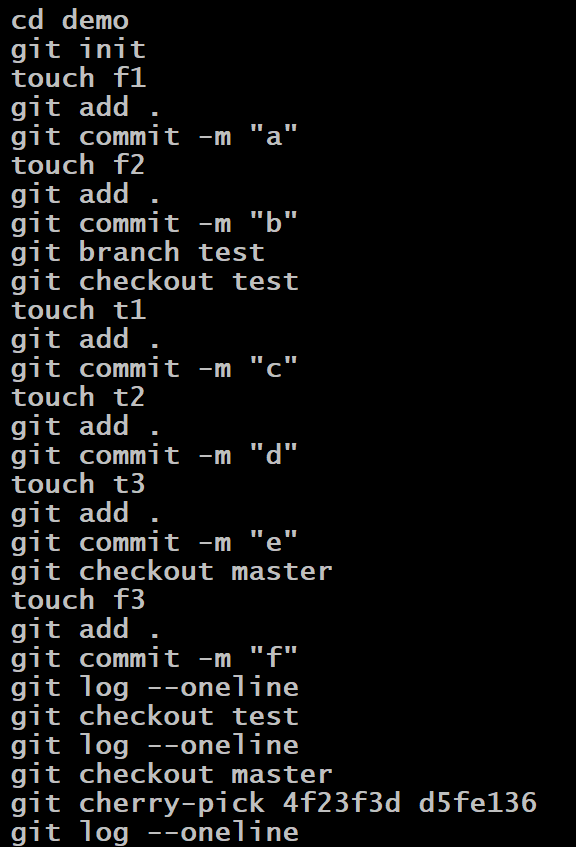






1. **How to selectively pickup the commits from child branch(cherry-pick)**

$git cherry-pick commit id’s

****

**Difference between rebase and cherry-pick:**

* Rebase is a command that allows you to integrate changes from one branch into another by moving the entire branch to a new base commit.
* Cherry-pick, on the other hand, is a command that allows you to select one or more specific commits from one branch and apply them to another branch. This can be useful when you want to selectively apply changes from one branch to another without affecting the commit history.

**Git Stash:**

* stash is a feature is used for leaving unfinished work, in such a way that git cannot access it and continue work on some other files.
* Sometimes you want to switch the branches, but you are working on an incomplete part of your current project.
* You don’t want to make a commit of half-done work. Git stashing allows you to do so.
* Normally when you switch branch you will commit the code and switch to the new branch.
* If you switch branch without committing. Two things will happen.

1. Switches to the branch carrying the changes.
2. Git will not allow to switch the branch and asks to commit or stash the changes.

The git stash command enables you to switch branches without committing the current branch.

The stash’s meaning is “to store something safely in a hidden place.” The sense in Git is also the same for stash; Git temporarily saves your data safely without committing.

* To stash the staged files

$ git stash

* To stash staged and untracked files

$ git stash -u

* To see the list of stashes

$ git stash list

* To get back the stashed files

$ git stash pop

* To bring the older stash out

$ git stash pop stash@stash\_number

Commands:

git init

touch f1 f2

git add .

git commit -m “a”

touch f3 f4

git add .

touch f5 f6

git status (2 staged file, 2 untracked files)

git stash

git status (2 untracked files)

git add .

git commit -m “b”

git status (working tree is clean)

git stash pop

git status (2 staged files)

git commit -m “c”

touch f7 f8

git add .

touch f9 f10

git status (2 staged, 2 untracked)

git stash -u

git status (working tree is clean)

git commit –amend -m “a” (it avoids new commit and we can use existing commit)

git reset –hard “commit id” (rollback to previous command, discards all changes in the working directory and resets the repository to the state of the specified commit)

**Git Tags**

Tagging in git or any other VCS refers to creating specific points in history for your repository/data. This is usually done to mark release points. (v1.0, v1.1, …)

Why we create tags?

To mark release points for your code/data.

To create historic restore points.

When to create tags?

When you want to create a release point for a stable version of your code.

When you want to create a historic point for your code/data that you can refer at any future time. (To restore your data)

**How to create Tags in Git**

**Step 1**: Checkout the branch where you want to create the tag.

**git checkout <branch name>**

**Step 2:** create tag with some name**.**

**git tag <tag name>**

**git tag -a v1.0 -m “ver1 of..” ( to create annotated tags)**

**Step 3:** Display or show tags

git tag

**git show v1.0**

**git tag -l “v1.\*”**

**Step 4:** Push tags to remote

**git push origin v1.0**

**git push origin –tags**

**git push –tags**

(to push all tags at once)

**Step 5:** Delete tags (if required only)

To delete tags from local:

**git tag -d v1.0**

**git tag –delete v1.0**

To delete tags from remote:

**git push origin -d v1.0**

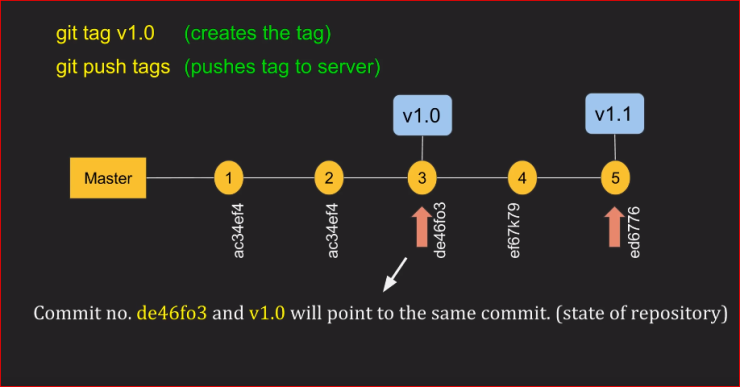
**git push origin --delete v1.0**

**git push origin :v1.0**

To delete multiple tags at once:

**git tag -d v1.0 v1.1 (local)**

**git push origin -d v1.0 v1.1 (remote)**



**How to checkout TAGs**(when required**)**

**We cannot checkout tags in git**

**We can create a branch from a tag and checkout the branch**

**git checkout -b <branch name> <tag name>**

eg: git checkout -b ReleaseVer1 v1.0

**Can I create a tag from some past commit?**

Yes

**Git tag <tag name> <reference of commit>**

Eg: git tag v1.2 5fcdb03