Demo 1 - Creating commits with the plumbing tools

```
# Adds a commit with no parent
git add Foo.txt
git write-tree # Returns tree SHA-1 '1abc'
git commit-tree 1abc -m "No parent" # Returns commit SHA-1 '2abc'
git update-ref refs/heads/master 2abc

git add Foo.txt
# Prints contents of working directory and index,
# at this stage only the blob is displayed
git ls-files --stage
# Creates a tree with the contents of the index
git write-tree
# Creates a commit with the specified tree, message and parent commit
git commit-tree 2abc -p 1abc -m "Message"
# Updates the 'master' branch to point to the newly created commit
git update-ref refs/heads/master 3abc

git cat-file commit 3abc # Prints the commit and the tree
git ls-tree -rt 2abc # Recursively prints the tree under the commit
git cat-file blob 4abc # Prints the content of the file
```

Demo 2 - The Index (a.k.a. “Staging area”)

```
git add Foo.txt
git status
# Modify Foo.txt to see that it's been cached
git diff
git diff --cached
# Makea the Index look like it does in the commit
# referenced by HEAD but leaves the working directory alone,
# effectively unstaging the file
git reset HEAD Foo.txt

# Make multiple changes to the file
# but stage only one hunk (i.e. a patch commit)
git add -p Foo.txt

# Interactive staging
git add -i
```

Demo 3 - Branches, merging, rebasing

```
git branch Experiment
git checkout Experiment
git add Foo.txt
git commit -m "Allows the user to quit the program."

git checkout master
git add Foo.txt
git commit -m "Adds XML docs."

# Create merge commit
git merge Experiment
git log
git log --no-merges

# Remove merge commit, clear working directory and start over
# Soft reset: updates HEAD, but not the Index nor the working directory,
# where both retain the changes from the earlier commit
git reset --soft HEAD^

# Mixed reset: updates HEAD and the Index, but not the working directory,
# which still retains has the changes from the earlier commit
git reset HEAD

# Hard reset: updates HEAD, the Index and the working directory,
thus eliminating the changes from the previous commit
git reset --hard master

# Rebase 'Experiment' on top of 'master'
git checkout Experiment
git rebase master
git checkout master

# Create a fast-forward commit obtaining a linear history
git merge Experiment
```

Demo 4 - Rewriting history

```
# Note that modifying commits changes their SHA-1.
# This should not be done for commits that have been shared with others
# since Git will treat them as totally different commits, even though they
# contain the same snapshots

# Correct the last commit (amending)
# First, moves HEAD to the parent commit and updates the Index
# but leaves the working directory untouched
git reset HEAD^
git add Foo.txt
git commit -m "Adds valuable comment"

# Corrects the message and author in the last commit
git commit --amend -m "Better message" --author "Pippo <pippo@disney.com>"

# Interactive rebase
# starting from 4 commits before HEAD
git rebase -i HEAD~4
# Demonstrate reword, squash, reorder and delete
```

Demo 5 - Remotes

```
# Clone the repo locally into another directory
cd ..
git clone --local Demo Pippo
cd Pippo
# Lists the available remotes
git remote
git remote show origin

git branch
git branch --all
# Creates local tracking branch
# that is associated to a remote branch
git checkout --track origin/Experiment
gid add Foo.txt
git commit -m "Bold change"
git push origin

cd ..\Demo
git checkout master
git add Foo.txt
git commit -m "Bugfix"

cd ..\Pippo
git checkout master
git fetch origin
# Shows the commits that are in the remote 'master' branch
# but not in the local 'master'
git log master..origin/master
git merge origin/master

# Fetch and merge can be done in one go with 'pull'
```

Demo 6 - Bisect

```
# Mark the starting point of a bisect session
git bisect start
# Mark that the current commit has a bug
git bisect bad
# Mark that 5 commits ago things were good
git bisect good HEAD~5

# Git moves HEAD to the commit in the middle between the good and bad ones
# Run your tests
# Declare the current commit bad
git bisect bad

# Git moves HEAD to the commit in the middle between the current commit and the good one
# Run your tests
# Declare the current commit good
git bisect good

# Git moves HEAD to the commit in the middle between the current commit and the previous middle one
# and so on until one commit is left, that's were the bug sneaked in.
# Go back to the start commit
git bisect reset
```