Using Git and Github
Version Control With Git

Level: Intermediate
Author : WIN HTUT
Org: National Institute of Science and Technology
Series: Part I

Version control systems(VCSs) ဆိုတာ project တစ်ခုလုံးရဲ့ version တွေကို ထိန်းချုပ်စီမံ ခြင်း ဖြစ်ပါတယ်။ ဥပမာ စာရေးသူတို့ အနေဖြင့် shopping website တစ်ခု ရေးသည် ဆိုပါစို့ ပထမဆုံး version တွင် shopping website အတွက် ငွေချေဖို့ ဘဏ်နဲ့ တိုက်ရိုက် ချိတ်ဆက် ထားတာမျိုး မရှိပါ ဒုတိယ version မှသာ ဘဏ်နဲ့ တိုက်ရိုက်ချိတ်ဆက် ငွေချေတဲ့ အပိုင်းကို ထည့်လိုက်ပါတယ်။ ထိုနည်းတူ တတိယ version မှာတော့ ပထမ version မှ မလိုအပ်တော့ တာတွေ ဖြုတ်လိုက်သလို လိုအပ်တဲ့ data အချက်လက် များကိုလည်း ထပ်ထည့်သည် ဆိုပါစို့ စာရေးသူတို့ shopping website တွေ version 3 ခု ရှိသွားပါပြီ။ Git ဖြင့် version control လုပ်ခြင်းသည် ထို versions 3 ခုလုံးတွင် ထပ်တိုးခြင်း(addition) , ပြုပြင်ခြင်း(edition) , နှင့် ဖျက်ထုတ်ခြင်း (remove) များကို မှတ်တမ်း တင်ပေးထားခြင်း ဖြစ်သည်။ အကျိုးကျေးဇူးအားဖြင့် တတိယ version တွင် error တက်လာပါကလည်း ဒုတိယ version သို့ အလွယ်တကူ ပြန်ချိန်းနိုင်သလို version တစ်ခုနှင့် တစ်ခုကြား အပြောင်းအလဲ များကိုလည်း အသေးစိတ် သိနိုင်သည်။

Version Control System VCS မှာ နှစ်မျိုးရှိပါတယ် Git and CVS တို့ ဖြစ်ပါတယ်။ Git သည် Distributed VCS ဖြစ်ပြီး CVS သည် Centralized Version Control System ဖြစ်ပါတယ်။ သူတို့ နှစ်ခုကြားမှာ အဓိက ခြားနားချက်ကတော့ Centralized VCS သည် programmer တွေရဲ့ program ပြောင်းလဲမှုတွေအားလုံးကို server တစ်ခုတည်းပေါ်မှာ သိမ်းထားပါတယ်။ ဆိုလိုချင်တာကတော့ programmer တွေဟာ သူတို့ရဲ့လုပ်ဆောင်ချက်တွေကို အခြားသူတွေနဲ့ sharing လုပ်ထားပါတယ် သို့သော် server နဲ့ connection ပြတ်တောက်သွားရင်တော့ ဆက်လုပ်လို့ မရတော့ပါဘူး။

Distributed VCS ကတော့ project တစ်ခုလုံးနှင့် အတူ history အားလုံးကို မိမိတို့ရဲ့ computer ထဲမှာ local copy အနေနဲ့ သိမ်ထားပါတယ် ထို့ကြောင့် project ထဲမှာ ထပ်ပေါင်းတာတွေ စတဲ့ ပြုပြင် ပြောင်းလဲ မှုတွေ အတွက် online ဖြစ်နေစရာ မလိုပါဘူး။

Git ကို အသုံးပြုဖို့ အတွက် အောက်ပါ နည်းလမ်းတွေကို အသုံးပြုနိုင်ပါတယ်။

1. Command line

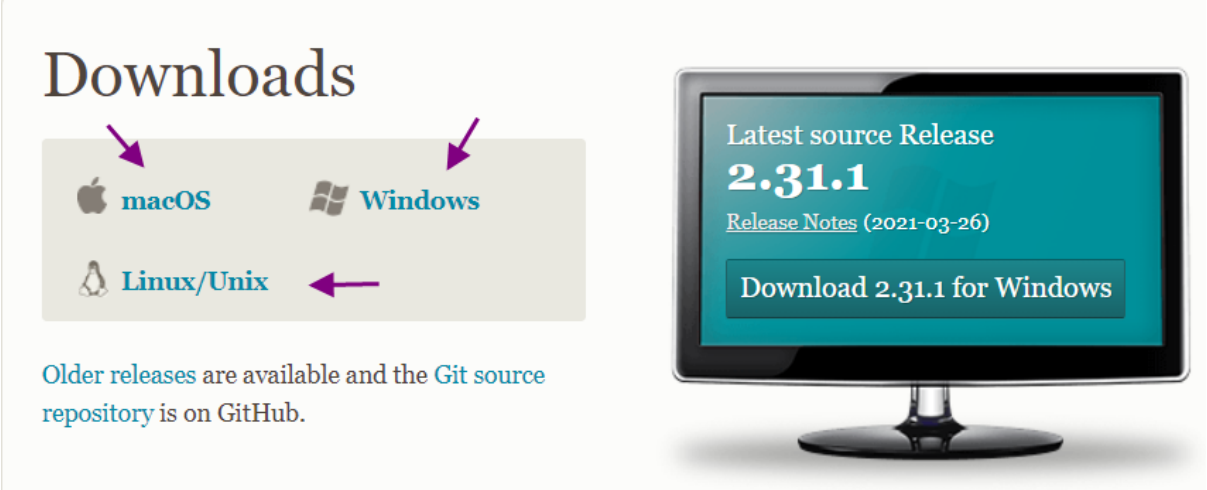
   Terminal သို့မဟုတ် command promt ကို အသုံးပြုပြီး git commands တွေကို ရေးသားကာ အသုံးပြုနိုင်ပါတယ်။ Git ကို အသုံးပြုတဲ့ နည်းတွေထဲမှာ ယခု

နည်းကတော့ အမြန်ဆုံး ဖြစ်ပြီး အလုပ်ကို မြန်မြန် ပြီးမြောက်စေနိုင်ပါတယ် ထို့ကြောင့် programmer အများစုဟာ git ကို command line ကနေ တစ်ဆင့် အသုံးပြုကြပါတယ်။

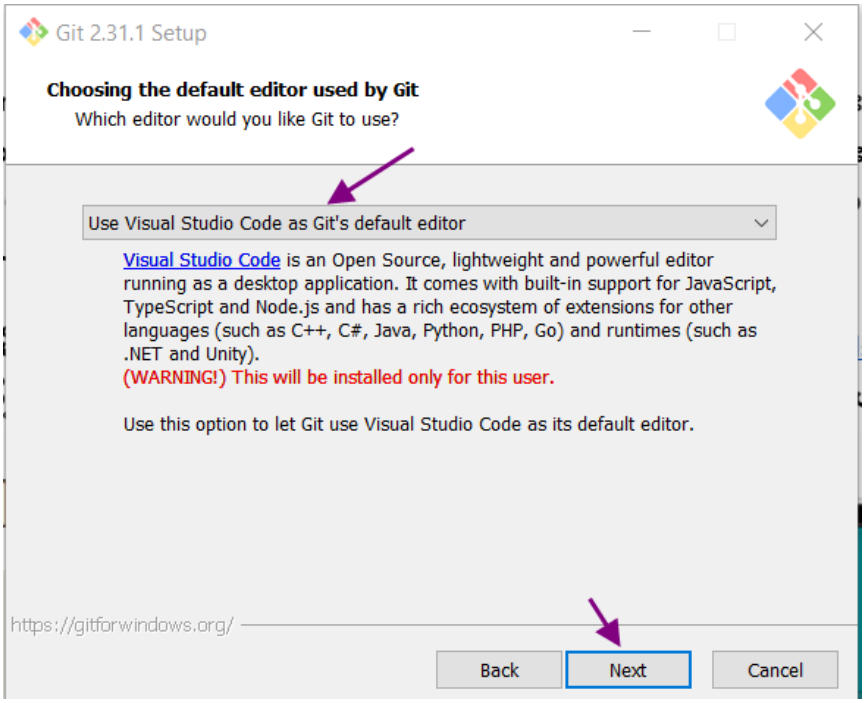
2. Code editors and IDEs အကယ်၍ သင်ဟာ command line ကနေ အသုံးပြုရတာ မကြိုက်ဘူး မရင်းနှီးဘူး ဆိုလျှင် ယနေ့ခေတ် code editors and ide တွေမှာ git ရဲ့ features အတော်များများကို တပါတည်း တွဲထည့်ပေးလာကြပါတယ်။ဥပမာ Vistudio code မှာ git ရဲ့ features တွေ ကို အသုံးပြုနိုင်ဖို့ ပါဝင်လာသလို GitLens ဆိုတဲ့ extension ကို အသုံးပြုပြီးတော့လည်း git ရဲ့ features များစွာကို အသုံးပြုနိုင်ပါသေးတယ်။
3. GUI(Graphical User Interface) အထက်ပါ နည်းလမ်း နှစ်ခုလုံးကို မကြိုက်ပဲ GUI နဲ့သာ အသုံးပြုလိုတယ် ဆိုလျှင်တော့ https://git-scm.com/downloads/guis ယခု ဖော်ပြထားတဲ့ နေရာမှာ မိမိတို့ အသုံးပြုသည့် OS အလိုက် download ဆွဲပြီး အသုံးပြုနိုင်ပါသေးတယ်။ GitKraken ကလည်း Git GUI မှာ နံမည်ကြီးတဲ့ tool တစ်ခု ဖြစ်ပါတယ် opensource ဖြစ်ပြီး windows , linux and mac တို့အတွက်လည်း support လုပ်ပေးထားပါတယ် ပိုကောင်းတာ သုံးချင်ရင်တော့ annual fee ပေးသွင်းရပါတယ်။ နောက်ထပ် Git GUI tool တစ်ခုကတော့ Sourcetree ဖြစ်ပြီး သူကတော့ completely free ဖြစ်ပါတယ် သို့သော် windows and mac အတွက်ပဲ support လုပ်ထားပါတယ်။

Installation Git

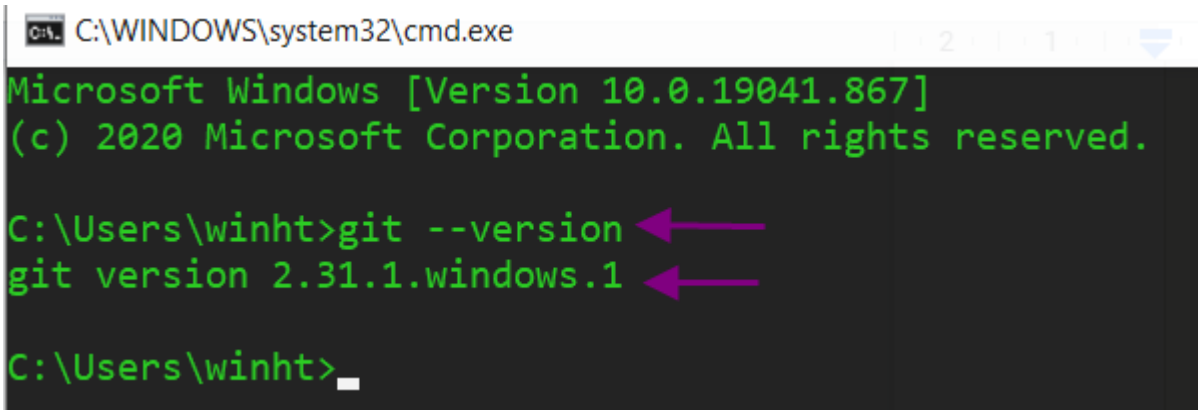
Git ကို မိမိတို့ စက်မှာ install လုပ်ဖို့ အတွက် https://git-scm.com/downloads ယခု Link ကို သွားပြီး မိမိတို့အ သုံးပြုသည့် Operating System အလိုက် download ရယူနိုင်ပါတယ်။



Download ဆွဲလို့ ရလာသော git software အား Installation လုပ်ပေးပါ step အားလုံးတွင် next များကိုသာ နှိပ်ပြီး သွားနိုင်သည် အောက်ပါ အဆင့်ရောက်လျှင်တော့ မိမိတို့ default အနေဖြင့် အသုံးပြုလိုသည့် ide ကို ရွေးပေးနိုင်သည် စာရေးသူသည် visual studio code ကို ရွေးချယ်ထားပါသည်။



ကျန်သည့် step များတွင်လည်း Next ကိုသာနှိပ်ပြီး installationပြီးမြောက်နိုင်ပါတယ်။ Installation အောင်မြင်ခြင်းကို စစ်ဆေးနိုင်ရန် အတွက် terminal or command promt ကိုဖွင့်၍ git --version ဟုရေးကာ စစ်ဆေးကြည့်နိုင်သည်။



Configuration Settings

Configuration setting ဆိုတာ repository တစ်ခု အတွက် name , email , default editor , line ending စတဲ့ အချက်လက်တွေကို သက်မှတ်ပေးတာပါ။ ထို့သို့ သတ်မှတ်ပေးတဲ့ နေရာမှာလည်း level သုံးခု ရှိပါတယ် ။

```
C:\Users\winht>git --version
git version 2.31.1.windows.1

C:\Users\winht>git config --global user.name "Win Htut"

C:\Users\winht>git config --global user.email winhtutonline@gmial.com

C:\Users\winht>git config --global core.editor "code --wait"

C:\Users\winht>
```

အထက်တွင် ဖော်ပြထားတဲ့ အတိုင်းရေးပြီး setting များ သတ်မှတ်ပေးနိုင်ပါတယ် အထက်တွင် --global ကိုသုံးထားပါတယ် system , global , local ဆိုသည့် level သုံးခု ရှိပါတယ် system level သည် user အားလုံး အတွက် ဖြစ်ပြီး global သည် ယခုလက်ရှိ အသုံးပြုသူမှ repository အတွက်ဖြစ်ပါတယ်။ Local ကတော့ ယခု လက်ရှိ အသုံးပြုနေတဲ့ repository အတွက်သာလျှင် ဖြစ်ပါတယ်။

Core.editor Line တွင်ရေးထားသော “code --wait” သည် code ဆိုတာ visual studio code ကို ဆိုလိုခြင်း ဖြစ်သည်။ wait ကတော့ visual studio code ကို cmd or terminal မှ တစ်ဆင့် ဖွင့်ပြီး အလုပ်လုပ်သော အချိန်၌ terminal တွင် စောင့်နေစေရန် ဖြစ်သည် ဥပမာကို အောက်တွင် ဖော်ပြထားသည်။ ထို command တစ်ကြောင်းလုံး၏ ဆိုလိုရင်းသည် visual studio code အား default editor အဖြစ် git သို့ အပ်လိုက်ခြင်း ဖြစ်သည်။

```
C:\WINDOWS\system32\cmd.exe - git  config --global -e
Microsoft Windows [Version 10.0.19041.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\winht>git config --global -e
hint: Waiting for your editor to close the file...
```

File Edit Selection View Go Run Terminal Help .gitconfig - Source

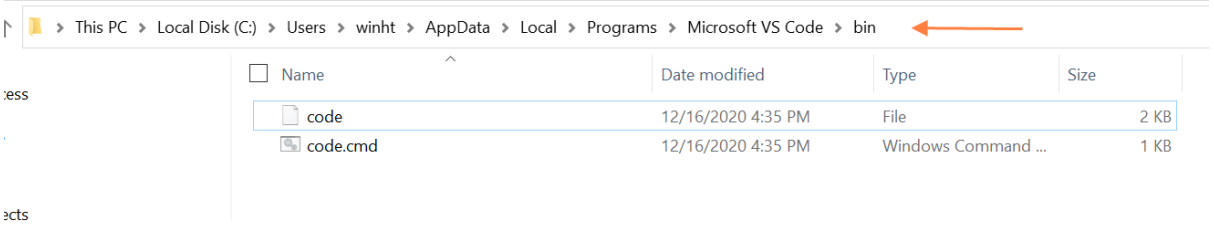
hello.cpp .gitconfig

C: > Users > winht > .gitconfig

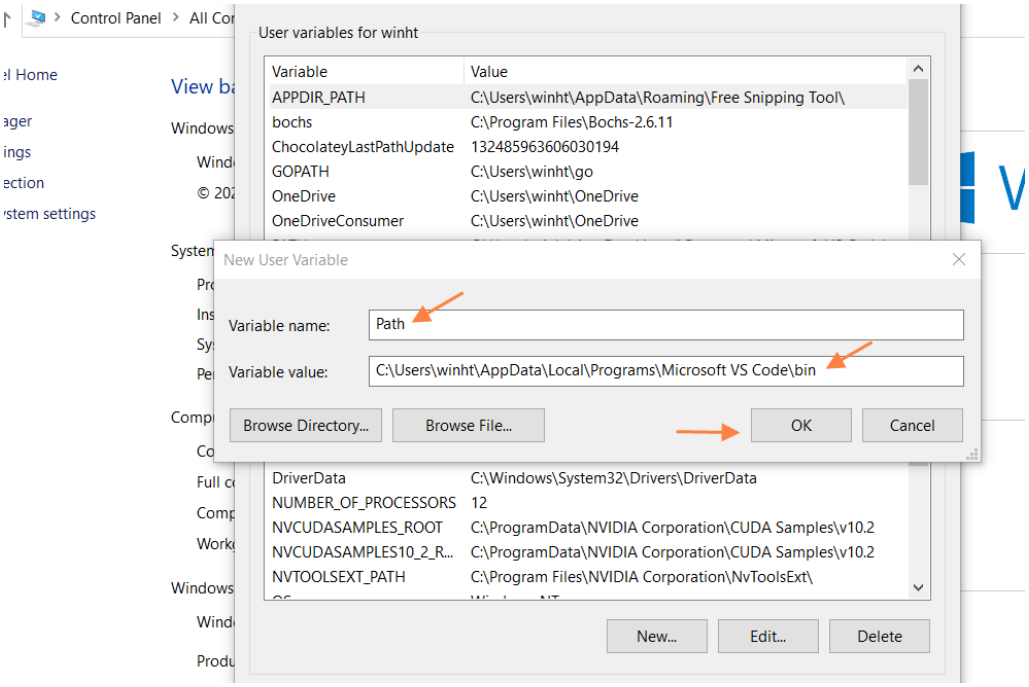
```
1   [core]
2       editor = code --wait
3   [user]
4       name = Win Htut
5       email = winhtutonline@gmial.com
6
```

အထက်ပါပုံတွင် git config --global -e ဟုရေးလိုက်သော အခါတွင် default editor အဖြစ်ပေးထားသည့် visual studio code ပွင့်လာမည်ဖြစ်ပြီး vs code ကို မပိတ်မချင်း hint: Waiting ဆိုသည့် စာသားလေး ပေါ်နေကာ စောင့်နေမည် ဖြစ်သည်။

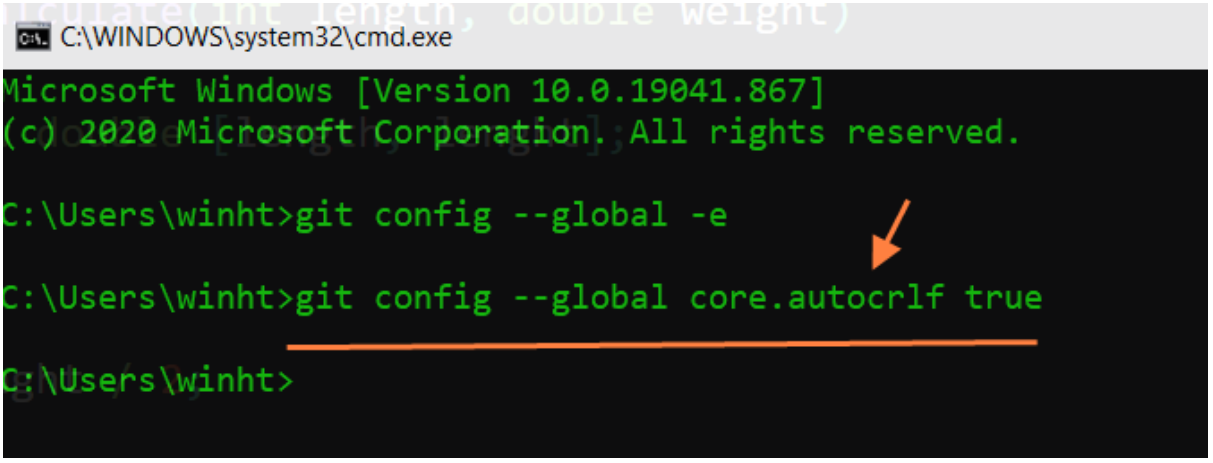
အကယ်၍ terminal or cmd တွင် code ဟု ရိုက်သော်လည်း vs code ပွင့်မလာပါက အောက်ပါပုံထဲက အတိုင်းသွားပြီး system path ထဲတွင် သွားရောက် အပ်ပေးရပါမည်။



C:\Users\winht\AppData\Local\Programs\Microsoft VS Code\bin ယခု လက်ရှိ ဖေါ်ပြ ထားသော စာသာ စာရေးသူ၏ computer အတွက် ဖြစ်ပြီး C:\Users\ နောက်မှ winht နေရာတွင် စာဖတ်သူတို့ရဲ့ computer name our username ဖြစ်ရပါမည်။ အထက်ပါ စာအား copy ကူးပြီး အောက်ပါ ပုံထဲက အတိုင်း envrionment variable ထဲတွင် သွားရောက် အပ်ပေးရမည်။



အထက်ပါ အတိုင်းအပ်ပြီးပါက terminal or cmd ကို restart လုပ်ပြီး code ဆိုတာကို ရေးကြည့်ပါက vs vode ပွင့်လာသည်ကို တွေ့ရပါမည်။ နောက်ထပ် git command အနေဖြင့် new line ကို ဆက်လက် ဖော်ပြပါမည် new line သည် windows တွင် \n (line feed) , \r (carriage return) တို့ဖြစ်ပြီး macOS/Linux တို့တွင် \n ကို အသုံးပြုပါသည်။ git command ကိုရေးရာတွင် windows အတွက် property ကို true ဟုရေးပြီး mac or linux တွင် input ဟုရေးပါသည် သို့မှသာ operating system မတူသော်လည်း line endings ကို ထိန်းညှိနိုင်ဖို့အတွက် အဆင်ပြေပါမည်။ စာရေးသူသည် windows ဖြစ်သည့် အတွက် true ကို အသုံးပြုထားပြီး autocrlf တွင် cr သည် carriage return ဖြစ်ပြီး lf သည် line feed ဖြစ်သည်။

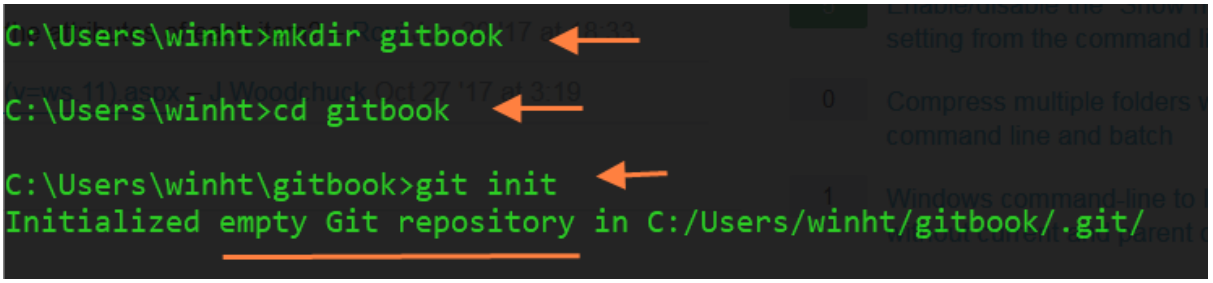


Git config နှင့်ပတ်သက်သော command များကို သိလိုပါက https://git-scm.com/docs/git-config ယခု Link တွေလည်း လေ့လာ နိုင်သလို git config --help ဟုရေးပြီးလည်း ကြည့်ရှု နိုင်သည်။ အကယ်၍ help ကို short term ဖြင့် ကြည့်လိုပါကလည်း git config -h ဟုရေးပြီး ကြည့်နိုင်ပါသည်။

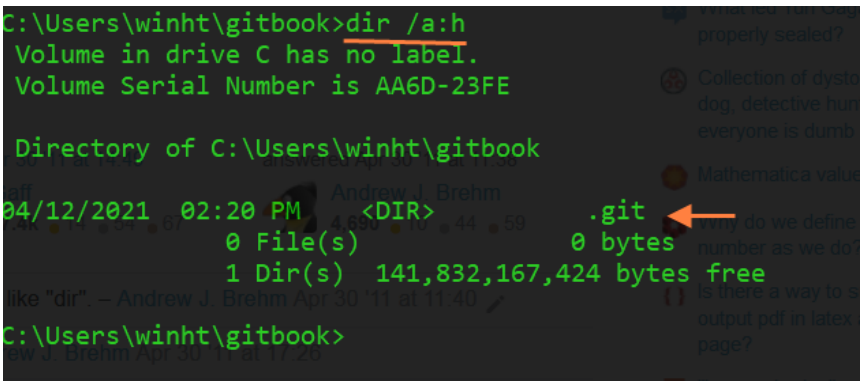
```
C:\Users\winht>git config -h
usage: git config [<options>]

Config file location
    --global              use global config file
    --system              use system config file
    --local               use repository config file
    --worktree            use per-worktree config file
    -f, --file <file>     use given config file
    --blob <blob-id>      read config from given blob object

Action
    --get                 get value: name [value-pattern]
    --get-all             get all values: key [value-pattern]
    --get-regexp          get values for regexp: name-regex [value-pattern]
    --get-urlmatch        get value specific for the URL: section[.var] URL
    --replace-all         replace all matching variables: name value [value-pattern]
    --add                 add a new variable: name value
    --unset               remove a variable: name [value-pattern]
    --unset-all           remove all matches: name [value-pattern]
    --rename-section      rename section: old-name new-name
    --remove-section      remove a section: name
    -l, --list            list all
    --fixed-value         use string equality when comparing values to 'value-pattern'
    -e, --edit            open an editor
    --get-color           find the color configured: slot [default]
    --get-colorbool       find the color setting: slot [stdout-is-tty]
```

Making a repository

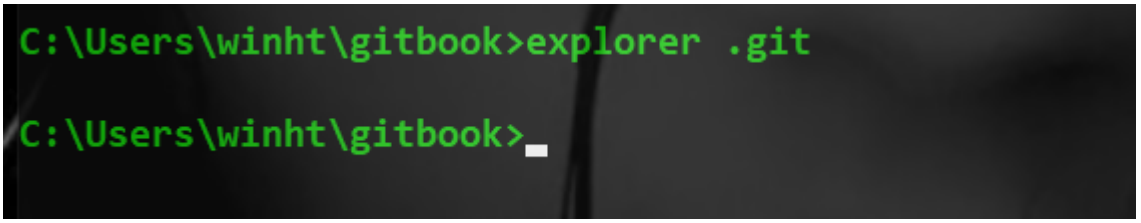
စာရေးသူ အနေဖြင့် gitbook ဆိုသည့် folder တစ်ခု စဆောက်ပါမည် ထို့နောက် gitbook ဆိုသည့် folder ထဲတွင် git repository တည်ဆောက်မည်။ Repository စဆောက်မည့် command သည် git init ဖြစ်သည် ထို့နောက်အောက်ပါ အတိုင်း repo တစ်ခု ဖန်တီးပြီးသည်ကို မြင်ရပါမည်။

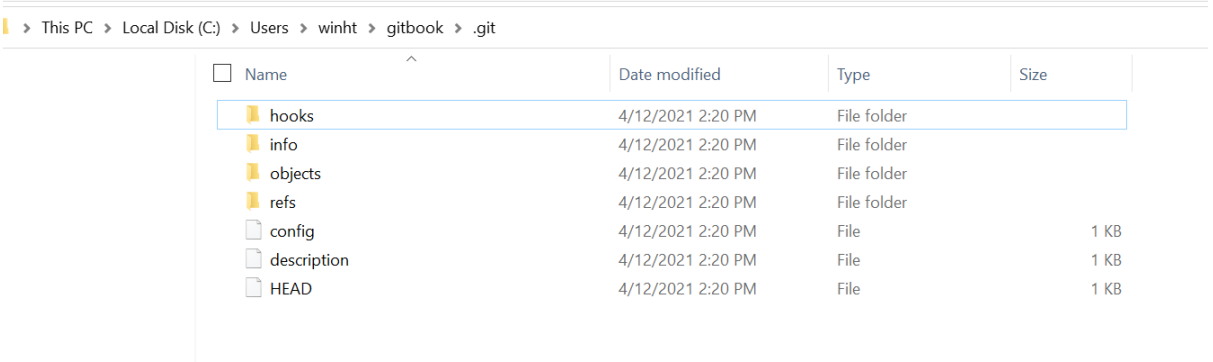


ထို့နောက် dir /a:h ဟု command ရေးကြည့်ပါက .git ဆိုသည့် folder တစ်ခု ရှိနေသည်ကို တွေ့ရမည် linux or mac ဆိုပါက ls -a ဆိုသည့် command ကို အသုံးပြုပါ။



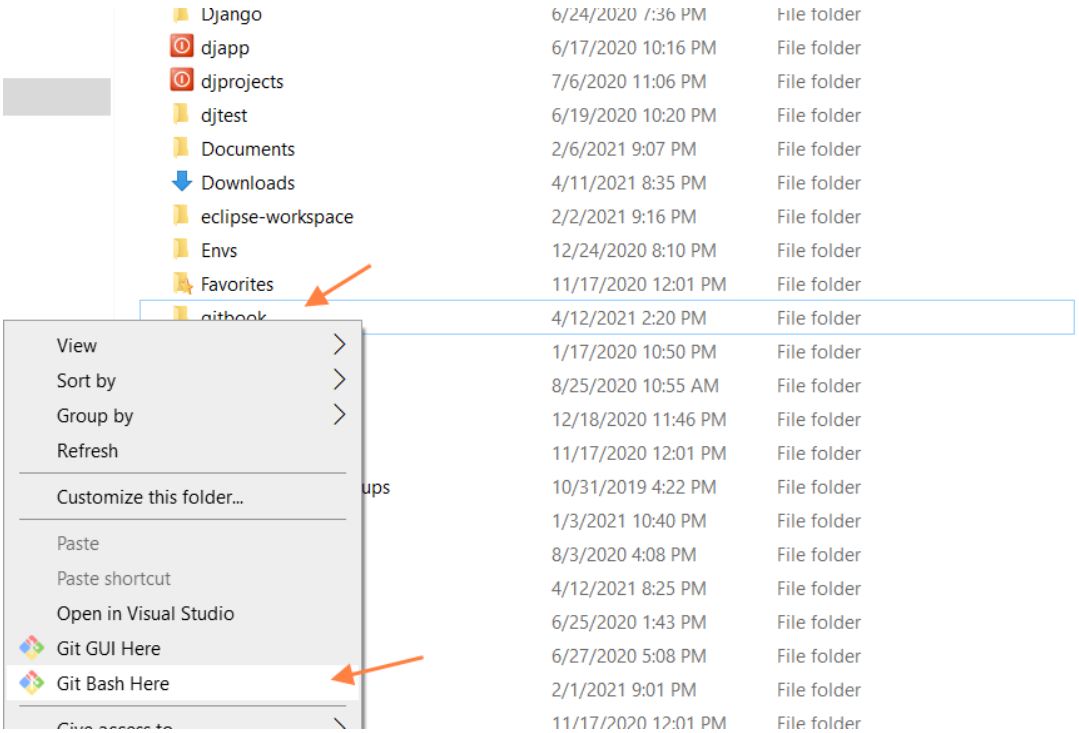
.git folder အား ဖွင့်လိုပါက explorer .git ဆိုသည့် command ကို ရေးပြီး ဖွင့်ကြည့်နိုင်သည် linux or mac တွင်မူ open .git ဟု ရေးပြီး ဖွင့်ကြည့် နိုင်သည်။

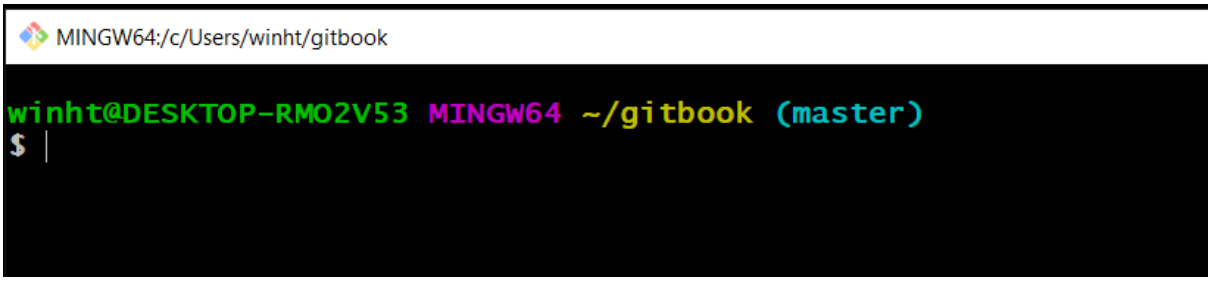




Git Workflow

ယခု lesson ကနေ စတင်ပြီး git ရဲ့ လုပ်ဆောင်ပုံတွေကို ဖော်ပြသွားမှာ ဖြစ်ပါတယ်။ ပထမဆုံး အနေဖြင့် gitbook folder ရှိသည့်နေရာသို့ သွားပါ ထို့နောက် right click ဖြင့် Git Bash Here ဆိုသည့် နေရာအား နှိပ်ပြီး Git bash ကိုဖွင့်လိုက်ပါ။ အောက်ပါ အတိုင်း ပွင့်လာပါမည်။





အထက်ပါ အတိုင်းပေါ်လာပါက gitbook folder ထဲတွင် hello1.txt ဆိုသည့် file တစ်ခုကို အောက်ပါ အတိုင်း တည်ဆောက်ပါမည်။ ထို gitbook folder ရှိသည့် နေရာကို working directory ဟုခေါ်ပါသည်။

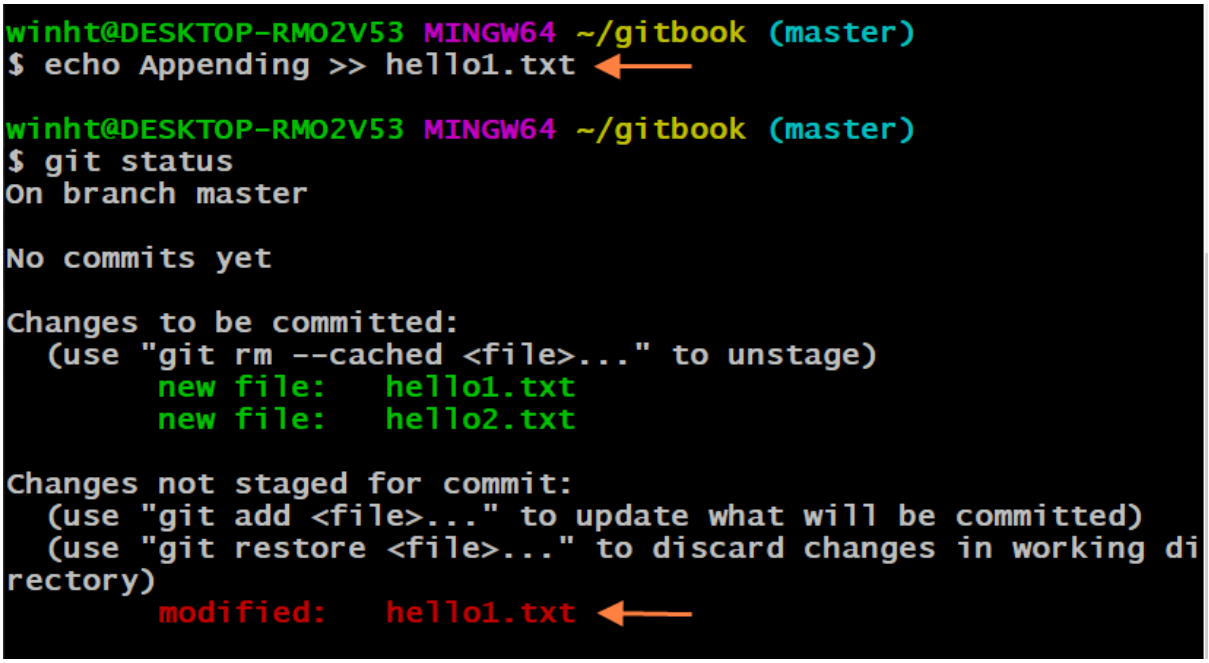
Echo command သည် git command မဟုတ်ပါဘူး echo သည် linux command ဖြစ်ပြီး ယခု သင်ခန်းစာမှာတော့ file တစ်ခု တည်ဆောက်ပြီး ထို file ထဲသို့ data ပါ တစ်ခါတည်း ထည့်လိုသော ကြောင့်သုံးခြင်း ဖြစ်ပါတယ် TestingWords သည် မိမိတို့ file ထဲသို့ ထည့်လိုသော data ဖြစ်ပြီး hello1.txt သည် txt file ဖြစ်ပါတယ် ထိုနေရာတွင် မိမိတို့

program file(python ,c, java,ect..) ဖြစ်နိုင်ပါတယ်။ File ဆောက်ပြီးပါက ls နှင့်ပြန်ကြည့်လိုက်သော အခါတွင် hello1.txt ဆိုသည့် file တစ်ခု ရောက်နေသည်ကို မြင်ရပါမည်။ ထိုနည်းတူ နောက်ထပ် file တစ်ခုကို ထပ် add ပါမည်။ ထို့နောက်တွင်မူ git status ဆိုသည့် command ကို အသုံးပြုပြီး ပြောင်းလဲသွားသည့် အချက်လက်များကို ကြည့်ပါမည်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ ls
hello1.txt  hello2.txt

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        hello1.txt
        hello2.txt

nothing added to commit but untracked files present (use "git add" to track)

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$
```

အထက်တွင် ကြည့်မည်ဆိုလျှင် ယခု လုပ်ဆောင်ချက်များသည် master ဆိုသည့် branch အောက်တွင် တည်ရှိနေသည်ကို ဖော်ပြ နေပါသည် နောက်ပိုင်းတွင် မိမိတို့ တည်ဆောက်လိုသည့် branch များကို တည်ဆောက်ပြီး ထို branch အောက်တွင် file များကို add နိုင်ပါသည်။ ထို့ပြင် No cmmits yet ဆိုသည်မှာ ယခုအချိန်ထိ file များအား repository ထဲသည့် မထည့်ရသေးဘူး ဆိုတာကို ဖော်ပြချင်တာ ဖြစ်ပါတယ်။ ထို့နောက်တွင် untracked files ဆိုသည့် စာသားကို မြင်ရမည် ထိုစာသားသည် hello1.txt နှင့် hello2.txt ဆိုသည့် file များအား မှတ်တမ်းတင်ခြင်း သိမ်းဆည်းခြင်း မရှိသေးကြောင်းကို ဖော်ပြခြင်း ဖြစ်ပါတယ်။ File များကို track လုပ်လိုပါက add ဆိုသည့် git command ကိုသုံးပြီး add ပေးရန် လိုအပ်သည် ထိုသို့ အပ်ရာတွင် git add fileName ဟုရေးပြီး add လျှင်လည်း ရသလို git add *.txt ဆိုသည့် command ကို သုံးမည်ဆိုလျှင် .txt နှင့် ဆုံးသည့် files အားလုံးကို add မှာ ဖြစ်ပါတယ်။ အကယ်၍ git add . ဆိုသည့် command ကို သုံးမည် ဆိုလျှင် ယခုလက်ရှိ working directory မှာ ရှိသည့် files အားလုံးကို add ပေးလိုက်မှာ ဖြစ်ပါတယ်။ စာရေးသူ အနေဖြင့် git add *.txt ဆိုသည့် command ကိုသုံးပြီး txt file အားလုံးကို add လိုက်ပါတယ်။

Git status နှင့်ပြန်ကြည့်မည်ဆိုလျှင် file နှစ်ခုသည် စိမ်းသွားသည်ကို မြင်ရပါမည် ထိုအခြေနေသည် staging ထဲသို့ file နှစ်ခုကို add လိုက်ခြင်းသည် ဖြစ်သည် repository ထဲသို့ add ခြင်း မဟုတ်သေးပါ ထို့ကြောင့် No commits yet ဆိုသည့် စာသားကို မြင်နေရဦးမှာ ဖြစ်ပါတယ်။ နောက်တစ်ဆင့်အ နေဖြင့် hello1.txt ဆိုသည့် file အား data များ ထပ်ပြီး ထည့်ကြည့်ပါမည် ထို့နောက် git status နှင့် ပြန်ကြည့်ပါမည်။



အထက်ပါ ပုံတွင် modified: hello1.txt ဆိုသည့် စာသားကို တွေ့ရပါမည် အဘယ်ကြောင့်ဆိုသော် echo Appending >> hello1.txt ဟု ရေးလိုက်သောကြောင့် hello1.txt ဆိုသည့် file ထဲသို့ Appending ဆိုသည့် စာသားကို ထပ်ထည့်လိုက်ခြင်းဖြစ်ပါတယ် ထိုသို့ append လုပ်ရာတွင် >> ကို သုံးပါသည်။

Commit

နောက်တစ်ဆင့် အနေဖြင့် git commit ဆိုသည့် command ကိုသုံးပြီး repository ထဲသို့ စတင် store လုပ်ပါမည်။ Git commit ဆိုသည့် command ကို သုံးရာတွင် အခြေခံ အားဖြင့် နည်းလမ်း နှစ်ခုကို သုံးနိုင်ပါသည် git commit -m “some messages” စသည်ဖြင့် ရေးနိုင်သလို git commit ဟုလည်း ရေးနိုင်သည်။ -m သည် message ကို ကိုယ်စားပြုခြင်း

ဖြစ်ပြီး git commit ကိုသာ အသုံးပြုမည်ဆိုလျှင် မိမိတို့ default အနေဖြင့် သတ်မှတ်ပေးထားသည့် editor ဖြင့် COMMIT_EDITMSG ဆိုသည့် file တစ်ခု ပွင့်လာပါမည် ထို file ထဲတွင် မိမိတို့ ရေးလိုသည့် မှတ်ချက် များကို ရေးပေးနိုင်သည်။ အောက်ပါ ပုံတွင် git commit ကိုသာ သုံးထားပါသည်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   hello1.txt
        new file:   hello2.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   hello1.txt


winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git commit
hint: Waiting for your editor to close the file...
```

အောက်ပါ ပုံတွင် ကြည့်လျှင် စာရေးသူ ထည့်ထားသော မှတ်ချက်များကို line 1 and 2 တွင် တွေ့ရမည် ဖြစ်ပြီး line 11 and 12 တွင်လည်း committed လုပ်ပြီးသည့် file များကို ဖော်ပြထားပါသည်။

```
C: > Users > winht > gitbook > .git > COMMIT_EDITMSG
1   Initial Commit
2      This is first commit for our project.
3   # Please enter the commit message for your changes. L
4   # with '#' will be ignored, and an empty message abor
5   #
6   # On branch master
7   #
8   # Initial commit
9   #
10  # Changes to be committed:
11  #   new file:   hello1.txt
12  #   new file:   hello2.txt
13  #
14  # Changes not staged for commit:
15  #   modified:   hello1.txt
16  #
17
```

အောက်ဆုံး line 14 and 15 တွင် အရေးကြီးသည့် message တစ်ကြောင်းကို ဖော်ပြထားပါသည် ဆိုလိုသည်မှာ hello1.txt ဆိုသည့် file ထဲသို့ data များ ထပ်ထည့်ပြီး modified လုပ်ထားသော်လည်း ထို modified လုပ်ထားသည့် data ချက်လက်များသည် commit ထဲတွင် မပါဝင်သေးပါ အဘယ်ကြောင့်ဆိုသော် hello1.txt ဆိုသည့် file အား modified လုပ်ပြီးသော်လည်း staging area ထဲသို့ မထည့်ရသေးသောကြောင့် ဖြစ်ပါတယ်။

ထို့နောက် editor အား ပြန်ပိတ်လိုက်လျှင် အောက်ပါ အတိုင်း commit လုပ်လိုက်သော အချက်လက်များ အား အသေးစိတ် တွေ့ရပါမည်။ File နှစ်ခု changes ဖြစ်သွားပုံနှင့် repository ထဲသို့ နှစ်ခု သွင်းလိုက်ကြောင်းကို ဖော်ပြထားပါသည်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git commit
[master (root-commit) 90e3fa3] Initial Commit    This is first commit for our
oject.
 2 files changed, 2 insertions(+)
 create mode 100644 hello1.txt
 create mode 100644 hello2.txt

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$
```

hello1.txt file အား add ပါမည် သို့မှသာ staging area ထဲသို့ ရောက်မည် ဖြစ်ပြီး commit လုပ်သော အခါတွင် ပါဝင်သွားမည် ဖြစ်ပါသည်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git add hello1.txt
warning: LF will be replaced by CRLF in hello1.txt.
The file will have its original line endings in your working directory

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   hello1.txt
```

Removing Files

Git မှာလည်း rm ဆိုတဲ့ command ကို သုံးပြီး မိမိတို့ မလိုအပ်တော့တဲ့ files တွေကို ဖျက်ထုတ်လို့ရပါတယ်။ ပထမဆုံး အနေဖြင့် git ls-files ဆိုသည့် command ကိုသုံးပြီး files များကို ကြည့်ပါမည်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git ls-files
hello1.txt
hello2.txt
```

ထို့နောက် hello2.txt file ထဲသို့ echo command ကိုသုံးပြီး data ထပ်ထည့်ပါမည်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ echo AddingData >> hello2.txt

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   hello1.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working director
y)
        modified:   hello2.txt
```

File ထဲသို့ data ထပ်ထည့်ပြီး hello1.txt သည် committed ဖြစ်ပြီး hello2.txt သည် staging area တွင်သာ ရှိနေပြီး commit မလုပ်ထားကြောင်း modified လုပ်ထားကြောင်းကို ပြနေပါမည်။ ထို့နောက် အောက်ပါ အတိုင်း git commit -am ဆိုသည့် command ကို သုံးပြီး commit လုပ်ပါမည် -a သည် all ကို ဆိုလိုခြင်း ဖြစ်ပြီး m သည် message ကို ဆိုလိုခြင်း ဖြစ်သည်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git commit -am "Adding more data to hello2.txt"
warning: LF will be replaced by CRLF in hello2.txt.
The file will have its original line endings in your working director
y
[master f21aeff] Adding more data to hello2.txt
 2 files changed, 2 insertions(+)
```

ထို့နောက် hello1.txt ဆိုသည့် file အား ဖျက်ပါမည်။ ထိုသို့ ဖျက်ရန် rm hello1.txt ဆိုသည့် command ကိုသုံးပါမည်။ ထို့နောက် git status ဖြင့် ပြန်ကြည့်ပါက အောက်ပါ အတိုင်း တွေ့ရပါမည်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ rm hello1.txt

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working director
y)
        deleted:    hello1.txt

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

ယခု နေရာတွင် အရေးကြီးသည့် အချက်မှာ hello1.txt ဆိုသည့် file ကို ဖျက်ထုတ်လိုက်သော်လည်း မူရင်း committed လုပ်ထားသည့် နေရာတွင် hello1.txt ဆိုသည့် file သည် ပျက်သွားခြင်း မရှိသေးပါ သက်သေ အနေဖြင့် git ls-files ဆိုသည့် command ကို ရိုက်ကြည့်ပါက files နှစ်ခုလုံး ရှိနေသေးသည်ကို တွေ့ရပါမည်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git ls-files
hello1.txt
hello2.txt
```

ထို file ပျက်သွားဖို့ အတွက် git add hello1.txt ဆိုသည့် command ကို ထပ်ရေးဖို့ လိုအပ်ပါသေးသည်။ ထို့နောက် git status ကို ကြည့်ပါက hello1.txt ကို ဖျက်လိုက်သည် ဆိုသည့် အစိမ်းရောင်ဖြင့် ရေးထားသော စာသားကို တွေ့ရပါမည်။ git ls-files ဆိုသည့် command ဖြင့် ကြည့်ပါကလည်း hello2.txt ဆိုသည့် file တစ်ခုတည်းကိုသာ တွေ့ရပါတော့မည်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git add hello1.txt

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    hello1.txt


winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git ls-files
hello2.txt

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$
```

နောက်ဆုံး အနေဖြင့် repository ထဲကနေပါ ဖျက်ရန် git commit ပေးရန် လိုအပ်ပါသည်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git commit -m "Removed hello1.txt"
[master 4bcecb6] Removed hello1.txt
 1 file changed, 2 deletions(-)
 delete mode 100644 hello1.txt

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$
```

အထက်တွင် ဖော်ပြထားသည်များကို ပြန်ကြည့်ပါက delete လုပ်ရန် ပထမဦးစွာ rm command ကို သုံးရသည် ထို့နောက် add command ကို သုံးရပါသည်။ အကယ်၍ files များကို files system ထဲကနေ သာမက git repository ထဲကနေပါ တခါတည်း ဖျက်လိုပါက အောက်ပါ အတိုင်း rm command ကို git ဖြင့် တွဲပြီး အသုံးပြုနိုင်သည်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ echo for delete > delete1.txt

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ echo for delete > delete2.txt

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        delete1.txt
        delete2.txt

nothing added to commit but untracked files present (use "git add" to track)
```

ပထမဦးစွာ delete1.txt နှင့် delete2.txt ဆိုသည့် file နှစ်ခုကို တည်ဆောက်လိုက်ပါသည် ထို့နောက် file နှစ်ခုအား git add command ကို သုံးပြီး staging area ထဲသို့ add လိုက်သည် ပြီးလျှင် git commit ကိုသုံးပြီး repository ထဲသို့ ထည့်လိုက်ပါသည်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git add delete1.txt delete2.txt
warning: LF will be replaced by CRLF in delete1.txt.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in delete2.txt.
The file will have its original line endings in your working directory

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   delete1.txt
        new file:   delete2.txt


winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git commit -m "Delete Testing"
[master 9215ec5] Delete Testing
 2 files changed, 2 insertions(+)
 create mode 100644 delete1.txt
 create mode 100644 delete2.txt
```

အထက်ပါ အတိုင်းပြီးပါက ls ကိုသုံးပြီး file များကို ပြန်ကြည့်ပါမည် ထို့နောက် rm ကို သုံးပြီး delete လုပ်ပါမည်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git ls-files
delete1.txt
delete2.txt
hello2.txt

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git rm delete1.txt delete2.txt
rm 'delete1.txt'
rm 'delete2.txt'
```

ယခင် remove လုပ်သော သင်ခန်းစာတွင် rm command ကိုသာ တိုက်ရိုက် သုံးခဲ့ခြင်း ဖြစ်ပြီး rm ရှေ့တွင် git မပါဝင်ပါ ယခု အခါတွင်မူ git ပါဝင်ပါသည်။ ထို့နောက် git status and ls ဖြင့် ပြန်ကြည့်ပါက အောက်ပါအတိုင်း file system and repository ထဲကနေပါ အားလုံး ဖျက်ပြီးသည်ကို တွေ့ရပါမည်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    delete1.txt
        deleted:    delete2.txt


winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git ls-files
hello2.txt
```

Removing Same File Types

ယခု သင်ခန်းစာမှာတော့တူညီနေတဲ့ files တွေ အားလုံးကို command တစ်ခုတည်းနဲ့ ဖျက်မျာ ဖြစ်ပါတယ်။ ဥပမာ စာရေးသူ အနေဖြင့် git repo ထဲတွင် test1.js , test2.js , test3.js စသည့် file အားလုံးကို တစ်ခါတည်း ဖျက်လိုသည် ဆိုပါစို့ git rm*js ဟု ရေးပြီး အားလုံးကို ဖျက်နိုင်ပါသည်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git rm *js
rm 'test1.js'
rm 'test2.js'
rm 'test3.js'

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    test1.js
        deleted:    test2.js
        deleted:    test3.js


winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git ls-files
hello2.txt
```

Files များကို add ရာတွင် sub folder အောက်မှ files များအားလုံးကိုပါ တစ်ပါတည်း add ချင်ပါက git add -A ဆိုသည့် command ကိုလည်း အသုံးပြုနိုင်ပါသေးတယ်။ စာရေးသူ အနေဖြင့် အောက်တွင် backdoor ဆိုသည့် folder တစ်ခု တည်ဆောက်လိုက်ပါသည် ထို့နောက် ထို backdoor folder ထဲတွင် test1.js , test2.js and test3.js စသည့် files များကို ဖန်တီးလိုက်ပါသည်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ mkdir backdoor

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ cd backdoor
```

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook/backdoor (master)
$ ls
test1.js  test2.js  test3.js

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook/backdoor (master)
$ cd ..

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ ls
backdoor/  hello2.txt

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git add -A
```

အထက်ပါ ပုံအတိုင်း backdoor folder ထဲမှ cd .. command ဖြင့် ထွက်ပြီး gitbook folder ထဲသို့ ပြန်သွားပါ ထို့နောက် git add -A ဟုရေးလိုက်ပါက backdoor folder ထဲမှာပါ ရှိသည့် files အားလုံးကို staging area ထဲသို့ add လိုက်သည်ကို တွေ့ရပါမည်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   backdoor/test1.js
        new file:   backdoor/test2.js
        new file:   backdoor/test3.js
        deleted:    test1.js
        deleted:    test2.js
        deleted:    test3.js
```

နောက်တစ်ဆင့် အနေဖြင့် staging area ထဲတွင် ရှိသည့် files အားလုံးကို commit ပြုလုပ်ပါမည်။ ထို့နောက် commit ပြုလုပ်ထားသည့် backdoor folder ထဲမှ files များကို ဖျက်ရန် files များကြည့်လိုပါက git rm backdoor/* --dry-run ဆိုသည့် command ကို အသုံးပြု နိုင်ပါသည်။

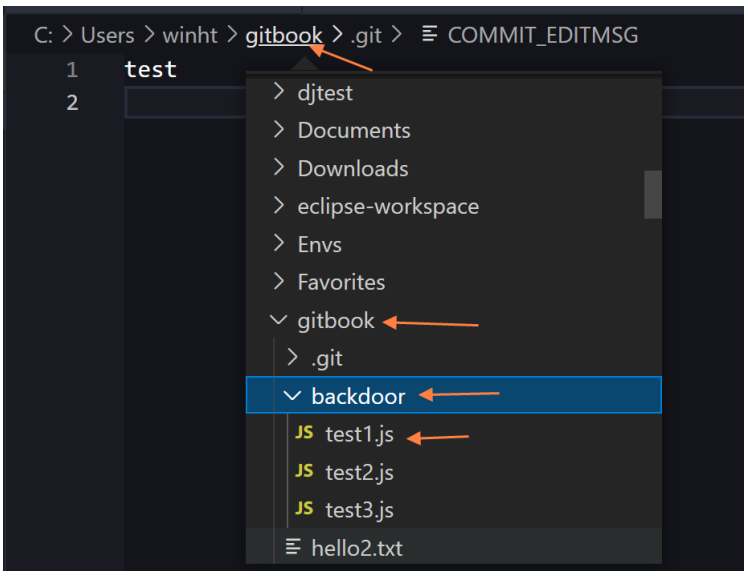
```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git rm backdoor/* --dry-run
rm 'backdoor/test1.js'
rm 'backdoor/test2.js'
rm 'backdoor/test3.js'
```

Only Remove From Repository

အကယ်၍ file တစ်ခုကို repository ထဲကနေသာ ဖျက်လိုပြီး file system ထဲကနေ မဖျက်လိုပါက --cached ဆိုသည့် command ကို အသုံးပြုနိုင်ပါသည်။ စာရေးသူတို့ အနေဖြင့် backdoor ဆိုသည့် folder ထဲမှ test1.js ဆိုသည့် file ကို ဖျက်လိုပါသည် သို့သော် repo ထဲကသာ ဖျက်ပြီး file system ထဲက မဖျက်လိုပါကအောက်ပါ အတိုင်း ရေးနိုင်သည်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git rm backdoor/test1.js --cached
rm 'backdoor/test1.js'

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git ls-files
backdoor/test2.js
backdoor/test3.js
hello2.txt
```

အထက်ပါ အတိုင်းပြီးပါက git ls-files ဖြင့် စစ်ကြည့်ပါက backdoor folder ထဲတွင် test1.js ဆိုသည့် file မရှိတော့သည်ကို မြင်ရပါမည်။ File System ထဲတွင် ရှိသည် မရှိသည်ကို စစ်ဆေးလိုပါက code ဆိုသည့် command ကို ရေးပြီး visual studio code တွင် အောက်ပါ အတိုင်း ကြည့်နိုင်ပါသည်။ gitbook folder ထဲမှ backdoor folder ထဲတွင် ကြည့်ပါ test1.js ဆိုသည့် file ရှိနေ သေးသည်ကို တွေ့ရပါမည်။



Move or Rename

(mv) သည် move ရဲ့ အတိုကောက် ဖြစ်ပြီး Unix command တစ်ခု ဖြစ်ပါတယ် mv ကို file တစ်ခု သို့မဟုတ် တစ်ခုထပ်ပိုပြီး နေရာတစ်ခုကနေ အခြားနေရာ တစ်ခုဆီသို့ ရွှေ့လို့သော အခါမျိုးမှာ သုံးပါတယ်။ အကယ်၍ file system တစ်ခုတည်း အောက်မှာ ရှိနေသည် ဆိုလျှင် file ကို rename လုပ်ရန် အသုံးပြုပြီး file အဟောင်းကိုတော့ဖျက်ပစ်လိုက်ပါတယ်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ ls
backdoor/  hello2.txt

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ mv hello2.txt backdoor.py

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    backdoor/test1.js

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    hello2.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        backdoor.py
        backdoor/test1.js
```

အထက်ပါ ပုံတွင် hello2.txt ကို mv command သုံးပြီး backdoor.py အဖြစ်သို့ ပြောင်းလိုက်ခြင်း ဖြစ်ပါတယ်။ သို့သော် file အဟောင်းကို ဖျက်ပစ်ပြီး အသစ် ဖန်တီးဖို့အတွက် add ပေးရန် လိုအပ်ပါသေးသည်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git add hello2.txt

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git add backdoor.py
warning: LF will be replaced by CRLF in backdoor.py.
The file will have its original line endings in your working directory

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        renamed:    hello2.txt -> backdoor.py
        deleted:    backdoor/test1.js
```

အထက်ပါ အတိုင်း add ပြီးပါက အောက်ဆုံးတွင် hello2.txt ကို backdoor.py သို့ ပြောင်း ပြီးဆိုသည့် စာသား ကိုတွေ့ရပါမည်။ အထက်ပါ နည်းလမ်းအရ ဆိုလျှင် အဆင့် နှစ်ဆင့်ကို ပြုလုပ်ရပါသည် ပထမ အဆင့်သည် mv command ကိုသုံးရပြီး ဒုတိယ အဆင့်သည် add command ကို သုံးရပါသည် အကယ်၍ အဆင့် နှစ်ခု မပြုလုပ်လို ပဲ တစ်ခုတည်း နှင့် ပြီးမြောက်လိုပါက အောက်ပါ အတိုင်း mv command ကို git ဖြင့် တွဲသုံး နိုင်ပါသည်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ ls
backdoor/  backdoor.py

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git mv backdoor.py malware.py
```

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        renamed:    hello2.txt -> malware.py
```

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git commit -m "changing file name and extension"
[master 559e3f2] changing file name and extension
 1 file changed, 0 insertions(+), 0 deletions(-)
 rename hello2.txt => malware.py (100%)

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$
```

နောက်ဆုံး အနေဖြင့် commit လိုက်ပါက 1 file changed သွားကြောင်းနှင့် name change သွားကြောင်းပြသည့် message ကိုတွေ့ရပါမည်။

.gitignore

Git သည် file တစ်ခုကို နည်းလမ်း သုံးခု နဲ့ မြင်ပါသည် 1.tracked , 2.untracked , 3.ignored တို့ဖြစ်ပါတယ်။ Tracked သည် file တစ်ခုအား staging area သို့ committed လုပ်ထားခြင်းဖြစ်ပြီး untracked ကတော့ tracked ရဲ့ ပြောင်းပြန်ဖြစ်ပါတယ်။ ignored

ကတော့ file တစ်ခု သို့မဟုတ် folder တစ်ခုလုံးအား commit လုပ်ခြင်း tracking လုပ်ခြင်းမှ ဖယ်ထုတ်ထားရန် ဖြစ်သည်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~
$ cd gitbook

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ touch .gitignore

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore

nothing added to commit but untracked files present (use "git ad
d" to track)

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ touch trojan.py
```

အထက်ပါ ပုံအတိုင်း စာရေးသူတို့ အနေဖြင့် gitbook ဆိုသည့် folder ထဲသို့ ဝင်လိုက်ပါသည် ထို့နောက် touch command ကိုသုံးပြီး .gitignore ဆိုသည့် file တစ်ခုကို တည်ဆောက် လိုက်ပါသည်။ ထို့နောက် git status ဖြင့် ပြန်ကြည့်ရာ .gitignore file သည် untracked လုပ်ထားကြောင်း တွေ့ရပါမည်။ ထို့နောက် touch trojan.py ဆိုသည့် Python file တစ်ခုကို ထပ်တည်ဆောက်ပါသည် git status ဖြင့် ပြန်ကြည့်သော အခါ .gitignore နှင့် trojan.py ဆိုသည့် file နှစ်ခုလုံးကို untrack အနေဖြင့် မြင်ရပါမည်။ စာရေးသူတို့ အနေဖြင့် tracking or commit လုပ်ရာတွင် ထို python file အား မပါဝင် စေလိုပါ ထို့ကြောင့် code .gitignore ဆိုသည့် command ကိုသုံးကာ visual studio code တွင် .gitignore file ကို ဖွင့်လိုက်ပါသည်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        trojan.py

nothing added to commit but untracked files present (use "git ad
d" to track)

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ code .gitignore
```

ထို့နောက် အောက်ပါ ပုံအတိုင်း မိမိတို့ tracking or commit ထဲတွင် မပါဝင်စေချင်သည့် file name ကို ရေးပေးနိုင်သည် ထို့ပြင် .py ဖြင့် ဆုံးသည့် files များစွာကို မပါဝင် စေလိုပါက *.py စသည်ဖြင့် ရေးနိုင်သည်။

ထို့နောက် git status ဖြင့် ပြန်ကြည့်သော အခါ အောက်ပါ အတိုင်း .gitignore file တစ်ခုတည်းသာ ရှိတော့သည်ကို တွေ့ရပါမည်။ ထို့နောက် file အားလုံးအား -A command ကိုသုံးပြီး add လုပ်ကာ commit လုပ်နိုင်ပါပြီ။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git add -A

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   .gitignore


winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git commit -m "adding ignore file"
[master f5f09aa] adding ignore file
 1 file changed, 1 insertion(+)
 create mode 100644 .gitignore
```

နောက် တစ်ဆင့် အနေဖြင့် code ကို ရေးကာ vs code ကို ဖွင့်ပြီး trojan.py file ထဲတွင် မိမိတို့ ရေးလိုသည့် စာရေးပါ ထို့နောက် save ပြီး git status ဖြင့် ပြန်ကြည့်ပါက မည့်သည့် စာသားမျှ မဖော်ပြပဲ untrack လုပ်ထားကြောင်းကို တွေ့ရပါမည်။


```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ code

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   .gitignore
```

ထို့ပြင် git ls-files ဖြင့် ကြည့်ပါက အောက်ပါ အတိုင်း files များကို မြင်ရမည် ဖြစ်ပြီး trojan.py ဆိုသည့် file မပါဝင်ပါ သို့သော် ls ဖြင့် ကြည့်ပါက trojan.py ပါ ရှိနေသည်ကို မြင်ရပါမည်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git ls-files
.gitignore
backdoor/test1.js
backdoor/test2.js
backdoor/test3.js
malware.py

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ ls
backdoor/  malware.py  trojan.py

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
```

Ignore Repository File

ယခု သင်ခန်းစာမှာတော့ repository ထဲသို့ commit လုပ်ထားပြီးသား file ကို ignore ပြုလုပ်တဲ့ အကြောင်း ဖော်ပြ သွားမှာ ဖြစ်ပါတယ်။ ပထမဆုံး အနေဖြင့် folder တစ်ခု ဖန်တီးပါ ထို folder ထဲတွင် file တစ်ခု တည်ဆောက်ပြီး commit လုပ်ပါ။

```
winht@DESKTOP-RMO2V53 MINGW64 ~
$ cd gitbook

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ mkdir compiled

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ echo compiler > compiled/com.bin

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   .gitignore
        modified:   logbook/log.bat

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        compiled/
```

အထက်ပါ ပုံအတိုင်း compiled ဆိုသည့် folder တစ်ခု ထပ်ပြီး တည်ဆောက်လိုက်ပါသည် ထို့ပြင် compiled folder ထဲ၌ com.bin ဆိုသည့် file တစ်ခု ထပ်တည်ဆောက်ပြီး compiler ဆိုသည့် စာသား ရေးထည့်ထားလိုက်ပါသည်။ git status ဖြင့် ကြည့်မည် ဆိုလျှင် compiled ဆိုသည့် folder တစ်ခုသည် untracked ဖြစ်နေကြောင်းကို တွေ့ရပါမည်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git add .
warning: LF will be replaced by CRLF in compiled/com.bin.
The file will have its original line endings in your working directory

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   .gitignore
        new file:   compiled/com.bin
        modified:   logbook/log.bat


winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git commit -m "Adding compiled folder"
[master 570ebc1] Adding compiled folder
 3 files changed, 4 insertions(+), 2 deletions(-)
 create mode 100644 compiled/com.bin
```

နောက်တစ်ဆင့် အနေဖြင့် compiled folder အား add လုပ်ကာ အထက်ပါ ပုံအတိုင်း Adding compiled folder ဟူသော် message ဖြင့် commit လုပ်ထားပါသည်။ နောက်တစ်ဆင့် အနေဖြင့် compiled folder အား gitignore ထဲသို့ ထည့်ပေးပါမည် သို့မှသာ စာရေးသူတို့ရဲ့ compiled folder ကို git က tracking မလုပ်မှာ ဖြစ်ပါတယ် သို့သော် compiled folder သည် commit လုပ်ထာပြီးသား ဖြစ်သည့် အတွက် repository ထဲမှ ပြန်ထုတ်ရန် လိုအပ်ပါသည် ။ Repository ထဲမှ ပြန်ထုတ်လိုပြီး file system ထဲမှ ပျက်မသွားစေလိုသော အခါမျိုးတွင် rm --cached ဆိုသည့် command ကို အသုံးပြုရပါမည် ထို့ပြင် recursive removal ဖြစ်လိုသော အခါတွင် -r ကိုပါ နောက်တွင် ထပ်ထည့်ပေးရပါမည်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git rm --cached -r compiled
rm 'compiled/com.bin'

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    compiled/com.bin

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        compiled/
```

အထက်ပါ ပုံတွင် compiled folder ကို deleted လုပ်ပြီး untrack လုပ်ထားကြောင်းကို တွေ့ရပါမည်။ ထို့နောက် အောက်ပါ ပုံအတိုင်း code .gitignore command ကိုရေးပြီး visual studio code ပွင့်လာလျှင် compiled folder ကို ထည့်ပေးပါ။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ code .gitignore
```

```
C: > Users > winht > gitbook > .gitignore
1   trojan.py
2   compiled/
```

အထက်ပါ အဆင့်များပြီးသွားလျှင် git status ဖြင့် ပြန်ကြည့်ပါက gitignore file ကို modified လုပ်ထားကြောင်း တွေ့ရပါမည် ။ ထို့နောက် git add ပြုလုပ်ပြီး လက်ရှိ အခြေနေများကို commit ပြန်လုပ်ပေးပါ။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    compiled/com.bin

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   .gitignore


winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git add .

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git commit -m "moving compiled folder to ignore"
[master 8c12344] moving compiled folder to ignore
 2 files changed, 2 insertions(+), 2 deletions(-)
 delete mode 100644 compiled/com.bin
```

ယခု အဆင့်များ ပြီးသွားလျှင် git status ဖြင့် ပြန်ကြည့်ပြီး compiled folder ထဲမှ com.bin file ထို့သို့ data အနည်းငယ် ထပ်ထည့်ပြီး git status ဖြင့် ပြန်ကြည့်ပါက tracking မလုပ်တော့ကြောင်းကို တွေ့ရပါမည်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ echo compileData > compiled/com.bin

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Git Log

Git log သည် မိမိတို့ commits လုပ်လိုက်သည့် repository ထဲသို့ ရောက်သွားသော history များကို ပြန်ကြည့်နိုင်ဖို့အတွက် command တစ်ခု ဖြစ်ပါတယ်။ git ထဲတွင် git log command အသုံးပြုကြည့်မည်ဆိုလျှင် အချက် ၄ ချက် ပြန်ရလာသည်ကို တွေ့ရပါမည်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git log
commit 8c12344034469ff2a9b246f1b14a94572957491b (HEAD -> master)
Author: Win Htut <winhtutonline@gmial.com>
Date:   Mon Apr 26 20:51:03 2021 +0630

    moving compiled folder to ignore
```

Commits history များကို ပြန်ပြတဲ့ အခါမှာ reverse chronological order နဲ့ ပြန်ပြပေးပါတယ် နောက်ဆုံးမှာ လုပ်ထားတဲ့ commit ကို ပထမဦးဆုံး အနေနဲ့ ပြန်ပြပေးတာပါ။

1. Commit Hash: commit log ရဲ့ ပထမဆုံး စာကြောင်းသည် commit hash ဖြစ်ပါတယ် git ကနေပြီး commit တစ်ခုခြင်းစီ တိုင်း အတွက် ထုတ်ပေးထားတဲ့ hash values တွေ ဖြစ်ပါတယ်။
2. Author : ယခုစာကြောင်းကတော့ repository ထဲမှာ changes ဖြစ်သွားဖို့ commit လုပ်ထားတဲ့ author name ကိုဖော်ပြ ပေးထားတာ ဖြစ်ပါတယ်။
3. Data : ယခု အပိုင်းမှာတော့ date and time အပြင် commit လုပ်သော အချိန်ရဲ့ နေ့ရက် နှင့်အတူ time zone ကိုပါ ဖော်ပြ ပေးထားပါတယ်။ ဘာကြောင့် time zone ကိုပါ

အတိအကျ ဖော်ပြ ပေးတာလည်း ဆိုတော့ မတူညီတဲ့ အရပ်ဒေသက geographical locations လူတွေဟာ project တစ်ခုတည်းမှာ အတူတကွ အလုပ်လုပ်ကြတဲ့ အတွက် ဖြစ်ပါတယ် ဥပမာ +0630 သည် UTC(Coordinated Universal Time) 6:30 hours ကို ဆိုလိုခြင်း ဖြစ်ပါတယ်။ +6 :30 သည် မြန်မာနိုင်ငံရဲ့ UTC Offset time ဖြစ်ပါတယ်။

4. နောက်ဆုံး အပိုင်းကတော့ developer က commit လုပ်လိုက်တဲ့ အခါမှာ တစ်ပါတည်း သတ်မှတ် ပေးလိုက်တဲ့ message ဖြစ်ပါတယ်။

Developer တွေဟာ git log ကိုအသုံးနည်းပါတယ် ဘာကြောင့်လည်းဆိုတော့ git log သည် history များစွာ ထုတ်ပေးတဲ့ အတွက် အချက်အလက် အများကြီးကို မလိုအပ်သေးပဲ မကြည့်လိုကြပါဘူး ထို့အတွက်လည်း git ကနေပြီး အသုံးဝင် မယ့် command တွေကို စီစဉ် ပေးထားပါသေးတယ်။ Developer များ အနေဖြင့် commit တစ်ခုစီ တိုင်းအတွက် message တစ်ကြောင်းတည်းကိုသာ ကြည့်လိုပါက git log --oneline ဆိုသည့် command ကို အသုံးပြုနိုင်ပါတယ်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git log --oneline
8c12344 (HEAD -> master) moving compiled folder to ignore
570ebc1 Adding compiled folder
fd3a1c4 adding logbook folder
7220e64 Adding Logs Folder
f5f09aa adding ignore file
559e3f2 changing file name and extension
ef5a61d adding all files form sub
edd9278 testing file deletion
9215ec5 Delete Testing
4bcecb6 Removed hello1.txt
f21aeff Adding more data to hello2.txt
90e3fa3 Initial Commit    This is first commit for our project.
```

Git log --oneline

Git log မှာတော့ commit နဲ့ပတ်သက်သည့် အသေးစိတ် အချက်လက် အားလုံးကို ပြပေးသော်လည်း git log --oneline မှာတော့ အထက်ပါ ပုံတွင် ပြထားသည့် အတိုင်း commit id နှင့် commit message တို့ကိုသာ ပြပေးပါတယ်။ git user တစ်ယောက်ကနေပြီး commit လုပ်လိုက်တိုင်းမှာ commit id ရှိပါတယ်။ ထို commit id သည် commit hash ရဲ့ compressed version ဖြစ်ပါတယ်။

View Commit History by Commit ID

Git user များ အနေဖြင့် commit id ကို သုံးပြီး git log ပြန်ကြည့် နိုင်ပါသေးတယ် ဥပမာ commit id သည် 8c12344 ဆိုပါစို့ git log 8c12344 ဟု ရေးပြီး ကြည့်မည်ဆိုလျှင် ထို id နှင့် ပတ်သက်သည့် အသေးစိတ် အချက်လက် အားလုံးကို ပြန်ကြည့် နိုင်မှာ ဖြစ်ပါတယ်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git log 8c12344
commit 8c12344034469ff2a9b246f1b14a94572957491b (HEAD -> master)
Author: Win Htut <winhtutonline@gmial.com>
Date:   Mon Apr 26 20:51:03 2021 +0630

    moving compiled folder to ignore

commit 570ebc1c870966c2a8f212933d817bf67f092264
Author: Win Htut <winhtutonline@gmial.com>
Date:   Mon Apr 26 20:49:01 2021 +0630

    Adding compiled folder

commit fd3a1c418070d79cd6d49e5e98c55b435f824b38
Author: Win Htut <winhtutonline@gmial.com>
```

ယခု အတိုင်း ကြည့်လိုက်မည် ဆိုလျှင် commit id 8c12344 ရဲ့ history အားလုံးကို အသေးစိတ် ပြန်မြင်ရမှာ ဖြစ်ပါတယ်။

History of a File

Git user များ အနေဖြင့် file တစ်ခုတည်းရဲ့ history ကို အသေးစိတ် ပြန်လည် ကြည့်နိုင်ဖို့အ တွက်လည်း git ကနေ စီစဉ်းပေးထားပါသေးတယ် အဲဒါကတော့ git log <filename> ဟုရေးပြီး ကြည့်နိုင်ပါတယ်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook/compiled (master)
$ git log com.bin
commit 8c12344034469ff2a9b246f1b14a94572957491b (HEAD -> master)
Author: Win Htut <winhtutonline@gmial.com>
Date:   Mon Apr 26 20:51:03 2021 +0630

    moving compiled folder to ignore

commit 570ebc1c870966c2a8f212933d817bf67f092264
Author: Win Htut <winhtutonline@gmial.com>
Date:   Mon Apr 26 20:49:01 2021 +0630

    Adding compiled folder
```

အထက်ပါ ပုံမှာ ဆိုလျှင် compiled folder ထဲက com.bin ဆိုတဲ့ file တစ်ခုရဲ့ commit history ကို ကြည့်ပြ ပေးထားခြင်း ဖြစ်ပါတယ်။

Commits History for a range

Commits history များကို ပြန်ကြည့်တဲ့ အခါမှာ git user တွေရဲ့ လိုသလောက် range အတွင်းမှာသာ ပြန်ကြည့်နိုင်ဖို့အတွက် git က စီစဉ် ပေးထားပါသေးတယ် specific commit history လို့ခေါ်ပါတယ် commit id တစ်ခုနဲ့ တစ်ခုကြားမှာ ရှိတဲ့ commits history အားလုံးကိုပြန်ထုတ် ပြ ပေးတာပါ။ အထက်တွင် ဖော်ပြပြီး သင်ခန်းစာများတွင် commit history ၌ commit id အားလုံးကို ဖော်ပြ ပေးပါတယ် ထို commit id တစ်ခုနဲ့ တစ်ခုကြားက commit တွေကို ပြန်ကြည့်လိုသော အခါတွင် git log <since commid id>..<until commit id> ဟုရေးပြီး ပြန်ကြည့် နိုင်ပါတယ်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git log f5f09aa..8c12344
commit 8c12344034469ff2a9b246f1b14a94572957491b (HEAD -> master)
Author: Win Htut <winhtutonline@gmial.com>
Date:   Mon Apr 26 20:51:03 2021 +0630

    moving compiled folder to ignore

commit 570ebc1c870966c2a8f212933d817bf67f092264
Author: Win Htut <winhtutonline@gmial.com>
```

အထက်ပါ ပုံအတိုင်း git log ကို သုံးကြည့်ပါက f5f09aa id နှင့် 8c12344 id ကြားမှာ ရှိတဲ့ commits log တွေကို တွေ့ရမှာ ဖြစ်ပါတယ်။သတိ ပြုရန် အချက်မှာ since commit id နေရာတွင် ရေးသော commit id ရဲ့ အချက်လက်တွေတော့ ပါလာမှာ မဟုတ်ပါဘူး။ အောက်ပါ ပုံတွင် ကြည့်မည် ဆိုလျှင် f5f09aa ရဲ့ commit အချက်လက်တွေ ပါဝင်လာမည် မဟုတ်ပါ။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git log --oneline
8c12344 (HEAD -> master) moving compiled folder to ignore
570ebc1 Adding compiled folder
fd3a1c4 adding logbook folder
7220e64 Adding Logs Folder
f5f09aa adding ignore file
559e3f2 changing file name and extension
```

အကယ်၍ git user များ အနေဖြင့် git log range ကိုလည်း oneline ပုံစံဖြင့် ကြည့်လိုပါကလည်း git log <since commid id>..<until commit id> --oneline ဟုရေးပြီး ကြည့်နိုင်ပါသေးတယ်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git log f5f09aa..8c12344 --oneline
8c12344 (HEAD -> master) moving compiled folder to ignore
570ebc1 Adding compiled folder
fd3a1c4 adding logbook folder
7220e64 Adding Logs Folder
```

Git log နှင့် ပတ်သက်ပြီး အသုံးများသည့် command တစ်ခုမှာ နောက်ဆုံးတစ်ခုမှာ git log -n 3 --oneline ဆိုသည့် command ဖြစ်ပါသည် ထို command သည် repository ထဲသို့ နောက်ဆုံး commit လုပ်ခဲ့သည့် အချက်လက်များကို ပြန်လည် ဖော်ပြ ပေးပါသည် အကယ်၍ git user အနေဖြင့် နောက်ဆုံး commit လုပ်ခဲ့သည့် အချက်လက် နှစ်ခုကိုသာ ကြည့်လိုပါကလည်း git log -n 2 --oneline ဖြစ်ပါသည်။ အောက်ပါ ပုံတွင် စာရေးသူ အနေဖြင့် နောက်ဆုံး commit လုပ်ခဲ့သည့် အချက်လက် လေးခုကို ဖော်ပြထားပါသည်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook (master)
$ git log -n 4 --oneline
8c12344 (HEAD -> master) moving compiled folder to ignore
570ebc1 Adding compiled folder
fd3a1c4 adding logbook folder
7220e64 Adding Logs Folder
```

Diff command

File တစ်ခုပေါ်မှ ပြောင်းလဲသွားတဲ့ မတူညီသောဖြစ်စဉ်တွေကို အသေးစိတ် သိနိုင်ဖို့အတွက် diff ဆိုတဲ့ git command ကို အသုံးပြုပါတယ်။ Git သည် version control system ဖြစ်သည်နှင့် အညီ changes ဖြစ်သွားတဲ့ အရာတွေကို tracking လုပ်နိုင်ဖို့က အရေးကြီးဆုံးဖြစ်ပါတယ်။ Diff command ကို အသုံးပြုရာတွင် input အနေဖြင့် files နှစ်ခုကို ရယူပြီး ထို files နှစ်ခုကြားမှာ ရှိတဲ့ မတူညီမှုတွေကို ပြန်ဖော်ပြ ပေးပါတယ်။ Files နှစ်ခုဟုဆိုရခြင်း ဥပမာ စာရေးသူ အနေဖြင့် hello.wt ဆိုသည့် file တစ်ခုကို create လုပ်လိုက်သည် ဆိုပါစို့ ထို့နောက် ထို file ထဲကို စာသား အချို့ ထည့်ပြီး commit လုပ်ထားလိုက်ပါသည်(File a) နောက်တစ်ဆင့် အနေဖြင့် ထို file ထဲသို့ စာသားများ ထပ်ထည့်ပြီး commit မလုပ်သေးပဲ staging area တွင်သာ ထားထားပါသည်(File b) ။ ယခုဆိုလျှင် file နှစ်ခု ဆိုတာ နားလည်သွားလိမ့်မယ်လို့ မျှော်လင့် ပါတယ် အသေးစိတ် ပိုမို သိရှိနိုင်ရန် အောက်တွင် diff ဆိုသည့် folder တစ်ခု တည်ဆောက်ပြီး hello.wt ဆိုသည့် file တစ်ခု တည်ဆောက်ကာ Hello ဆိုသည့် စာသားကို ထည့်ထားပါသည်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook/diff (master)
$ echo Hello > hello.wt
```

နောက်တစ်ဆင့် အနေဖြင့် ထို file အား add ပြီး commit လုပ်ထားပါသည်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook/diff (master)
$ git add .
warning: LF will be replaced by CRLF in diff/hello.wt.
The file will have its original line endings in your worki
ng directory

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook/diff (master)
$ git commit -m "Testing for diff"
[master 637a5f2] Testing for diff
```

ထို့နောက် hello.wt ထဲသို့ world ဆိုသည့် စာသားကို ထပ်ထည့်ပြီး git add . ထားပါသည်။

```
winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook/diff (master)
$ echo World > hello.wt

winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook/diff (master)
$ git add .
warning: LF will be replaced by CRLF in diff/hello.wt.
The file will have its original line endings in your worki
ng directory
```

ယခု ဆိုလျှင် file နှစ်ခု ရှိသွားပါပြီ ပထမ တစ်ခုသည် commit လုပ်ထားပြီးသား a file ဖြစ်ပြီး ဒုတိယ တစ်ခုသည် hello.wt ထဲသို့ စာသား ထပ်ထည့်ထားသည့် commit မလုပ်ရသေးပဲ staging area တွင်သာရှိသေးသော b file ဖြစ်ပါသည်။ git diff --staged ဟူသော စာသားကို ရေးပြီး different ကိုအောက်တွင်ကြည့်နိုင်ပါပြီ။


winht@DESKTOP-RMO2V53 MINGW64 ~/gitbook/diff (master)
$ git diff --staged
diff --git a/diff/hello.wt b/diff/hello.wt
index e965047..216e97c 100644
--- a/diff/hello.wt
+++ b/diff/hello.wt
@@ -1 +1 @@
-Hello
+World