

Лабораторная работа №2

Операционные системы

Ежова Алиса Михайловна

Содержание

1	Цель работы	4
2	Задание	5
3	Выполнение лабораторной работы	6
4	Выводы	12
5	Контрольные вопросы	13

Список иллюстраций

3.1	Настройка git	6
3.2	Настройка utf-8	6
3.3	ssh ключ	7
3.4	ssh ключ	7
3.5	Создание репозитория	7
3.6	Клонирование репозитория	8
3.7	Генерация ключа	8
3.8	Настройка ключа	8
3.9	Настройка ключа	9
3.10	Добавление gpg ключа на GitHub	9
3.11	Формат gpg ключа	9
3.12	Ключ на GitHub	10
3.13	Настройка коммитов	10
3.14	Настройка каталога	10
3.15	Отправка	10
3.16	Отправка	11

1 Цель работы

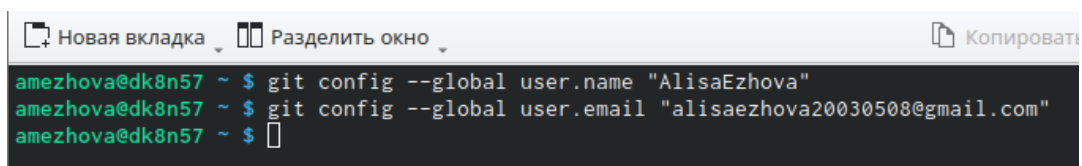
Изучить идеологию и применение средств контроля версий. Освоить умения по работе с git.

2 Задание

1. Создать базовую конфигурацию для работы с git.
2. Создать ключ SSH.
3. Создать ключ PGP.
4. Настроить подписи git.
5. Зарегистрироваться на Github.
6. Создать локальный каталог для выполнения заданий по предмету.

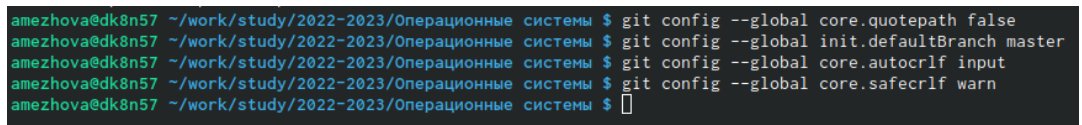
3 Выполнение лабораторной работы

1) Базовая настройка git:



```
amezhova@dk8n57 ~ $ git config --global user.name "AlisaEzhova"
amezhova@dk8n57 ~ $ git config --global user.email "alisaezhova20030508@gmail.com"
amezhova@dk8n57 ~ $
```

Рис. 3.1: Настройка git



```
amezhova@dk8n57 ~/work/study/2022-2023/Операционные системы $ git config --global core.quotePath false
amezhova@dk8n57 ~/work/study/2022-2023/Операционные системы $ git config --global init.defaultBranch master
amezhova@dk8n57 ~/work/study/2022-2023/Операционные системы $ git config --global core.autocrlf input
amezhova@dk8n57 ~/work/study/2022-2023/Операционные системы $ git config --global core.safecrlf warn
amezhova@dk8n57 ~/work/study/2022-2023/Операционные системы $
```

Рис. 3.2: Настройка utf-8

2) Создание ключей ssh:

```

amezhova@dk8n57 ~/work/study/2022-2023/Операционные системы $ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/afs/.dk.sci.pfu.edu.ru/home/a/m/amezhova/.ssh/id_rsa):
/afs/.dk.sci.pfu.edu.ru/home/a/m/amezhova/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /afs/.dk.sci.pfu.edu.ru/home/a/m/amezhova/.ssh/id_rsa
Your public key has been saved in /afs/.dk.sci.pfu.edu.ru/home/a/m/amezhova/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:THhjQI3A3S1wp8Pj0kH3yElU7kausJIB/ARhQKvhLoM amezhova@dk8n57
The key's randomart image is:
+---[RSA 4096]-----+
|oo+.. o.=..|
|+.o * X =|
|..= o @ X +|
|o. = * X =|
|.. * = S +|
|o + o o|
|E. o . .|
|.. .|
|_|
+----[SHA256]-----+

```

Рис. 3.3: ssh ключ

```

amezhova@dk8n57 ~/work/study/2022-2023/Операционные системы $ ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/afs/.dk.sci.pfu.edu.ru/home/a/m/amezhova/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /afs/.dk.sci.pfu.edu.ru/home/a/m/amezhova/.ssh/id_ed25519
Your public key has been saved in /afs/.dk.sci.pfu.edu.ru/home/a/m/amezhova/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:TcxEeJ0EnQt8eZqWPNAypGxFN8ZMWvmFbEYEHTCP0ik amezhova@dk8n57
The key's randomart image is:
+--[ED25519 256]--+
|      +=&&%+o      |
|      . oBX&*B .   |
|      + E*B@..    |
|      . =0 .      |
|      S...        |
|_|
+----[SHA256]-----+

```

Рис. 3.4: ssh ключ

3) Создание репозитория курса на основе шаблона:

```

amezhova@dk6n55 ~ $ mkdir -p ~/work/study/2022-2023/"Операционные системы"
amezhova@dk6n55 ~ $ cd ~/work/study/2022-2023/"Операционные системы"

```

Рис. 3.5: Создание репозитория

```
amezhova@dk6n55 ~/work/study/2022-2023/Операционные системы $ git clone --recursive https://github.com/AlisaEzhova/os-intro.git
Клонирование в «os-intro»...
remote: Enumerating objects: 27, done.
remote: Counting objects: 100% (27/27), done.
remote: Compressing objects: 100% (26/26), done.
remote: Total 27 (delta 1), reused 11 (delta 0), pack-reused 0
Получение объектов: 100% (27/27), 16.93 КиБ | 8.47 МиБ/с, готово.
Определение изменений: 100% (1/1), готово.
```

Рис. 3.6: Клонирование репозитория

4) Создание pgp ключа:

```
amezhova@dk8n57 ~/work/study/2022-2023/Операционные системы $ gpg --full-generate-key
```

Рис. 3.7: Генерация ключа

```
gpg (GnuPG) 2.2.40; Copyright (C) 2022 g10 Code GmbH
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Выберите тип ключа:
  (1) RSA и RSA (по умолчанию)
  (2) DSA и Elgamal
  (3) DSA (только для подписи)
  (4) RSA (только для подписи)
  (14) Имеющийся на карте ключ
Ваш выбор? 1
длина ключей RSA может быть от 1024 до 4096.
Какой размер ключа Вам необходим? (3072) 4096
Запрошенный размер ключа - 4096 бит
Выберите срок действия ключа.
  0 = не ограничен
  <n> = срок действия ключа - n дней
  <n>w = срок действия ключа - n недель
  <n>m = срок действия ключа - n месяцев
  <n>y = срок действия ключа - n лет
Срок действия ключа? (0) 0
Срок действия ключа не ограничен
Все верно? (y/N) y
```

Рис. 3.8: Настройка ключа


```

Ваше полное имя: Алиса
Адрес электронной почты: alisaekhova20030508@gmail.com
Примечание:
Используется таблица символов 'utf-8'.
Вы выбрали следующий идентификатор пользователя:
    "Алиса <alisaekhova20030508@gmail.com>"

Сменить (N)Имя, (C)Примечание, (E)Адрес; (O)Принять/(Q)Выход? o

```

Рис. 3.9: Настройка ключа

5) Добавление PGP ключа в GitHub:

```

amezhova@dk8n57 ~/work/study/2022-2023/Операционные системы $ gpg --list-secret-keys --keyid-format LONG
gpg: проверка таблицы доверия
gpg: marginals needed: 3 completes needed: 1 trust model: pgp
gpg: глубина: 0 достоверных: 1 подписанных: 0 доверие: 0-, 0q, 0n, 0m, 0f, 1u
/afs/.dk.sci.pfu.edu.ru/home/a/m/amezhova/.gnupg/pubring.kbx
-----
sec   rsa4096/997BB5B4ED930E83 2023-02-16 [SC]
      D6A07624F1BD40CB450DF529997BB5B4ED930E83
uid           [ абсолютно ] Алиса <alisaekhova20030508@gmail.com>
ssb   rsa4096/44A3DC84BCB0C934 2023-02-16 [E]

```

Рис. 3.10: Добавление gpg ключа на GitHub

```

amezhova@dk6n55 ~ $ gpg --armor --export 573F6EF27C221D81 | xclip -sel clip
amezhova@dk6n55 ~ $ git config --global user.signingkey 573F6EF27C221D81
amezhova@dk6n55 ~ $ git config --global commit.gpgsign true


```

Рис. 3.11: Формат gpg ключа

GPG keys

[New GPG key](#)

This is a list of GPG keys associated with your account. Remove any keys that you do not recognize.



love

Email address: `alisaekhova20030508@gmail.com`

Key ID: `997BB5B4ED930E83`

Subkeys: `44A3DC84BCB0C934`

Added on Feb 16, 2023

Delete

Рис. 3.12: Ключ на GitHub

6) Настройка автоатических подписей коммитов git:

```
amezhova@dk6n55 ~ $ git config --global user.signingkey 573F6EF27C221D81
amezhova@dk6n55 ~ $ git config --global commit.gpgsign true
amezhova@dk6n55 ~ $ git config --global gpg.program $(which gpg2)
```

Рис. 3.13: Настройка коммитов

7) Настройка каталога курса: Перейдем в каталог, удалим лишние файлы и создадим необходимые каталоги:

```
amezhova@dk6n55 ~/work/study/2022-2023/Операционные системы $ cd ~/work/study/2022-2023/"Операционные системы"/os-intro
amezhova@dk6n55 ~/work/study/2022-2023/Операционные системы/os-intro $ rm package.json
amezhova@dk6n55 ~/work/study/2022-2023/Операционные системы/os-intro $ echo os-intro > COURSE
```

Рис. 3.14: Настройка каталога

Отправка файлов на сервер:

```
amezhova@dk6n55 ~/work/study/2022-2023/Операционные системы/os-intro $ git add .
amezhova@dk6n55 ~/work/study/2022-2023/Операционные системы/os-intro $ git commit -am 'feat(main): make course structure'
```

Рис. 3.15: Отправка

```
amezhova@dk6n55 ~/work/study/2022-2023/Операционные системы/os-intro $ git push
Username for 'https://github.com': AlisaEzhova
Password for 'https://AlisaEzhova@github.com':
Перечисление объектов: 65, готово.
Подсчет объектов: 100% (65/65), готово.
При сжатии изменений используется до 6 потоков
Сжатие объектов: 100% (55/55), готово.
Запись объектов: 100% (65/65), 359.91 КиБ | 7.34 МиБ/с, готово.
Всего 65 (изменений 5), повторно использовано 26 (изменений 1), повторно использ
remote: Resolving deltas: 100% (5/5), done.
To https://github.com/AlisaEzhova/os-intro.git
+ a1f4292...316d3f5 master -> master (forced update)
```

Рис. 3.16: Отправка

4 Выводы

Я изучила идеологию и применила средства контроля версий, а также освоила умения по работе с git.

5 Контрольные вопросы

1. Что такое системы контроля версий (VCS) и для решения каких задач они предназначаются? Система контроля версий (VCS) — это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов. Для примеров в этой книге мы будем использовать исходные коды программ, но на самом деле под версионный контроль можно поместить файлы практически любого типа. Если вы графический или веб-дизайнер и хотели бы хранить каждую версию изображения или макета — а этого вам наверняка хочется — то пользоваться системой контроля версий будет очень мудрым решением. даёт возможность возвращать отдельные файлы к прежнему виду, возвращать к прежнему состоянию весь проект, просматривать происходящие со временем изменения, определять, кто последним вносил изменения во внезапно переставший работать модуль, кто и когда внёс в код какую-то ошибку, и многое другое. Вообще, если, пользуясь, вы всё испортите или потеряете файлы, всё можно будет легко восстановить. Вдобавок, накладные расходы за всё, что вы получаете, будут очень маленькими.
2. Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия. Хранилище-система, которая обеспечивает хранение всех существовавших вариантов файлов Commit-фиксация изменений История-список предыдущих ревизий Рабочая копия-копия другой ветки Команде commit можно передать сообщение, описывающее изменения в

реvisions. Она также записывает идентификатор пользователя, текущее время и временную зону, плюс список измененных файлов и их содержимого. Сообщение, описывающее изменения, определяется через опцию -m, или -message. Можно также вводить сообщения, состоящие из нескольких строк; в большинстве оболочек вы можете сделать это оставив открытую кавычку в конце строки. `commit -m` “добавлен первый файл.

3. Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида. Системы контроля версий. Централизованная система контроля версий Subversion и децентрализованная система контроля версий Mercurial. Существуют СКВ централизованные, в которых имеется один репозиторий, в который собираются изменения со всех рабочих копий разработчиков, и децентрализованные, когда репозиториев много, и они могут обмениваться изменениями между собой. Централизованные СКВ - репозиторий один. У каждого разработчика своя рабочая копия. Время от времени разработчик может затягивать к себе в рабочую копию новые изменения из репозитория, или проталкивать свои изменения из своей рабочей копии в репозиторий. Прочие особенности централизованных СКВ зависят от реализации.
4. Опишите действия с VCS при единоличной работе с хранилищем. Традиционные системы управления версиями используют централизованную модель, когда имеется единое хранилище документов, управляемое специальным сервером, который и выполняет большую часть функций по управлению версиями. Пользователь, работающий с документами, должен сначала получить нужную ему версию документа из хранилища; обычно создаётся локальная копия документа, т. н. «рабочая копия». Может быть получена последняя версия или любая из предыдущих, которая может быть выбрана по номеру версии или дате создания, иногда и по другим признакам. После того, как в документ внесены нужные изменения, но-

вая версия помещается в хранилище. В отличие от простого сохранения файла, предыдущая версия не стирается, а тоже остаётся в хранилище и может быть оттуда получена в любое время. Сервер может использовать т. н. дельта-компрессию — такой способ хранения документов, при котором сохраняются только изменения между последовательными версиями, что позволяет уменьшить объём хранимых данных. Поскольку обычно наиболее востребованной является последняя версия файла, система может при сохранении новой версии сохранять её целиком, заменяя в хранилище последнюю ранее сохранённую версию на разницу между этой и последней версией. Некоторые системы (например, ClearCase) поддерживают сохранение версий обоих видов: большинство версий сохраняется в виде дельт, но периодически (по специальной команде администратора) выполняется сохранение версий всех файлов в полном виде; такой подход обеспечивает максимально полное восстановление истории в случае повреждения репозитория.

5. Опишите порядок работы с общим хранилищем VCS. Традиционные системы управления версиями используют централизованную модель, когда имеется единое хранилище документов, управляемое специальным сервером, который и выполняет большую часть функций по управлению версиями. Пользователь, работающий с документами, должен сначала получить нужную ему версию документа из хранилища; обычно создаётся локальная копия документа, т. н. «рабочая копия». Может быть получена последняя версия или любая из предыдущих, которая может быть выбрана по номеру версии или дате создания, иногда и по другим признакам. После того, как в документ внесены нужные изменения, новая версия помещается в хранилище. В отличие от простого сохранения файла, предыдущая версия не стирается, а тоже остаётся в хранилище и может быть оттуда получена в любое время. Сервер может использовать т. н. дельт компрессию — такой способ хранения документов, при котором сохраняются только измене-

ния между последовательными версиями, что позволяет уменьшить объём хранимых данных. Поскольку обычно наиболее востребованной является последняя версия файла, система может при сохранении новой версии сохранять её целиком, заменяя в хранилище последнюю ранее сохранённую версию на разницу между этой и последней версией. Некоторые системы (например, ClearCase) поддерживают сохранение версий обоих видов: большинство версий сохраняется в виде дельт, но периодически (по специальной команде администратора) выполняется сохранение версий всех файлов в полном виде; такой подход обеспечивает максимально полное восстановление истории в случае повреждения репозитория.

6. Каковы основные задачи, решаемые инструментальным средством git? Устанавливает единственную новую команду, git. Все возможности предоставляются через подкоманды этой команды. Вы можете просмотреть краткую справку командой help. Некоторые идеи группируются по темам, используйте help topics для списка доступных тем. Одна из функций системы контроля версий — отслеживать кто сделал изменения. В распределенных системах для этого требуется идентифицировать каждого автора уникально в глобальном плане. Большинство людей уже имеют такой идентификатор: email адрес. Bazaar достаточно умен, чтобы автоматически создавать email адрес из текущего имени и адреса хоста. Основные задачи: создание ветки, размещение веток, просмотр изменений, фиксация изменений, сообщение из текстового редактора, выборочная фиксация, удаление зафиксированных изменений, игнорирование файлов, просмотр истории, статистика ветки, контроль файлов и каталогов, ветвление, объединение веток, публикация ветки.
7. Назовите и дайте краткую характеристику командам git. Обновление рабочей копии По мере внесения изменений в проект рабочая копия на компьютере разработчика стареет, расхождение её с основной версией проекта

увеличивается. Это повышает риск возникновения конфликтных изменений (см. ниже). Поэтому удобно поддерживать рабочую копию в состоянии, максимально близком к текущей основной версии, для чего разработчик выполняет операцию обновления рабочей копии (update) насколько возможно часто (реальная частота обновлений определяется частотой внесения изменений, зависящей от активности разработки и числа разработчиков, а также временем, затрачиваемым на каждое обновление — если оно велико, разработчик вынужден ограничивать частоту обновлений, чтобы не терять время). Модификация проекта Разработчик модифицирует проект, изменяя входящие в него файлы в рабочей копии в соответствии с проектным заданием. Эта работа производится локально и не требует обращений к серверу VCS. Фиксация изменений Завершив очередной этап работы над заданием, разработчик фиксирует (commit) свои изменения, передавая их на сервер (либо в основную ветвь, если работа над заданием полностью завершена, либо в отдельную ветвь разработки данного задания). VCS может требовать от разработчика перед фиксацией обязательно выполнить обновление рабочей копии. При наличии в системе поддержки отложенных изменений (shelving) изменения могут быть переданы на сервер без фиксации. Если утверждённая политика работы в VCS это позволяет, то фиксация изменений может проводиться не ежедневно, а только по завершении работы над заданием; в этом случае до завершения работы все связанные с заданием изменения сохраняются только в локальной рабочей копии разработчика.

8. Приведите примеры использования при работе с локальным и удалённым репозиториями. Мы создаем новую ветку выполнив `git init` в уже созданном каталоге: `% mkdir tutorial % cd tutorial % ls -a ./ ../ % pwd /home/mbp/work/bzr.test/tutorial % % git init % ls -aF ./ ../ .git/ %` Мы обычно обращаемся к веткам на нашем компьютере просто передав имя каталога содержащего ветку. bzr также поддерживает доступ к веткам

через http и sftp, например: `git log http://bazaar-vcs.org git // git.dev/ git log sftp://bazaarvcs.org/bzr/bzr.dev/` Установив для git плагины можно также осуществлять доступ к веткам с использованием rsync. Команда status показывает какие изменения были сделаны в рабочем каталоге с момента последней ревизии: `% git status modified: foo bzr status` скрывает неинтересные файлы, которые либо не менялись, либо игнорируются. Также команде status могут быть переданы необязательные имена файлов, или каталогов для проверки. Команда diff показывает изменения в тексте файлов в стандартном формате diff. Вывод этой команды может быть передан другим командам, таким как "patch", "diffstat", "filterdiff" и "colordiff": `% git diff === added file 'hello.txt' — hello.txt 1970-01-01 00:00:00 +0000 +++ hello.txt 2005-10-18 14:23:29 +00006.2. Указания к лабораторной работе 75 @@ -0,0 +1,1 @@ +hello world` Команде commit можно передать сообщение описывающее изменения в ревизии. Она также записывает идентификатор пользователя, текущее время и временную зону, плюс список измененных файлов и их содержимого. `git commit -m "добавлен первый файл"` Если вы передадите список имен файлов, или каталогов после команды commit, то будут зафиксированы только изменения для переданных объектов. Например: `bzr commit -m "исправления документации" commit.py` Если вы сделали какие-либо изменения и не хотите оставлять их, используйте команду revert, что бы вернуться к состоянию предыдущей ревизии. Многие деревья с исходным кодом содержат файлы которые не нужно хранить под контролем версий, например резервные файлы текстового редактора, объектные файлы и собранные программы. Вы можете просто не добавлять их, но они всегда будут обнаруживаться как неизвестные. Вы также можете сказать git игнорировать их добавив их в файл .ignore в корне рабочего дерева. Для получения списка файлов которые игнорируются и соответствующих им шаблонов используйте команду `ignored: % ignored config.h ./config.h configure.in~ *~ log` Команда `bzr log` показывает список

предыдущих ревизий. Команда `log -forward` делает тоже самое, но в хронологическом порядке, показывая более поздние ревизии в конце может контролировать файлы и каталоги, отслеживая переименования и упрощая их последующее объединение: `% mkdir src % echo 'int main() {}' > src/simple.c % add src added src added src/simple.c % status added: src/src/simple.c bzip remove удаляет файл из под контроля версий, но может и не удалять рабочую копию файла2. Это удобно, когда вы добавили не тот файл, или решили, что файл на самом деле не должен быть под контролем версий. % rm -r src % remove -v hello.txt ? hello.txt % status removed: hello.txt src/ src/simple.c unknown: hello.txt` Часто вместо того что бы начинать свой собственный проект, выхотите предложить изменения для уже готового проекта. Что бы сделать это вам нужно получить копию готовой ветки. Canonical The Bazaar Team bazaar@lists.canonical.com Bazaar version control system: easily manage source code on Windows, Ubuntu, GNU/Linux, and Mac OS X Так как эта копия может быть потенциальной новой веткой. Если две ветки разошлись (обе имеют уникальные изменения) тогда `merge` — это подходящая команда для использования. Объединение автоматически вычислит изменения, которые существуют на объединяемой ветке и отсутствуют в локальной ветке и попытается объединить их с локальной веткой. `git merge URL`.

9. Что такое и зачем могут быть нужны ветви (branches)? Часто вместо того что бы начинать свой собственный проект, вы хотите предложить изменения для уже готового проекта. Что бы сделать это вам нужно получить копию готовой ветки. Так как эта копия может быть потенциальной новой веткой эта команда называется `branch`: Управление версиями `git branch cd git.dev` Эта команда копирует полную историю ветки и после этого вы можете делать все операции с ней локально: просматривать журнал, создавать и объединять другие ветки.

10. Как и зачем можно игнорировать некоторые файлы при commit? Нет проблем если шаблон для игнорирования подходит для файла под контролем версий, или вы добавили файл, который игнорируется. Шаблоны не имеют никакого эффекта на файлы под контролем версий, они только определяют показывающиеся неизвестные файлы, или просто игнорируются. Файл `git.rignore` обычно должен быть под контролем версий, что бы новые копии ветки видели такие же шаблоны: `git add . gitignore git commit -m "Добавлены шаблоны для игнорирования"`. Многие деревья с исходным кодом содержат файлы, которые не нужно хранить под контролем версий, например, резервные файлы текстового редактора, объектные файлы и собранные программы. Вы можете просто не добавлять их, но они всегда будут обнаруживаться как неизвестные. Вы также можете сказать `bzr` игнорировать их добавив их в файл в корне рабочего дерева. Этот файл содержит список шаблонов файлов, по одному в каждой строчке. Обычное содержимое может быть таким: `.o ~ .tmp .py [со]` Если шаблон содержит слеш, то он будет сопоставлен с полным путем начиная от корня рабочего дерева; иначе он сопоставляется только с именем файла. Таким образом пример выше игнорирует файлы с расширением `.o` во всех подкаталогах, но пример ниже игнорирует только `config.h` в корне рабочего дерева и HTML файлы в каталоге `doc/`: `./config.h doc/.html` Для получения списка файлов которые игнорируются и соответствующих им шаблонов используйте команду `git ignored` :
- ```
$ git ignored config.h ./config.h configure.in ~ ~ $
```